МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

РОЗРОБКА СИСТЕМИ ПЕРЕДБАЧЕНННЯ МАЙБУТНІХ ВІДСУТНОСТЕЙ ПРАЦІВНИКІВ

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:
студент 6 курсу, 607 групи
Кусяк Володимир Іванович
Керівник:
кандидат фізико-математичних наук,
доцент Олександр Матвій

До захисту допущено
на засіданні кафедри
протокол № 9 від 5 грудня 2023 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2023

АНОТАЦІЯ

Дана робота присвячена створенню інструменту для аналізу відпусток на підприємствах.

В даній роботі було створено програмне забезпечення за допомогою мови програмування Java та супутніх їх інструментів таких як: Spring Boot, Maven, H2, PostgreSQL, Deep Java Library (DJL).

Ключові слова: **Java, Spring Boot, DJL, AI, мікросервіс, навчання, передбачення, відпустка, планування.**

ABSTRACT

This work is dedicated to the development of a tool for analyzing vacations in enterprises.

In this project, software has been created using the Java programming language and related tools such as Spring Boot, Maven, H2, PostgreSQL, and the Deep Java Library (DJL).

Key words: **Java, Spring Boot, DJL, AI, microservice, learning, prediction, vacation, planning.**

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____ В. І. Кусяк
(підпис)

ЗМІСТ

ВСТУП

Актуальність роботи. Планування виконання робіт на підприємствах відіграє ключову роль в їх стабільному функціонуванні. Зважаючи на той фактор, що основним ресурсом підприємства, як правило, є його працівники, враховувати їх наявність або відсутність при планування певних видів робіт стає непростим завданням. На сьогоднішній день є чимало різних причин, чому людина не може з'явитись на робочому місці, серед них: хвороби, канікули, сезонні справи, тощо. Такі події часто залежать від різноманітних загальних факторів, таких як планові відпустки, сезонні хвороби, тощо, та персональних: сімейні обставини, хронічні хвороби, тощо.

Мета. Метою кваліфікаційної роботи є розробка інструменту для аналізу відсутностей працівника в минулому та можливостей їх передбачення в майбутньому. Даний інструмент надає можливість проводити краще планування виконання робіт на підприємствах.

Завдання роботи. Завданням кваліфікаційної роботи є розробка програмного рішення для аналізу та передбачення відсутностей працівників на підприємставах. Вимогами до застосунку є можливість його використання з іншими застосунками, масштабованість, зручне використання REST API.

Під час виконання кваліфікаційної роботи було розв'язано такі задачі:

1. дослідження та аналіз програмних рішень подібної тематики, порівняння їх функціональних можливостей, виділення переваг та недоліків;
2. розробка мікросервісу для створення, редагування, перегляду подій за допомогою технологій Java, Spring Boot, DJL;
3. тестування розробленого мікросервісу.

Створений мікросервіс складається тільки з серверної частини, для збереження використовується такі СКБД як H2 та PostgreSQL. Під час роботи використовувались наступні технології Java, Spring Boot, DJL, Docker.

Об’єкт дослідження. Система планування виконання робіт на підприємствах, зокрема проблеми, пов'язані з відсутністю працівників. Дослідження спрямоване на розробку інструменту для аналізу минулих відсутностей працівників та передбачення можливих відсутностей в майбутньому. Об'єктом також є розроблений мікросервіс, що включає в себе серверну частину та використовує технології Java, Spring Boot, DJL, Docker, а також бази даних H2 та PostgreSQL. У рамках дослідження вирішувалися завдання дослідження та аналізу аналогічних програмних рішень, розробки мікросервісу та його тестування.

Практичне застосування. Система може допомагати підприємствам аналізувати минулі відсутності працівників, враховуючи різноманітні фактори, такі як хвороби, канікули, сімейні обставини. Це дозволить краще планувати відпустки, а також адаптувати робочий графік до можливих відсутностей.

Розділ 1. Огляд аналогів за тематикою кваліфікаційної роботи

1.1 Проблематика

На сьогоднішній день існує чимало програмних рішень для ефективного планування роботи, а також технологій та підходів для визначення ефективності праці людей. Відомості про те, як працівники виконували свою роботу в минулому, беруться до уваги та впливають на планування в майбутньому.

Однією із таких технологій є Scrum - це популярна методологія розробки програмного забезпечення, яка входить в сім'ю Agile підходів. Основна ідея Scrum полягає в тому, щоб створити фреймворк для ефективного управління процесом розробки та забезпечення максимальної вартості продукту для клієнта. Вона дозволяє гнучко відповідати на зміни вимог, працювати над призначеними завданнями в ітераціях та забезпечувати високий рівень співпраці в команді.

Одним із найважливіших метрик із Scrum є Velocity - метрика, яка використовується в методології Scrum для вимірювання продуктивності команди під час розробки. Velocity допомагає командам Scrum прогнозувати, скільки роботи вони можуть завершити протягом певного періоду. Коли команда працює кілька періодів, Velocity дозволяє здійснювати більш точні прогнози для планування роботи в майбутньому.

Однак важливо враховувати, що Velocity не є абсолютним показником продуктивності чи якості. Він є інструментом для команди, який дозволяє їй краще розуміти свою ефективність та з часом вдосконалювати власну продуктивність.

Хоча методологія Scrum спочатку розроблялася з метою управління проектами у сфері розробки програмного забезпечення, вона успішно застосовується в різних галузях та компаніях поза межами IT-індустрії. Застосування Scrum може бути важливим для будь-якої галузі, де проектна

робота вимагає гнучкості, ефективного спілкування та відкритості до змін. Ось деякі приклади сфер діяльності, де використовується Scrum:

Маркетинг: Команди маркетингу можуть використовувати Scrum для планування та виконання кампаній, розробки контенту, організації заходів та інших ініціатив.

Консалтинг: В галузі консалтингу Scrum може допомагати в керуванні консалтинговими проектами, управлінні замовленнями та взаємодії з клієнтами.

Освіта: Застосування Scrum у сфері освіти може сприяти ефективному управлінню проектами, розробці навчальних матеріалів та організації навчальних заходів.

Здоров'я та медицина: Команди в галузі медицини можуть використовувати Scrum для впровадження інновацій, розробки нових методів лікування, та вдосконалення процесів медичного обслуговування.

Виробництво та виробничі процеси: Scrum може допомагати в управлінні проектами виробництва, розробці нових продуктів, а також у вирішенні проблем та удосконаленні виробничих процесів.

Хоча Scrum походить від розробки програмного забезпечення, його принципи можуть бути успішно адаптовані для різних галузей та видів проектів. Основний фокус на комунікації, гнучкості та визначенні пріоритетів може призвести до покращення ефективності та результативності в будь-якому середовищі проектної діяльності.

Врахування відпусток у даній методології є важливим аспектом управління та планування робіт, оскільки вони можуть вплинути на продуктивність та обсяг роботи, який може бути виконаний протягом певного періоду. Відпустки повинні враховуватися як складова частина планування.

Незаплановані відпустки можуть мати певний вплив на працівників та процес виконання робіт. Цей вплив залежить від ряду факторів, таких як тривалість відсутності, важливість ролі особи, яка відсутня, та загальна організація команди.

Ось деякі можливі впливи

Зменшення продуктивності: Якщо ключовий працівник відсутній, це може призвести до зниження продуктивності, тобто обсягу роботи, який команда може виконати в запланований період.

Затримка виконання робіт: Незаплановані відпустки можуть вплинути на вчасне виконання завдань, особливо якщо особа відповідальна за певну частину роботи.

Необхідність перерозподілу робіт: Коли працівник відсутній, інші члени команди можуть бути змушені переймати його обов'язки, що може вимагати часу на підготовку та вивчення нового матеріалу.

Відновлення після відпустки: Після повернення з відпустки співробітник може потребувати деякого часу на адаптацію та відновлення продуктивності.

Окрім вище перерахованих негативних впливів на роботу команди, варто враховувати той фактор, що деякі працівники можуть виконувати вкрай важливі обов'язки для функціонування підприємства.Тому, якщо важливі працівники часто відсутні через відпустки, це може вимагати докладного планування та управління, щоб забезпечити ефективність роботи.

Щоб зменшити негативний вплив незапланованих відпусток, важливо планувати заздалегідь, враховуючи можливі відсутності та визначаючи заміни членам команди за необхідності

Незаплановані відпустки можуть мати різні причини, іноді вони можуть бути непередбаченими і їх важко контролювати, наприклад:

Невідкладні особисті обставини: Неочікувані події, такі як сімейні екстрені ситуації, проблеми зі здоров'ям, нещасні випадки або інші непередбачені обставини, які можуть призвести до того, що працівник вирішить взяти відпустку.

Спонтанні відпустки: Іноді працівники можуть вирішити взяти відпустку в рамках своєї відпусткової квоти, або без неї, через спонтанний відпочинок або несподівані обставини.

Стан здоров’я: Хвороба або тимчасове погіршення здоров’я може змусити працівника взяти неочікувану відпустку.

Екстрені справи: Працівники можуть зіткнутися з ситуацією, що вимагають їх негайної уваги, наприклад, правові або адміністративні питання, які неможливо вирішити в неробочий час.

Конфлікти на роботі: Ситуації конфлікту або напруженості на роботі можуть викликати бажання працівника взяти кілька днів відпустки для відновлення емоційного стану.

Завчасне планування: Іноді працівники можуть вирішити взяти відпустку заздалегідь, але не повідомляти про це завчасно.

Специфічні ситуації на роботі: Непередбачені технічні або організаційні проблеми, можуть вимагати тимчасової відсутності працівників для їх вирішення.

Незалежно від причин, непередбачені відпустки можуть виникнути з різних факторів, і важливо забезпечити певний резерв у планах робіт, щоб враховувати можливі невідповідності у виробничому процесі.

Інерційні або сезонні характеристики незапланованих відпусток можуть виникнути з ряду факторів, які є певним чином пов'язані з природними циклами, сезонністю або системними тенденціями. Ось декілька прикладів:

Сезонні захворювання: Деякі хвороби, такі як грип чи застуда, можуть бути більш поширеними в певні сезони, зазвичай восени та взимку. Це може призвести до сезонного збільшення відпусток через захворювання.

Відпустки в школах: У багатьох регіонах літні місяці є періодом канікул для школярів, і багато батьків вибирають цей час для відпусток з дітьми. Це може призводити до більшої кількості незапланованих відпусток влітку.

Пікові обсяги роботи: Деякі галузі або види бізнесу можуть мати пікові періоди, коли обсяг роботи збільшується, наприклад, через святкові сезони, різдвяні канікули або інші події. Це може призвести до непередбачених відпусток для працівників.

Аграрні або сезонні галузі: У сільському господарстві або інших сезонно залежних галузях відпустки можуть бути пов'язані із сільськогосподарськими циклами, збором врожаю чи іншими сезонними обставинами.

Події та конференції: У деяких галузях певні події, які відбуваються щороку в один і той же період, можуть призвести до тимчасової відсутності працівників на роботі.

Цикли ринку: В окремих галузях цикли ринку можуть впливати на обсяги роботи та призводити до несподіваних відпусток або, навпаки, періодів інтенсивної роботи.

Ці фактори можуть впливати на незаплановані відпустки, враховуючи сезонні або інерційні тенденції в робочому середовищі.

1.2 Огляд можливостей програмних рішень для планування робіт

Існує багато програмних рішень для планування навантаження роботи, і вони можуть варіюватися в залежності від конкретних потреб та характеру бізнесу.

Ось деякі з популярних програмних рішень у цій області.

Microsoft Project [1]: Це один з найвідоміших інструментів для управління проектами. Microsoft Project дозволяє створювати графіки, визначати завдання, призначати ресурси та виконувати інші функції для планування навантаження.

Microsoft Project дозволяє розподіляти завдання по ресурсах (включаючи людей) і визначати трудові об'єми. Можна враховувати календарі ресурсів,

включаючи відпустки. Однак він може бути менш гнучким для адаптації до непередбачених змін порівняно з деякими більш гнучкими інструментами.

Smartsheet [2]: Це хмарне програмне забезпечення для управління проектами та завданнями, яке включає у себе електронні таблиці та інші інструменти для спільної роботи.

Smartsheet дозволяє створювати графіки ганта, розподіляти завдання та керувати ресурсами. Ви можете враховувати та планувати відпустки у Smartsheet, але рівень деталізації може залежати від конкретних вимог проекту.

Trello [3]: Це інструмент для управління завданнями, який базується на концепції "дошки" з картками, які представляють завдання. Це може бути корисним для планування завдань та роботи в команді.

Trello надає "дошки" та картки для організації завдань. Врахування відпусток може викликати певні труднощі, оскільки Trello не надає вбудованих засобів для керування графіками та ресурсами на тому рівні, як, наприклад, Microsoft Project.

Asana [4]: Це програмне забезпечення для управління завданнями та проектами, яке надає засоби для планування навантаження та управління термінами.

Asana дозволяє створювати проекти, завдання та календарі для планування. Ви можете використовувати календарі для відстеження та планування відпусток, але розподіл роботи може вимагати деякого додаткового навантаження на працівника, який розподілятиме обов’язки.

Jira [5]: Це інструмент, розроблений для розробки програмного забезпечення, але його можна використовувати для управління завданнями та планування роботи в інших галузях.

Jira, особливо в поєднанні з плагінами, може надавати функціонал для управління ресурсами та графіками. Можливість врахування відпусток залежить від конфігурації та використання плагінів.

Monday.com [6]: Це інструмент для управління роботою та проектами, який надає можливості для планування навантаження, створення графіків та спільної роботи.

Monday.com дозволяє створювати таблиці, графіки та відстежувати завдання. Можна використовувати графіки для планування відпусток, але розподіл ресурсів може вимагати вдосконалення через зовнішні інтеграції.

Wrike [7]: Це хмарний сервіс для управління роботою та проектами, який дозволяє створювати завдання, призначати ресурси та визначати терміни.

Wrike надає можливості для створення графіків, розподілу завдань та керування ресурсами. Відпустки можна враховувати в плануванні, але це вимагає налагодження.

Отже, враховуючи вищевикладене, можна дійти до висновку, що кожен з наведених інструментів має свої унікальні можливості та обмеження щодо врахування непередбачених відпусток.

1.3 Постановка задачі

Штучний інтелект (ШІ) може вивчати та прогнозувати значення, які мають інерційні або сезонні характеристики. Для цього використовуються різні методи машинного навчання та аналізу даних.

Моделі машинного навчання, можуть використовуватися для аналізу та прогнозування даних, враховуючи їхні інерційні або сезонні залежності.

Багато моделей машинного навчання можуть автоматично виявляти фактори та особливості, які впливають на значення цільової змінної без явного програмування. Цей процес називається “витягуванням ознак” або “автоматичним вибором ознак”. Існує кілька методів, які допомагають моделям визначати важливість та вплив ознак.

Завданням даної роботи є розробка програмного рішення, яке має наступні властивості:

- легко інтегрується з сторонніми додатками;
- дозволяє отримати доступ за допомогою мережі Інтернет
- масштабується при навантаженні;
- проводить аналітику та прогнозування незапланованих відсутностей працівників;
- легко підтримується працездатність коду в майбутньому

Розділ 2. Проектування та виконання робіт

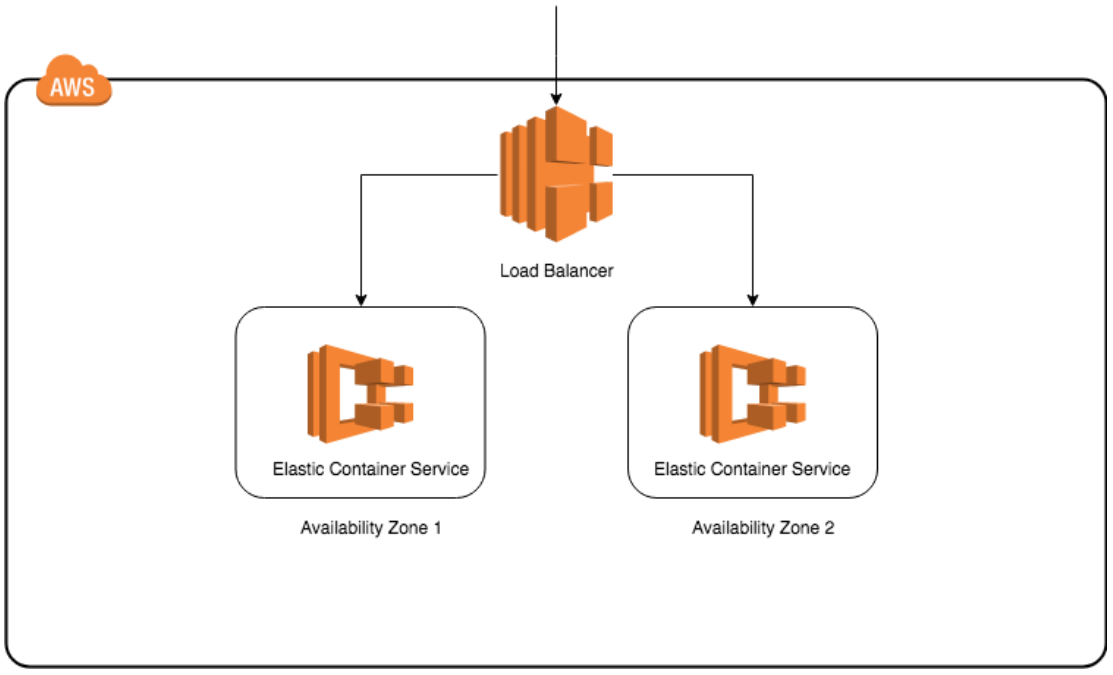
2.1 Розробка архітектури рішення

2.1.1 Спосіб розгортання та масштабованість

Розробка API (інтерфейсу програмування застосунків) та надання доступу до нього для сторонніх програм є важливим етапом у створенні масштабованого та взаємодіючого програмного забезпечення.

Оскільки навантаження на API та кількість сторонніх програм є невизначеними, потрібно мати змогу запускати декілька екземплярів застосунку, та розподіляти навантаження між ними.

Для вирішення цієї задачі було заплановано використання технології AWS Load Balancing [8].



Мал 2.1 – Діаграма зв’язку Load Balancer та Elastic Container Service

Дана технологія дозволяє автоматично запускати та розподіляти навантаження між декількома процесами, що забезпечує оптимальну швидкодію, та збереження коштів за оплату оренди використаних серверів (мал 2.1). Тобто при мінімальному навантаженні, буде використано мінімальну кількість задіяних ресурсів на сервері.

Для автоматичного масштабування процесів використовуватиметься Elastic Container Service (ECS). Він дозволяє запускати Docker Image з потрібним зразком програмного забезпечення. Фактично ECS - це технологія для запуску динамічного сервера.

Docker Image - це невеликий за об'ємом пам'яті та автономний пакет, який містить програмний код, середовище виконання, бібліотеки, налаштування та інші необхідні компоненти для функціонування конкретного застосунку чи сервісу. Docker використовує контейнеризацію для ізоляції та розгортання програмного забезпечення в стандартизованому середовищі.

2.1.2 Розробка спілкування з сторонніми додатками

Для спілкування із сторонніми додатками потрібно виконати 3 задачі:

- розробка API
- розробка захисту API
- розробка документації

Розробка API

Однією з вимог при проектуванні програмного рішення, яке розробляється в межах даної роботи є доступ до нього через мережу Інтернет, у зв'язку з чим, в процесі його розробки буде використовуватись REST API.

REST (Representational State Transfer) - це архітектурний стиль для створення веб-служб та API (інтерфейсів програмування застосунків).

Використання REST API має кілька переваг, які роблять його популярним для розробки веб-додатків та інтеграції систем, наприклад:

Простота та легкість розуміння. REST API використовує стандартні HTTP методи (GET, POST, PUT, DELETE), що робить його легким для розуміння та використання. Структура URL-адрес та використання HTTP методів є інтуїтивно зрозумілими для розробників.

Незалежність від мов та платформ. REST є мовонезалежним та може бути використаний майже будь-де. Клієнти та сервери можуть бути реалізовані на будь-якій мові програмування та працювати на будь-яких платформах.

Легкість інтеграції. REST API легко інтегрується з іншими системами, оскільки використовує стандартні протоколи (HTTP/HTTPS). Запити та відповіді передаються через HTTP, що дозволяє взаємодіяти з API з різних платформ та з різних пристроїв.

Скальованість. REST дозволяє скалькувати систему легко, оскільки кожен ресурс може бути оброблений та масштабований незалежно. *Стандартизація та уніфікація*. Використання стандартних HTTP методів та кодів стану робить REST API стандартизованим та уніфікованим. JSON часто використовується як формат обміну даними, що робить структури даних більш зрозумілими та уніфікованими.

Підтримка кешування. REST API може використовувати кешування для збереження копій ресурсів на клієнтському боці, що зменшує навантаження на сервер та покращує продуктивність.

Широке розповсюдження та підтримка. REST є однією з найпопулярніших архітектур для веб-служб, і вона підтримується багатьма технологічними стеками та інструментами.

Для реалізації було обрано фреймворк Spring-Boot [9]. Оскільки Spring Boot є фреймворком для розробки веб-додатків на мові Java, який спрощує та

прискорює процес розробки. За допомогою Spring Boot можна легко створювати RESTful API, яке використовує принципи архітектури REST.

Spring Boot надає можливість використовувати такі інструменти для розробки RESTful API як:

- Валідація та обробка помилок. Використання анотацій для валідації вхідних даних, таких як @Valid та @NotBlank. Обробка помилок за допомогою анотації @ExceptionHandler для специфікації методів, які обробляють конкретні типи помилок.
- Робота з базою даних. Використання Spring Data JPA для спрощення взаємодії з базою даних. Анотування класів та методів для визначення сутностей, запитів та транзакцій.
- Аутентифікація та авторизація. Використання Spring Security для налаштування аутентифікації та авторизації. Визначення ролей та конфігурація фільтрів безпеки.
- Інтеграція з Swagger[10] для Документації. Використання бібліотеки Springfox Swagger для автоматичної генерації документації API. Додавання анотацій @Api та @ApiOperation для докладної документації кожної точки доступу.

Розробка захисту API

Задачі захисту API буде виконувати Spring Security. Це потужний інструмент для забезпечення безпеки в Java-додатках, побудованих на основі Spring. Він дозволяє забезпечити аутентифікацію, авторизацію, захист від атак та інші функції безпеки в додатку. До основних концепцій та можливостей Spring Security можна віднести:

Аутентифікацію. Spring Security дозволяє легко інтегрувати різні методи аутентифікації, включаючи базову аутентифікацію, аутентифікацію на основі

форми, аутентифікацію через сторонні системи (LDAP, OAuth, OpenID Connect і інші).

Авторизацію. Можливість визначати ролі користувачів та встановлювати правила доступу для різних ресурсів додатка.

Фільтрацію та Ланцюжки Фільтрів. Spring Security використовує фільтри для обробки різних аспектів безпеки, таких як перехоплення аутентифікаційних спроб, захист від CSRF (Cross-Site Request Forgery), фільтрація запитів і т.д.

Управління Сесіями та Безпека Сесій. Контроль за сесіями користувачів для забезпечення безпеки. Можливість використовувати JSON Web Tokens (JWT) для аутентифікації та безпеки сесій.

Розширення для Захисту від Атак. Захист від атак Cross-Site Request Forgery (CSRF), атак вривання та інших загроз безпеці. Можливість використовувати анотації та аспекти для захисту від атак на рівні методів.

Отже, підсумовуючи все викладене вище, можна дійти до висновку, що Spring Security дозволяє розробникам зосередитися на функціональності додатків, надаючи при цьому потужний інструментарій для роботи з безпекою.

Розробка документації API

Для документації було обрано технологію Swagger, оскільки вона є однією із найбільш розповсюджених форматів документації RESTful API. Окрім того, для пришвидшення складання документації проекту було використано бібліотеку Spring Doc, яка надає можливості генерації автоматичної документації для веб-служб Spring Boot на основі специфікації OpenAPI (відомої як Swagger). SpringDoc дозволяє автоматично створювати інтерактивну документацію для RESTful API, використовуючи різні анотації та підходи. Використання SpringDoc спрощує процес створення та оновлення документації для API, забезпечуючи при цьому високий рівень гнучкості та керованості.

Також він дозволяє скористатись згенерованим веб інтрефейсом документації, відомим як Swagger UI, що являє собою інтерактивний веб-інтерфейс, який надає можливість взаємодії з документацією API, створеною за допомогою Swagger (OpenAPI). Вбудований Swagger UI є частиною інструментарію Swagger, і він може автоматично генерувати інтерфейс на основі визначень API.

Автоматична документація Spring Doc спрощує процес створення та оновлення документації для API, забезпечуючи високий рівень гнучкості та можливостей налаштування.

2.1.3 Розробка дизайну рішення

Аутенфікація

Розробку дизайну рішення було розпочато з системи аутентифікації та авторизації, для яких використовується спосіб авторизації - JWT.

Аутентифікація JWT — це механізм аутентифікації без стану (stateless) на основі маркерів (токенів) JWT. Його широко використовують як stateless сеанс на стороні клієнта, це означає, що серверу не потрібно повністю покладатися на сховище даних, або базу даних, для збереження інформації про сеанс. В способі JWT можна використовувати власне шифрування, однак вбудовані методи зазвичай закодовані та підписані, що дозволяє покластися на їх надійність.

Особливістю використання JWT є те, що токени мають термін дії. При успішній автентифікації сервер повертає клієнту 2 токени:

- для доступу, який використовується при кожному наступному запиті на API, та має короткий час життя;
- для оновлення, який зберігається як в клієнта, так і на сервері. Він використовується для отримання нової пари токенів для доступу та оновлення. Головна особливість в захисті від викрадення токенів полягає в тому, що

використати токен оновлення можна тільки 1 раз, тобто коли шахрай спробує його використати, він скоріш вже буде використаним.

Авторизація

Для користувачів розроблюваного програмного рішення передбачено 2 ролі:

- ROOT - призначена для створення, редагування та видалення користувачів із роллю SESSION_USER. Також роль ROOT спрощує організацію та написання JUnit тестів, на етапі перевірки коректності функціонування системи;
- SESSION_USER - буде використовуватись користувачами програмного рішення для роботи з розроблюваним мікросервісом.

Накопичення та збереження даних для аналітики

Предметом аналітики будуть дані, про відсутність працівників на робочому місці в минулому. До цих даних входять початок та кінець періоду, кількість запланованих та незапланованих відсутностей

Незаплановані відсутності - це такі відсутності, про які керівництво було повідомлене менше ніж за 2 тижні до їх настання.

Зазначені вище відомості будуть надаватись стороннім програмним рішенням, тому обмеження в 2 тижні є умовністю, яка використовуватиметься для тестування та подальшої інтеграції. Стороннє програмне рішення може самостійно встановити своє значення для цієї умовності.

З метою реалізації множинної аналітики всі періоди будуть додаватися в сесії. Користувач може створювати декілька сесій, в яких будуть міститись періоди, таким чином ШІ буде можливість вивчати кожну сесію незалежно

один від одної. Це рішення дозволяє уникнути впливу даних від різних користувачів та сесій між собою.

Структура Бази Даних

Мал. 2.2 Структура бази даних

З метою реалізації функціонування мікросервісу була розроблена мінімальна структура бази даних (мал 2.2), якої достатньо, щоб задовольнити вимоги програмного рішення.

2.1.4 Аналітичний модуль

Аналітична частина мікросервісу використовуватиме ШІ на базі Deep Java Library. Використання Deep Java Library (DJL) є обгрунтованим з кількох причин, особливо для розробників, які працюють з мовою програмування Java та бажають впровадити глибоке навчання у свої проекти. До основних переваг DJL варто віднести наступні:

Зручність для Java-розробників. DJL створений з урахуванням специфіки Java, що робить його зручним та легким у використанні для розробників, знайомих з цією мовою програмування.

*Підтримка різних фреймворків глибокого навчанн*я. DJL підтримує кілька популярних фреймворків глибокого навчання, таких як TensorFlow, PyTorch, MXNet та інші. Це дозволяє обирати фреймворк в залежності від потреб та вже існуючих знань.

Простота інтеграції. DJL добре інтегрується з Java-проектами та може бути використаний в різних областях, включаючи веб-розробку, мобільні додатки та інші. Він забезпечує зручність роботи з глибоким навчанням в середовищі Java.

Підтримка різних типів моделей. DJL підтримує різні типи моделей глибокого навчання, що робить його використання варіативним в залежності від поставленого завдання.

*Підтримка апаратного прискоренн*я. DJL може використовувати різні типи апаратного прискорення, такі як CPU, GPU або TPU, що дозволяє досягати високої продуктивності в обчисленнях.

Активна спільнота та розвиток. Як відкритий проект, DJL має активну спільноту розробників та продовжує розвиватися, що гарантує підтримку та оновлення.

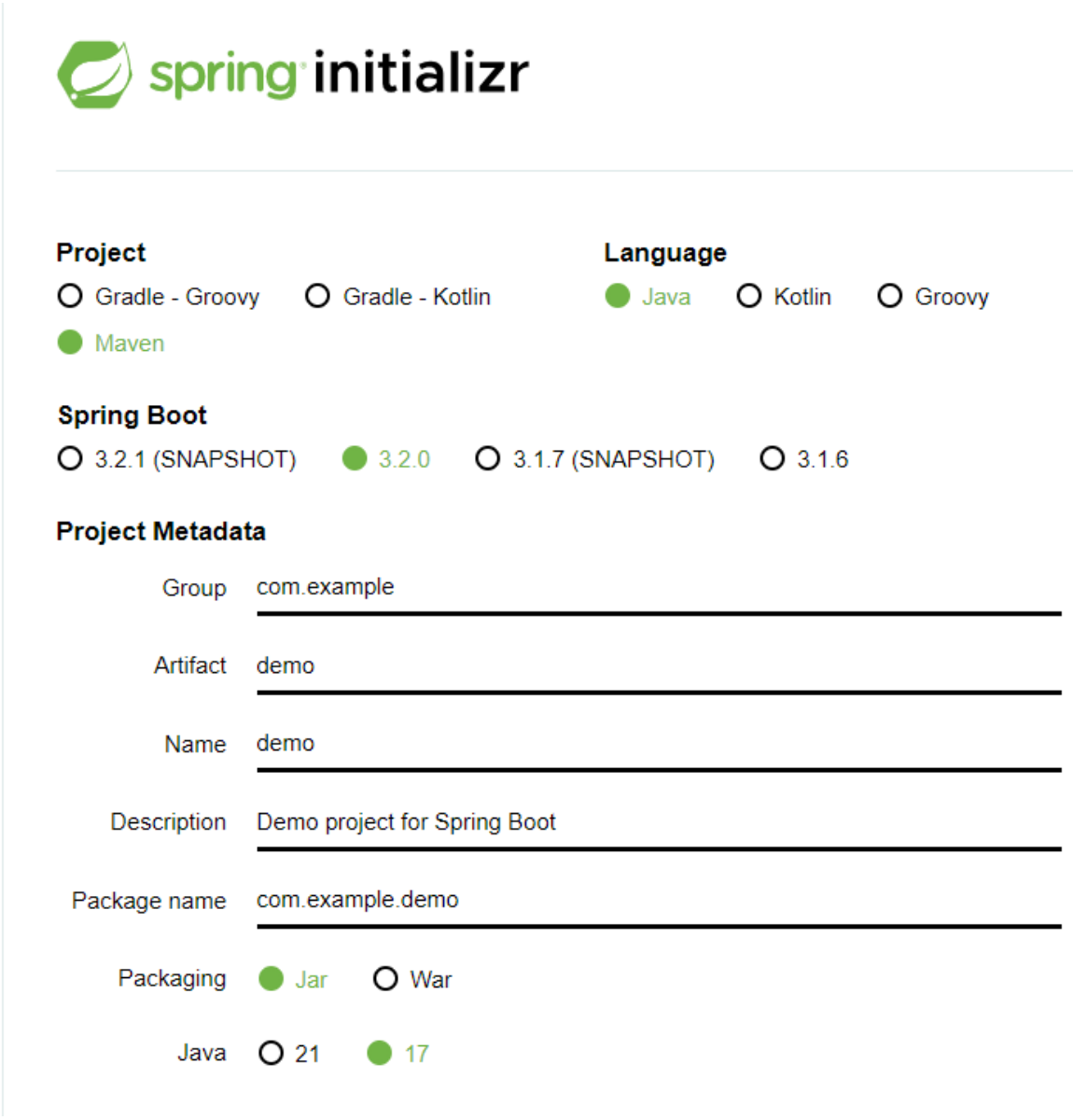
Отже, підсумовуючи вищевикладене, DJL - є оптимальним вибором для Java-розробників, які шукають зручний та ефективний спосіб використання глибокого навчання у своїх проектах.

2.2 Реалізація рішення

2.2.1 Створення проекту та тестової інфраструктури

За основу, для розроблюваного програмного рішення було згенеровано базовий Spring Boot проект, в сервісі “Spring Initializer” (мал 2.3), в ході генерування якого було обрано наступні додаткові опції:

- Language - Java
- Project - Maven
- Packaging - Jar
- Java - 17
- Spring Web
- Spring Security
- Spring Data JPA
- Flyway Migration
- H2 Database
- PostgreSQL Driver
- Validation

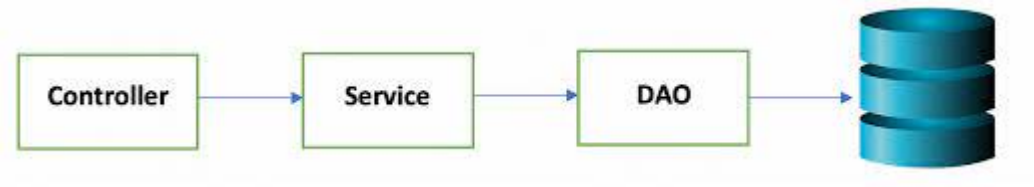


Мал 2.3 – Інтерфейс генерації Spring проекту

Наступним кроком було імплементовано логічні методи для роботи користувача в системі, в ході реалізації яких реалізовано наступний функціонал:

- можливість спілкування з базою даних
- entity User та його ролі
- рівні DAO та Service
- інструментарій для спрощення написання тестів
- створення автоматизованих тестів, з метою перевірки коректності функціонування системи.

Принцип функціонування окремих складових розроблюваної системи між собою, наведено на малюнку 2.4 в якості спрощеної системи реалізації рівнів відповідності.



Мал 2.4 – Спрощена схема реацізації рівнів відповідальності

Перша імплементація стала основою для реалізації наступних CRUD (Create, Read, Update, Delete) систем. Однак наявність відомостей про користувача в базі даних є недостатньою для швидкої реалізації подальшого функціоналу програмного рішення, оскільки існує необхідність обробки доступу до окремого функціоналу системи, в залежності від рівня доступу (ролі) авторизованого користувача.

```
@SpringBootTest
@ActiveProfiles(profiles = {"develop", "h2"})
public class AbstractServiceDBTest {

    @Autowired protected UserBuilder userBuilder;
    @Autowired protected AuthenticatedUserService authenticatedUserService;

    @BeforeEach
    void createUser() {
        loginInUser();
    }

    void loginInUser() {
        loginInUser( Role.SESSION_USER  );
    }

    void loginInUser( Role role ) {
        AuthenticatedUserBuilder.start().role( role).login();
    }

    void logoff() {
        SecurityContextHolder.clearContext();
    }

}
```

Мал 2.4 – Підготовка тестових даних

В подальшому, була реалізована система аутентифікації та авторизації користувача, що дозволила розмежувати доступ до окремих функціональних компонентів системи, а також здійснювати перевірку можливості отримання доступу до інформації, що обробляється та зберігається.

Реалізації системи аутентифікації та авторизації достатньо для пришвидшення розробки, власне, це дозволило покращити теству інфраструктуру, оскільки тепер кожний тест має можливість виконувати свій функціонал без надлишкової підготовки.

В наступному етапі розробки програмного комплексу було додано Builder-класи, які дозволили автоматизувати рутинні процеси підготовки автоматизованих тестів, що дало можливість сфокусуватись при написанні тестів виключно на основному функціоналі програмного рішення. Використовуючи таку інфраструктуру, код тесту, фактично являє собою чистий опис його сценарію (мал 2.5).

```
@Test
void encodePasswordOnCreate() {
    User user = userBuilder.buildNew();
    String password = user.getPassword();
    user = userService.create( user );
    assertTrue( EncryptingUtils.checkPassword( password, user.getPassword() ) );
}
```

Мал 2.5 – Зразок написання тесту

Для зменшення залежностей в середовищі розробника, та в середовищі виконання тестів, було використано СКБД H2.

H2 база даних може бути запущена в режимі вбудованої бази даних, що розгортається в оперативній пам'яті або у файловій системі. Це дозволяє тестам працювати швидше, оскільки вони можуть використовувати легкі та швидкодіючі бази даних.

Використання вбудованої H2 бази даних дозволяє тестам працювати в ізольованому середовищі. Кожен тест може створювати свою власну тимчасову базу даних, яка очищається після закінчення тесту. Це допомагає уникнути

непередбачених впливів тестів один на одного та забезпечує консистентність результатів.

Загалом, використання H2 в тестах дозволяє розробникам створювати ефективні, ізольовані та легкі тестові сценарії для баз даних в середовищі мови програмування Java.

2.2.2 Імплементація необхідних CRUD операцій

Для імплементації CRUD операцій на всі необхідні сутності було проаналізовано та змінено реалізацію CRUD для сутності User, що дозволило змінити персоналізовані імплементації CRUD на абстрактні, які, в свою чергу, вирішили всі типові CRUD задачі.

Важливість абстракцій для CRUD (Create, Read, Update, Delete) задач, особливо в контексті реалізації Multi-Tenancy в Java, стає особливо очевидною, оскільки Multi-Tenancy - це архітектурний підхід, де одна і та ж сама програма обслуговує різних користувачів або орендарів (tenants), ізолюючі їх дані так, щоб вони не могли взаємодіяти або бачити дані інших користувачів. Для цього важливо мати абстракції, які дозволяють ефективно вирішувати завдання CRUD для кожного конкретного орендаря (tenant).

В подальшому було реалізовано Row-Level Tenancy тип бази даних, тобто дані зберігаються в одній таблиці бази, але до кожного рядка, що містить запис, додається стовпець, який вказує на того користувача (tenant), якому належить даний рядок. Такий підхід вимагає відділення даних між користувачами на рівні запитів.

Кожне Entity в базі даних є нащадком UserTenencyEntity що дозволяє реалізувати AbstractUserTenencyService, в якому перед створенням записується саме автор змін в базі даних (користувач, який в даний момент авторизований в системі).

Для кожного запиту на отримання даних, було додано на абстрактному рівні фільтрацію по поточному користувачу, що суттєво знижує ризики випадково отримати доступ одним користувачем, до даних іншого, або якось вплинути на дані іншого (мал. 2.6).

```
protected Specification< T > addSpecifications( boolean includeArchivedEntities ) {
    Specification< T > specification = SpecificationUtil.initEmptySpec();

    if ( entityIsUser() ) {
        return specification;
    }

    if ( entityIsExtendedByUser() && !userAuthorizationService.isRoot() ) {
        Long currentUserId = userAuthorizationService.getCurrentUserId();
        UserSpecification< T > sessionSpecification = new UserSpecification<>( currentUserId );
        specification = specification.and( sessionSpecification );
    }

    if ( !includeArchivedEntities && entityIsArchived() ) {
        ArchivedSpecification< T > archivedSpecification = new ArchivedSpecification<>();
        specification = specification.and( archivedSpecification );
    }

    return specification;
}
```

Мал 2.6 – Реалізація фільтрів по замовчуванню в AbstractDao

Для видалення було релізована стратегія “soft delete”. У цьому випадку замість фактичного видалення запису йому присвоюється статус “archived”, і цей статус використовується для відзначення записів, які вже не повинні відображатися в активних даних, але зберігаються в базі.

Ця стратегія дозволяє зберегти історію змін, уникнути втрати даних, і забезпечити можливість відновлення видалених даних, якщо це буде необхідно. Крім того, вона може бути корисною для забезпечення відповідності правилам та стандартам безпеки даних. Для реалізації цього було зроблені відповідні зміни в AbstractDao (мал. 2.6-2.7)

```
public void delete(T entity, boolean ignorePermissions) {
    if (!ignorePermissions && !isEditAllowed(entity)) {
        throw LocalizationUtils.buildRemovalNotAllowedExceptionNoPermission(entity);
    }

    if (entityIsArchived()) {
        ((Archivable) entity).setArchived(true);
        getRepository().save(entity);
    } else {
        getRepository().deleteById(entity.getId());
    }
}
```

Мал. 2.7 Реалізація стратегії “soft delete”

2.2.3 Імплементація REST API

На етапі розробки REST API, для коректного функціонування системи, необхідно реалізувати 2 типи логіки: CRUD API та Authentication API.

Authentication API

Authentication API використовується для забезпечення аутентифікації і авторизації користувачів при взаємодії з RESTful веб-сервісами. REST є архітектурним стилем, який використовує HTTP-протокол для комунікації між клієнтом і сервером.

Використання accessToken та refreshToken є частиною процесу аутентифікації та авторизації в системах, що використовують токени для забезпечення безпеки та зручності. Однак важливо відзначити, що використання логіну та пароля безпосередньо в додатках або сервісах є менш безпечним і рекомендується уникати його, особливо з точки зору захисту особистої інформації користувачів.

На цьому етапі варто розглянути загальний процес використання accessToken та refreshToken у зв'язці з логіном та паролем:

Аутентифікація. Користувач вводить свій логін та пароль в додатку або веб-сайті. Сервер перевіряє логін та пароль, і якщо вони вірні - видає accessToken та refreshToken.

Отримання токенів. Клієнтська сторона (додаток або веб-сайт) зберігає отримані accessToken та refreshToken.

Використання accessToken. Кожен раз, коли клієнтська сторона намагається отримати доступ до захищених ресурсів на сервері, вона включає accessToken в заголовок запиту (наприклад, як "Bearer token"). Сервер перевіряє дійсність та права доступу accessToken. Якщо токен дійсний і має необхідні дозволи, сервер повертає запитані дані.

Оновлення accessToken за допомогою refreshToken. Якщо accessToken закінчився або став неактивним, клієнт може використовувати refreshToken для отримання нового accessToken без необхідності введення логіна та пароля. Сервер перевіряє дійсність refreshToken, і якщо він є дійсним, видає новий accessToken та оновлений refreshToken.

Такий підхід дозволяє зменшити ризик витоку логіну та пароля, оскільки вони використовуються лише під час першої аутентифікації, а далі для доступу використовуються токени. Також, оновлення токенів робить процес безпечнішим і менш залежним від введення логіну та пароля кожен раз.

Отже, на основі проаналізованої вище теоретичної бази, в процесі реалізації Authentication API було передбачено наступні точки доступу:

- "/login" - запит для аутентифікації та авторизації з використанням логіну та паролю. У разі успішної операції користувач отримає такі значення як: accessToken, refreshToken, accessTokenLifetimeMinutes, refreshTokenLifetimeMinutes.
- "/refresh" - запит для повторної аутентифікації. Клієнт надсилає попередньо отриманий refreshToken, для отримання нової комбінації accessToken та refreshToken. Використати refreshToken можливо тільки 1 раз.
- "/logout" - запит про інформування виходу з системи. Клієнт надсилає попередньо отриманий refreshToken, який видаляється з бази даних. Таким чином, з часом його не можна буде використати для повторної аутефікації.

CRUD API

CRUD API був розроблений для простоти та зручності завантаження, редагування та розподілу даних для навчання.

Було створено такі API як:

- Session

- Period

Session API дозволяє користувачеві створювати session, які будуть накопичувати дані. Дані між сесіями не поширюються таким чином, є можливість накопичення даних з розподілом.

Було реалізовано такі методи API як:

- GET /sessions - для отримання всіх значень session доступних користувачеві, з можливостю фільтрації та сортування;
- GET /sessions/id - для отримання session за параметром id;
- POST /sessions - для створення session;
- PUT /sessions/id - для оновлення session за параметром id;
- DELETE /sessions/id - для видалення session за параметром id.

Period API дозволяє користувачеві накопичувати дані в session. Дані використовуються в навчання нейромережі.

Було реалізовано такі методи API як:

GET /periods - для отримання всіх значень period доступних користувачеві, з можливостю фільтрації та сортування;

GET /periods/id - для отримання period за параметром id

POST /periods - для створення period

PUT /periods/id - для оновлення period за параметром id

DELETE /periods/id - для видалення period за параметром id

2.2.4 Імплементація аналітичного модуля

Було додано бібліотеку “deeplearning4j” для створення нейромережі [11].

Обрано саме її оскільки вона дозволяє донавчання вже навчаної моделі. Для продукту було обрано декілька етапів навчання:

- базове навчання;

- донавчання на основі даних користувача.

Базове навчання ґрунтується на анонімних реальних даних, які вдалось отримати. Воно потрібне для швидкого старту роботи із мікросервісом.

Кожна окрема сесія при передбаченню використовує свою модель нейромережі. Ця модель являє собою копією базової модель, яка проходила додаткове навчання згідно даних користувача. Таким чином модель для кожного користувача буде адаптуватись відповідно до даних користувача.

Дані для навчання та тестування тули формалізованими. Це є основою із вимог роботи з бібліотекою (мал. 2.8).

```
SplitTestAndTrain testAndTrain = new org.nd4j.linalg.dataset.DataSet(input, labels).splitTestAndTrain(0.99);

DataSet trainingData = testAndTrain.getTrain();
DataSet testData = trainingData;

DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainingData);
normalizer.transform(trainingData);
normalizer.transform(testData);
```

Мал. 2.8 Приклад нормалізації даних

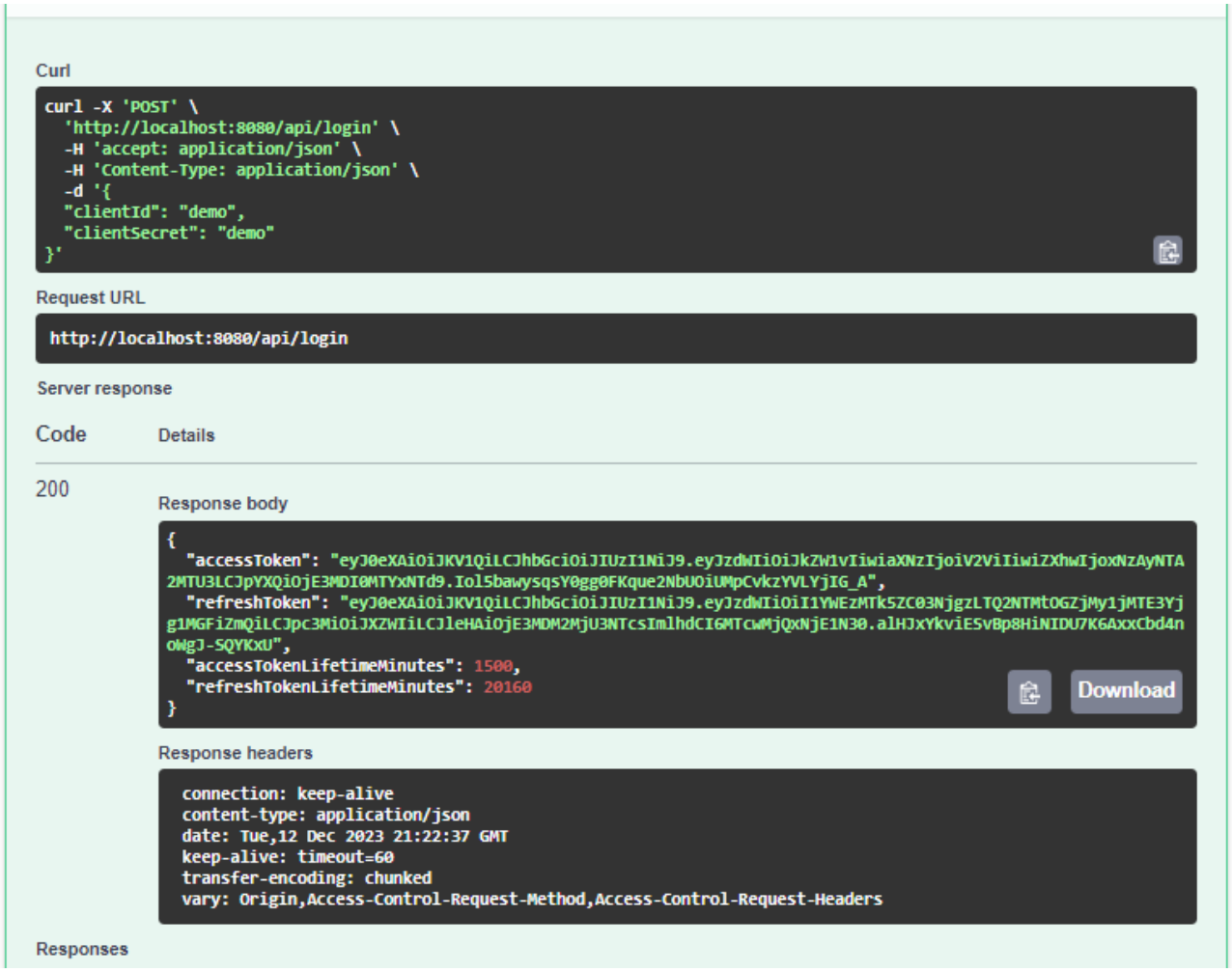
Розділ 3. Приклад роботи рішення

3.1 Авторизація

Для авторизації потрібно створити користувача, який буде працювати в системі. Для легкого налагодження мною був створений такий користувач.

В проекті доступна система документації Swagger (додаток Б), якою ми і скористаємось.

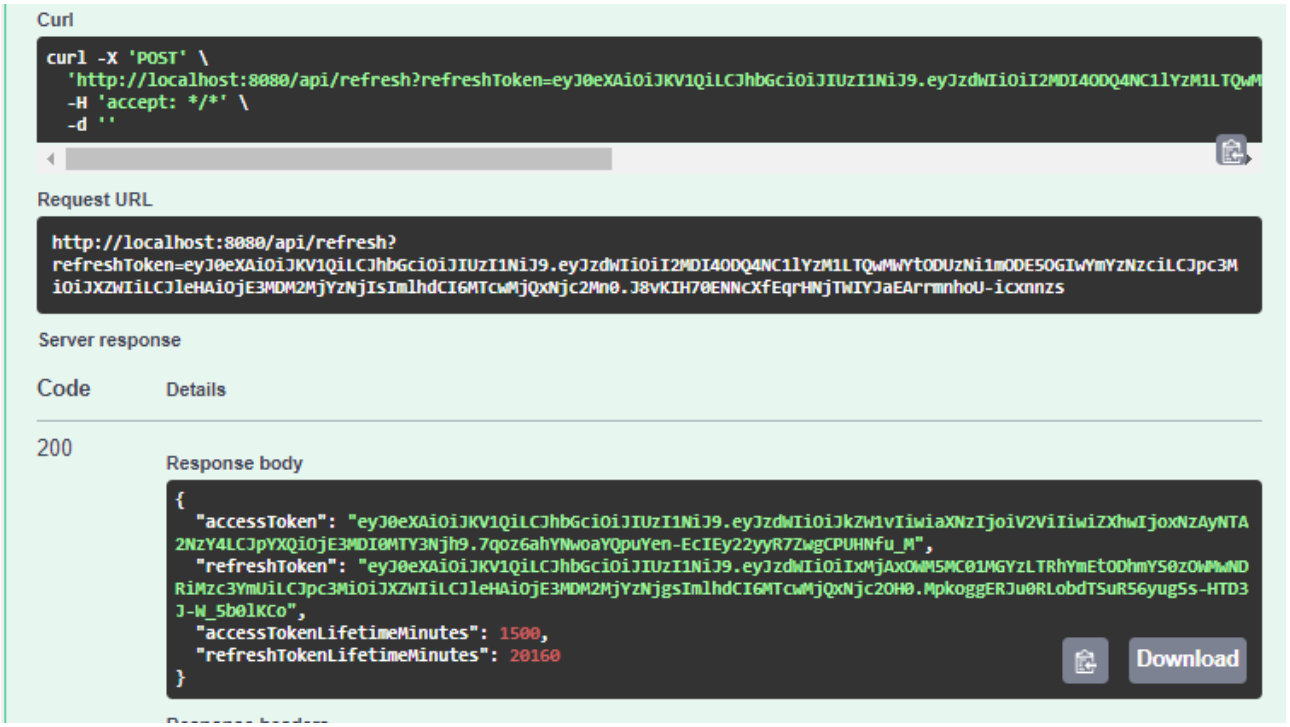
Зробивши POST /auth/login запит система верне ключі доступу, які можна використати в наступних запитах (мал. 3.1).



Мал. 3.1 POST /auth/login запит

Також відповідь містить час дійсності ключів доступу.

Якщо ключі доступу будуть не дійсні можна зробити POST /auth/refresh із refreshToken в запиті (мал. 3.2).

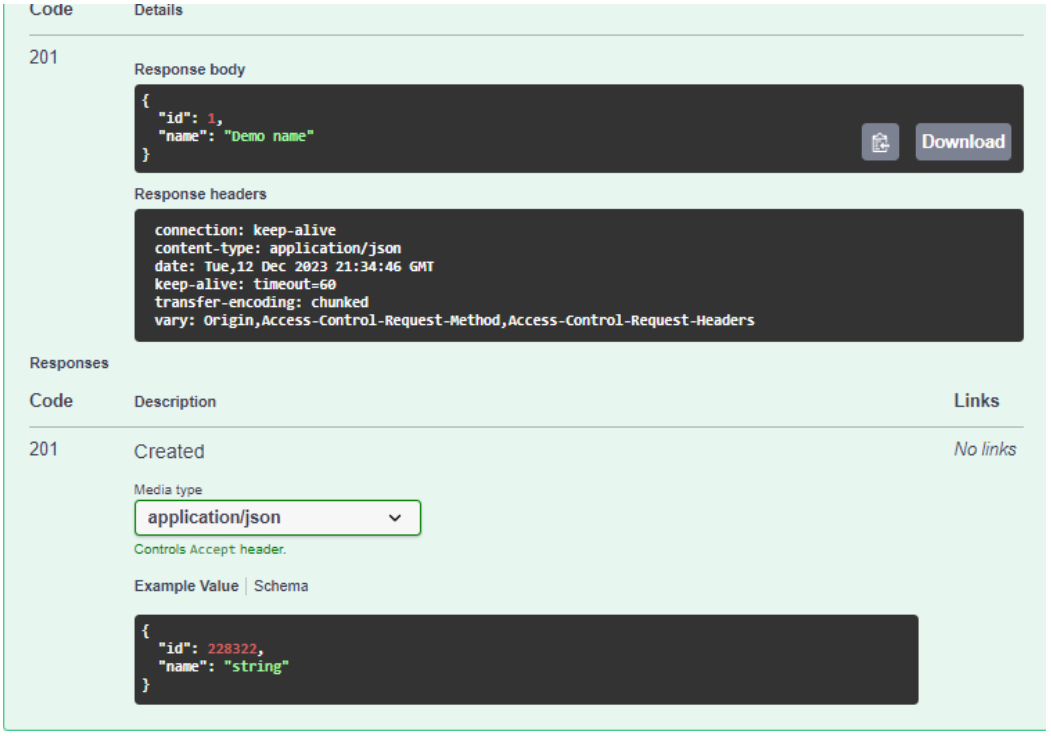


Мал. 3.2 POST /auth/refresh запит

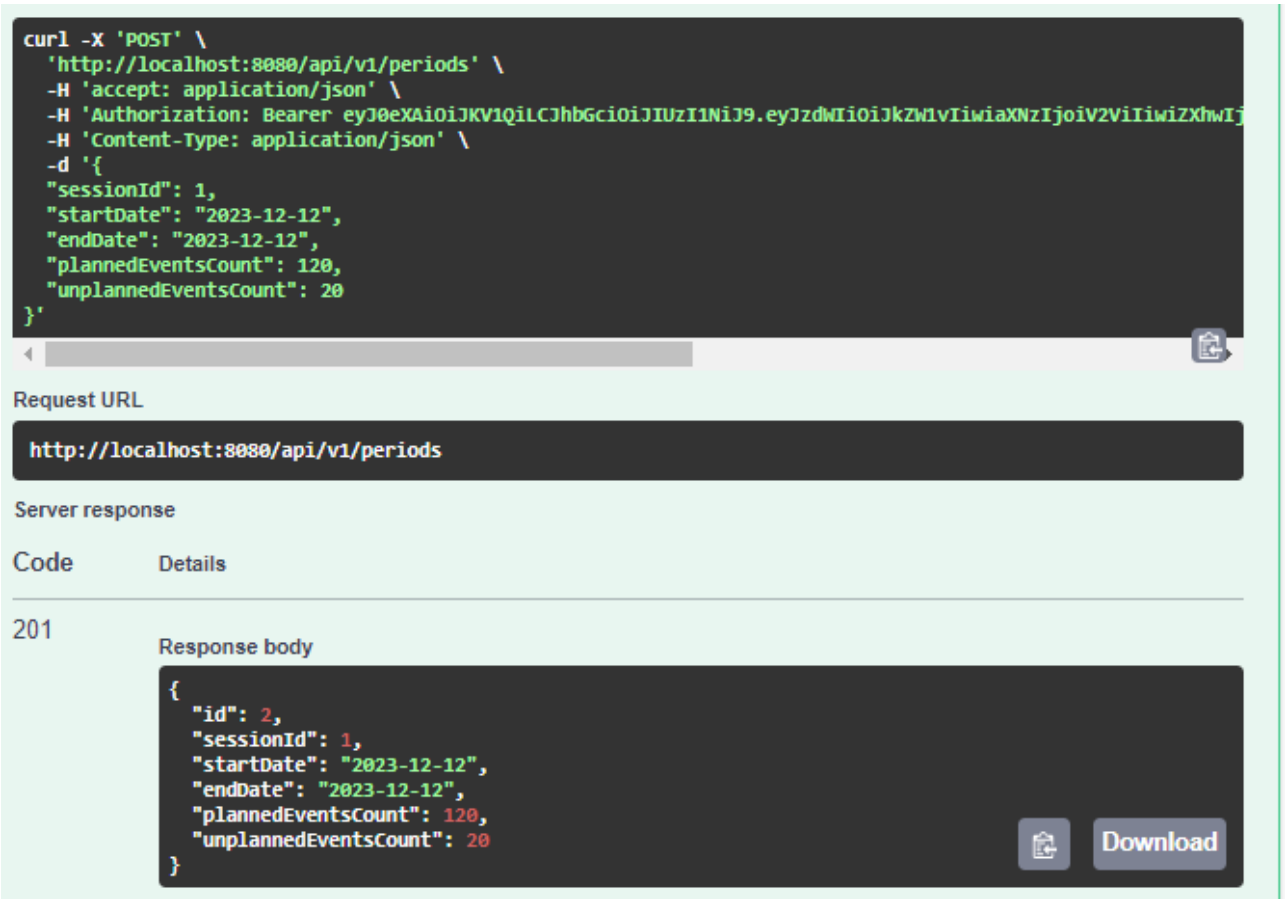
3.2 Заповнення даних

Для заповнення даних для користувача потрібно створити session (мал. 3.3).

Наступним кроком буде заповнення даних для навчання - такими даними є period (мал. 3.4).

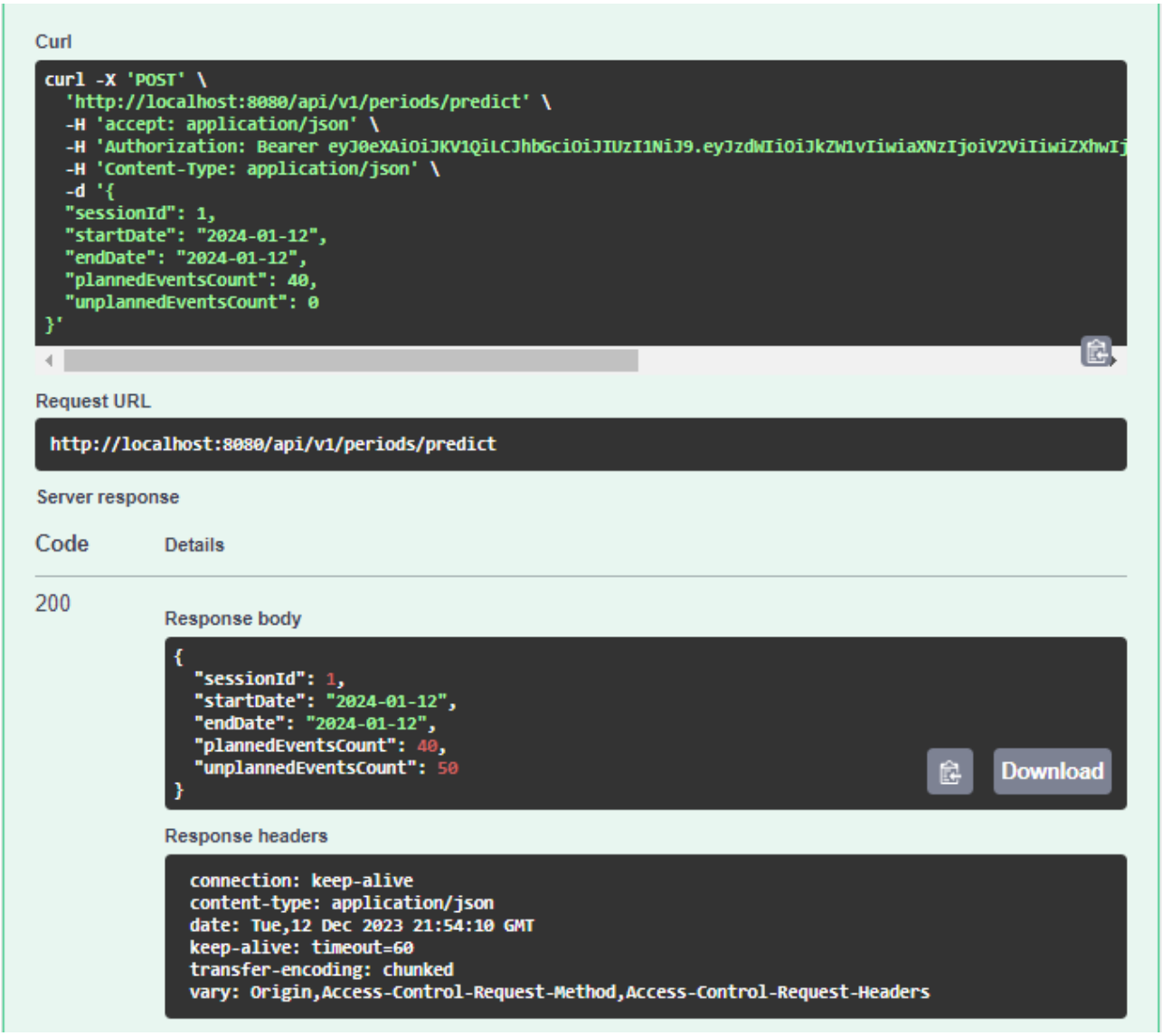


Мал. 3.3 Створення session



Мал. 3.4 Створення period

Як тільки заповнити дані можна виконувати запит на прогнозування значення на наступний period (мал. 3.5).



Мал. 3.5 Передбачення незапланованої відпустки

3.3 Використання та інтеграція

Мікросервіс надає змогу передбачати можливі незаплановані відпустки при плануванні робіт.

Розроблене рішення призначене для використання іншими програмними продуктами за допомогою доступу через REST API.

Приклад ймовірного використання є розробка плагіну для Jira. Який при плануванні робіт буде показувати користувачеві неочікуванні відпустки.

Вхідним даними для навчання є:

- початок періоду (дата);
- кінець періоду (дата);
- кількість запланованих відсутностей.

Вихідними дані - це кількість незапланованих відсутностей.

Мікросервіс моє вже навчену базову модель нейромережі. Рішення робить копію даної моделі для кожного користувача та сесії. Таким чином є можливість швидко розпочати роботу із мікросервісом та відділити можливий вплив між користувачами один на одного.

Дані для базової моделі були взяти із комерційного рішення AbsencePlanner, які відображають реальну поведінку користувачів (додаток А).

AbsencePlanner - це рішення для планування відпусток. Збір даних про відпустки, підтримка балансів відпусток в актуальному стані, відстеження та усунення збігів.

На даний момент відбувається тестовий процес інтеграції із продуктом AbsencePlanner. Він використовує вже начену модель для передбачення майбутнього періоду, а коли цей період настає, вже заповнює фактичними даними наш мікросервіс. Таким чином постійно адаптуючи його.

Мікросервіс є призначений для використання будь-який зовнішнім програмним забезпеченням, який має можливість спілкуватись за допомогою мережі інтернет.

Саме для цього була розроблена архітектура рішення таким чином, щоб мінімізувати сповільнення швидкодії продукту та, за необхідністю, динамічно використовувати ресурси сервера, забезпечити стабільну роботу із багатьма продуктами інтеграції.

ВИСНОВКИ

У даній кваліфікаційній роботі була розглянута актуальна проблема планування виконання робіт на підприємствах, зокрема у контексті відсутності працівників. Актуальність цієї теми зумовлена великим різноманіттям причин, які можуть впливати на наявність працівників на робочому місці, починаючи від хвороб і закінчуючи сімейними обставинами.

Метою роботи було розробити інструмент для аналізу минулих відсутностей працівників та їх передбачення в майбутньому. Розроблений інструмент базується на мікросервісній архітектурі з використанням технологій Java, Spring Boot, DJL, дозволяє виконувати функції створення, редагування та перегляду подій, пов'язаних з відсутностями працівників на робочому місці.

Завданням роботи було розробити програмне рішення, яке не лише виконує потрібні функції, але й відповідає вимогам до застосунку таким як можливість використання іншими застосунками, масштабованість та зручне використання REST API.

Під час виконання роботи були досліджені і порівняні програмні рішення подібної тематики, вирішені завдання з розробки та тестування мікросервісу. Створений мікросервіс використовує відомі технології та бази даних, що гарантує його ефективність та надійність.

Отже, робота виконала свою мету та завдання, надаючи практичний інструмент для управління відсутностями працівників на підприємствах, що сприятиме покращенню процесів планування та функціонування підприємства в цілому.

СПИСОК ЛІТЕРАТУРИ

1. Microsoft Project [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/project/project-management-software
2. Smartsheet [Електронний ресурс]. – Режим доступу: https://www.smartsheet.com
3. Trello [Електронний ресурс]. – Режим доступу: https://trello.com/tour
4. Asana [Електронний ресурс]. – Режим доступу: https://asana.com
5. Jira [Електронний ресурс]. – Режим доступу https://www.atlassian.com/software/jira/features
6. Monday.com [Електронний ресурс]. – Режим доступу: https://monday.com
7. Wrike [Електронний ресурс]. – Режим доступу: https://www.wrike.com
8. Elastic Load Balancing [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/elasticloadbalancing
9. Spring Boot [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-boot
10. Swagger [Електронний ресурс] – Режим доступу: https://swagger.io
11. Deeplearning4j Suite Overview [Електронний ресурс] – Режим доступу: https://deeplearning4j.konduit.ai

ДОДАТКИ

Додаток А. Приклад даних для навчання базової моделі

```
"total","unplanned","startperiod","endperiod"
"1","1","2016-11-21 00:00:00","2016-12-19 00:00:00"
"21","19","2016-12-19 00:00:00","2017-01-16 00:00:00"
"36","31","2017-01-16 00:00:00","2017-02-13 00:00:00"
"36","28","2017-02-13 00:00:00","2017-03-13 00:00:00"
"40","37","2017-03-13 00:00:00","2017-04-10 00:00:00"
"40","30","2017-04-10 00:00:00","2017-05-08 00:00:00"
"30","27","2017-05-08 00:00:00","2017-06-05 00:00:00"
"50","28","2017-06-05 00:00:00","2017-07-03 00:00:00"
"51","29","2017-07-03 00:00:00","2017-07-31 00:00:00"
"62","28","2017-07-31 00:00:00","2017-08-28 00:00:00"
"49","37","2017-08-28 00:00:00","2017-09-25 00:00:00"
"24","19","2017-09-25 00:00:00","2017-10-23 00:00:00"
"37","32","2017-10-23 00:00:00","2017-11-20 00:00:00"
"25","22","2017-11-20 00:00:00","2017-12-18 00:00:00"
"76","38","2017-12-18 00:00:00","2018-01-15 00:00:00"
"43","33","2018-01-15 00:00:00","2018-02-12 00:00:00"
"54","44","2018-02-12 00:00:00","2018-03-12 00:00:00"
"38","35","2018-03-12 00:00:00","2018-04-09 00:00:00"
"44","36","2018-04-09 00:00:00","2018-05-07 00:00:00"
"36","29","2018-05-07 00:00:00","2018-06-04 00:00:00"
"71","41","2018-06-04 00:00:00","2018-07-02 00:00:00"
"53","36","2018-07-02 00:00:00","2018-07-30 00:00:00"
"49","23","2018-07-30 00:00:00","2018-08-27 00:00:00"
"54","29","2018-08-27 00:00:00","2018-09-24 00:00:00"
"24","21","2018-09-24 00:00:00","2018-10-22 00:00:00"
"27","24","2018-10-22 00:00:00","2018-11-19 00:00:00"
"31","23","2018-11-19 00:00:00","2018-12-17 00:00:00"
"111","37","2018-12-17 00:00:00","2019-01-14 00:00:00"
"40","39","2019-01-14 00:00:00","2019-02-11 00:00:00"
"32","30","2019-02-11 00:00:00","2019-03-11 00:00:00"
"36","29","2019-03-11 00:00:00","2019-04-08 00:00:00"
"50","29","2019-04-08 00:00:00","2019-05-06 00:00:00"
"52","28","2019-05-06 00:00:00","2019-06-03 00:00:00"
"52","22","2019-06-03 00:00:00","2019-07-01 00:00:00"
"48","24","2019-07-01 00:00:00","2019-07-29 00:00:00"
"61","32","2019-07-29 00:00:00","2019-08-26 00:00:00"
"58","46","2019-08-26 00:00:00","2019-09-23 00:00:00"
"36","27","2019-09-23 00:00:00","2019-10-21 00:00:00"
"39","31","2019-10-21 00:00:00","2019-11-18 00:00:00"
"48","39","2019-11-18 00:00:00","2019-12-16 00:00:00"
"127","39","2019-12-16 00:00:00","2020-01-13 00:00:00"
"44","40","2020-01-13 00:00:00","2020-02-10 00:00:00"
"55","47","2020-02-10 00:00:00","2020-03-09 00:00:00"
"31","31","2020-03-09 00:00:00","2020-04-06 00:00:00"
"37","34","2020-04-06 00:00:00","2020-05-04 00:00:00"
"38","36","2020-05-04 00:00:00","2020-06-01 00:00:00"
"47","46","2020-06-01 00:00:00","2020-06-29 00:00:00"
"77","61","2020-06-29 00:00:00","2020-07-27 00:00:00"
"81","56","2020-07-27 00:00:00","2020-08-24 00:00:00"
"62","39","2020-08-24 00:00:00","2020-09-21 00:00:00"
"51","45","2020-09-21 00:00:00","2020-10-19 00:00:00"
"51","46","2020-10-19 00:00:00","2020-11-16 00:00:00"
"48","41","2020-11-16 00:00:00","2020-12-14 00:00:00"
"129","24","2020-12-14 00:00:00","2021-01-11 00:00:00"
"36","29","2021-01-11 00:00:00","2021-02-08 00:00:00"
"46","38","2021-02-08 00:00:00","2021-03-08 00:00:00"
"69","49","2021-03-08 00:00:00","2021-04-05 00:00:00"
```

```
"63","56","2021-04-05 00:00:00","2021-05-03 00:00:00"
"55","39","2021-05-03 00:00:00","2021-05-31 00:00:00"
"37","31","2021-05-31 00:00:00","2021-06-28 00:00:00"
"74","52","2021-06-28 00:00:00","2021-07-26 00:00:00"
"45","25","2021-07-26 00:00:00","2021-08-23 00:00:00"
"83","66","2021-08-23 00:00:00","2021-09-20 00:00:00"
"93","53","2021-09-20 00:00:00","2021-10-18 00:00:00"
"45","39","2021-10-18 00:00:00","2021-11-15 00:00:00"
"39","35","2021-11-15 00:00:00","2021-12-13 00:00:00"
"130","53","2021-12-13 00:00:00","2022-01-10 00:00:00"
"50","48","2022-01-10 00:00:00","2022-02-07 00:00:00"
"77","72","2022-02-07 00:00:00","2022-03-07 00:00:00"
"40","38","2022-03-07 00:00:00","2022-04-04 00:00:00"
"49","42","2022-04-04 00:00:00","2022-05-02 00:00:00"
"46","44","2022-05-02 00:00:00","2022-05-30 00:00:00"
"71","69","2022-05-30 00:00:00","2022-06-27 00:00:00"
"84","78","2022-06-27 00:00:00","2022-07-25 00:00:00"
"54","47","2022-07-25 00:00:00","2022-08-22 00:00:00"
"106","98","2022-08-22 00:00:00","2022-09-19 00:00:00"
"97","90","2022-09-19 00:00:00","2022-10-17 00:00:00"
"94","83","2022-10-17 00:00:00","2022-11-14 00:00:00"
"69","58","2022-11-14 00:00:00","2022-12-12 00:00:00"
"154","108","2022-12-12 00:00:00","2023-01-09 00:00:00"
"55","48","2023-01-09 00:00:00","2023-02-06 00:00:00"
"64","61","2023-02-06 00:00:00","2023-03-06 00:00:00"
"94","91","2023-03-06 00:00:00","2023-04-03 00:00:00"
"98","90","2023-04-03 00:00:00","2023-05-01 00:00:00"
"96","89","2023-05-01 00:00:00","2023-05-29 00:00:00"
"122","117","2023-05-29 00:00:00","2023-06-26 00:00:00"
"108","87","2023-06-26 00:00:00","2023-07-24 00:00:00"
"114","91","2023-07-24 00:00:00","2023-08-21 00:00:00"
"124","105","2023-08-21 00:00:00","2023-09-18 00:00:00"
"106","97","2023-09-18 00:00:00","2023-10-16 00:00:00"
"86","73","2023-10-16 00:00:00","2023-11-13 00:00:00"
"81","60","2023-11-13 00:00:00","2023-12-11 00:00:00"
"17","0","2023-12-11 00:00:00","2024-01-08 00:00:00"
```

Додаток Б. REST API документація


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "Web API",
    "license": {
      "name": "Apache 2.0",
      "url": "http://springdoc.org"
    },
    "version": "v1"
  },
  "servers": [
    {
      "url": "http://localhost:8080",
      "description": "Generated server url"
    }
  ],
  "paths": {
    "/api/v1/sessions/{id}": {
      "get": {
        "tags": [
          "Session"
        ],
        "summary": "Retrieve a specific record by id",
        "description": "This operation retrieves specific record of current entity
from database\n- <b>expand_fields</b> - is used to specify the entity fields that
need to be included in the response.\n- <b>id</b> - is used to specify the
identification number of record which must be returned.",
        "operationId": "getRecord",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "integer",
              "format": "int64"
            }
          },
          {
            "name": "expand_fields",
            "in": "query",
            "description": "Specify fields which has to be expanded in response.
To check fields which support expanding, please refer to response model",
            "required": false,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
```

```
                "$ref": "#/components/schemas/SessionResponse"
              }
            }
          }
        }
      },
      "security": [
        {
          "Bearer Authentication": []
        }
      ]
    },
    "put": {
      "tags": [
        "Session"
      ],
      "summary": "Update a specific record, based on the id",
      "description": "Update a specific record, based on the id<br>All values
will be replaced by the input data",
      "operationId": "updateRecord",
      "parameters": [
        {
          "name": "id",
          "in": "path",
          "required": true,
          "schema": {
            "type": "integer",
            "format": "int64"
          }
        }
      ],
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/SessionRequest"
            }
          }
        },
        "required": true
      },
      "responses": {
        "406": {
          "description": "Invalid property value",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/SessionResponse"
              }
            }
          }
        },
        "404": {
          "description": "The record to be updated does not exist.",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/SessionResponse"
              }
            }
          }
```

```
    },
    "200": {
      "description": "OK",
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/SessionResponse"
          }
        }
      }
    }
  },
  "security": [
    {
      "Bearer Authentication": []
    }
  ]
},
"delete": {
  "tags": [
    "Session"
  ],
  "summary": "Delete record by id",
  "operationId": "deleteRecord",
  "parameters": [
    {
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {
        "type": "integer",
        "format": "int64"
      }
    }
  ],
  "responses": {
    "204": {
      "description": "No Content"
    }
  },
  "security": [
    {
      "Bearer Authentication": []
    }
  ]
},
"patch": {
  "tags": [
    "Session"
  ],
  "summary": "Patch a specific record, based on the id",
  "operationId": "patchRecord",
  "parameters": [
    {
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {
        "type": "integer",
        "format": "int64"
      }
```

```
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "additionalProperties": {
                  "type": "object"
                }
              }
            }
          },
          "required": true
        },
        "responses": {
          "404": {
            "description": "The request was empty",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/SessionResponse"
                }
              }
            }
          },
          "405": {
            "description": "The PATCH method is not implemented for this
endpoint.",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/SessionResponse"
                }
              }
            }
          }
        },
        "security": [
          {
            "Bearer Authentication": []
          }
        ]
      }
    },
    "/api/v1/periods/{id}": {
      "get": {
        "tags": [
          "Period"
        ],
        "summary": "Retrieve a specific record by id",
        "description": "This operation retrieves specific record of current entity
from database\n- <b>expand_fields</b> - is used to specify the entity fields that
need to be included in the response.\n- <b>id</b> - is used to specify the
identification number of record which must be returned.",
        "operationId": "getRecord_1",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "required": true,
```

```
            "schema": {
              "type": "integer",
              "format": "int64"
            }
          },
          {
            "name": "expand_fields",
            "in": "query",
            "description": "Specify fields which has to be expanded in response.
To check fields which support expanding, please refer to response model",
            "required": false,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/PeriodResponse"
                }
              }
            }
          }
        },
        "security": [
          {
            "Bearer Authentication": []
          }
        ]
      },
      "put": {
        "tags": [
          "Period"
        ],
        "summary": "Update a specific record, based on the id",
        "description": "Update a specific record, based on the id<br>All values
will be replaced by the input data",
        "operationId": "updateRecord_1",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "integer",
              "format": "int64"
            }
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/PeriodRequest"
              }
            }
          },
```

```
      "required": true
    },
    "responses": {
      "406": {
        "description": "Invalid property value",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/PeriodResponse"
            }
          }
        }
      },
      "404": {
        "description": "The record to be updated does not exist.",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/PeriodResponse"
            }
          }
        }
      },
      "200": {
        "description": "OK",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/PeriodResponse"
            }
          }
        }
      }
    },
    "security": [
      {
        "Bearer Authentication": []
      }
    ]
  },
  "delete": {
    "tags": [
      "Period"
    ],
    "summary": "Delete record by id",
    "operationId": "deleteRecord_1",
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer",
          "format": "int64"
        }
      }
    ],
    "responses": {
      "204": {
        "description": "No Content"
      }
```

```
        },
        "security": [
          {
            "Bearer Authentication": []
          }
        ]
      },
      "patch": {
        "tags": [
          "Period"
        ],
        "summary": "Patch a specific record, based on the id",
        "operationId": "patchRecord_1",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "integer",
              "format": "int64"
            }
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "additionalProperties": {
                  "type": "object"
                }
              }
            }
          },
          "required": true
        },
        "responses": {
          "404": {
            "description": "The request was empty",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/PeriodResponse"
                }
              }
            }
          },
          "405": {
            "description": "The PATCH method is not implemented for this
endpoint.",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/PeriodResponse"
                }
              }
            }
          }
        },
        "security": [
```

```
          {
            "Bearer Authentication": []
          }
        ]
      }
    },
    "/api/v1/sessions": {
      "get": {
        "tags": [
          "Session"
        ],
        "summary": "Retrieve a list of all records",
        "operationId": "getAll",
        "parameters": [
          {
            "name": "index",
            "in": "query",
            "description": "Page number of the requested page. The default value
is 0",
            "required": false,
            "schema": {
              "type": "integer",
              "format": "int32"
            },
            "example": 0
          },
          {
            "name": "size",
            "in": "query",
            "description": "The size of the requested page. The default value is
25. Maximum size is 100",
            "required": false,
            "schema": {
              "type": "integer",
              "format": "int32"
            },
            "example": 25
          },
          {
            "name": "search",
            "in": "query",
            "description": "Search expression used to filter results. To check
fields which support filtering, please reffer to the response model",
            "required": false,
            "schema": {
              "type": "string"
            },
            "example": "field1=value1,field2!=value2,field3:value3"
          },
          {
            "name": "sort",
            "in": "query",
            "description": "Specify fields to sort. The default sorting is
ascending",
            "required": true,
            "schema": {
              "$ref": "#/components/schemas/Sort"
            }
          },
          {
            "name": "expand_fields",
```

```
            "in": "query",
            "description": "Specify fields which have to be expanded in the
response. To check fields which support expanding, please refer to the response
model",
            "required": false,
            "schema": {
              "type": "string"
            },
            "example": "field1,field2"
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/PageResponseSessionResponse"
                }
              }
            }
          }
        },
        "security": [
          {
            "Bearer Authentication": []
          }
        ]
      },
      "post": {
        "tags": [
          "Session"
        ],
        "summary": "Create record",
        "operationId": "createRecord",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/SessionRequest"
              }
            }
          },
          "required": true
        },
        "responses": {
          "201": {
            "description": "Created",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/SessionResponse"
                }
              }
            }
          }
        },
        "security": [
          {
            "Bearer Authentication": []
          }
```

```
        ]
      }
    },
    "/api/v1/periods": {
      "get": {
        "tags": [
          "Period"
        ],
        "summary": "Retrieve a list of all records",
        "operationId": "getAll_1",
        "parameters": [
          {
            "name": "index",
            "in": "query",
            "description": "Page number of the requested page. The default value
is 0",
            "required": false,
            "schema": {
              "type": "integer",
              "format": "int32"
            },
            "example": 0
          },
          {
            "name": "size",
            "in": "query",
            "description": "The size of the requested page. The default value is
25. Maximum size is 100",
            "required": false,
            "schema": {
              "type": "integer",
              "format": "int32"
            },
            "example": 25
          },
          {
            "name": "search",
            "in": "query",
            "description": "Search expression used to filter results. To check
fields which support filtering, please reffer to the response model",
            "required": false,
            "schema": {
              "type": "string"
            },
            "example": "field1=value1,field2!=value2,field3:value3"
          },
          {
            "name": "sort",
            "in": "query",
            "description": "Specify fields to sort. The default sorting is
ascending",
            "required": true,
            "schema": {
              "$ref": "#/components/schemas/Sort"
            }
          },
          {
            "name": "expand_fields",
            "in": "query",
```

```
            "description": "Specify fields which have to be expanded in the
response. To check fields which support expanding, please refer to the response
model",
            "required": false,
            "schema": {
              "type": "string"
            },
            "example": "field1,field2"
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/PageResponsePeriodResponse"
                }
              }
            }
          }
        },
        "security": [
          {
            "Bearer Authentication": []
          }
        ]
      },
      "post": {
        "tags": [
          "Period"
        ],
        "summary": "Create record",
        "operationId": "createRecord_1",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/PeriodRequest"
              }
            }
          },
          "required": true
        },
        "responses": {
          "201": {
            "description": "Created",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/PeriodResponse"
                }
              }
            }
          }
        },
        "security": [
          {
            "Bearer Authentication": []
          }
        ]
```

```
  }
},
"/api/v1/periods/predict": {
  "post": {
    "tags": [
      "Period"
    ],
    "summary": "Create record",
    "operationId": "createRecord_2",
    "requestBody": {
      "content": {
        "application/json": {
          "schema": {
            "$ref": "#/components/schemas/PeriodRequest"
          }
        }
      },
      "required": true
    },
    "responses": {
      "200": {
        "description": "OK",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/PeriodRequest"
            }
          }
        }
      }
    },
    "security": [
      {
        "Bearer Authentication": []
      }
    ]
  }
},
"/api/refresh": {
  "post": {
    "tags": [
      "Authentication"
    ],
    "operationId": "refreshToken",
    "parameters": [
      {
        "name": "refreshToken",
        "in": "query",
        "required": true,
        "schema": {
          "type": "string"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "OK",
        "content": {
          "*/*": {
            "schema": {
              "$ref": "#/components/schemas/AuthenticationResponse"
```

```
              }
            }
          }
        }
      }
    },
    "/api/logout": {
      "post": {
        "tags": [
          "Authentication"
        ],
        "operationId": "logout",
        "parameters": [
          {
            "name": "refreshToken",
            "in": "query",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          }
        }
      }
    },
    "/api/login": {
      "post": {
        "tags": [
          "Authentication"
        ],
        "operationId": "authenticationRequest",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/AuthenticationRequest"
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/AuthenticationResponse"
                }
              }
            }
          }
        }
      }
    }
  },
```

```
"components": {
  "schemas": {
    "SessionRequest": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        }
      }
    },
    "SessionResponse": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "description": "The id of the record<br/>Sortable",
          "format": "int64",
          "example": 228322
        },
        "name": {
          "type": "string"
        }
      }
    },
    "PeriodRequest": {
      "type": "object",
      "properties": {
        "sessionId": {
          "type": "integer",
          "format": "int64"
        },
        "startDate": {
          "type": "string",
          "format": "date"
        },
        "endDate": {
          "type": "string",
          "format": "date"
        },
        "plannedEventsCount": {
          "type": "integer",
          "format": "int32"
        },
        "unplannedEventsCount": {
          "type": "integer",
          "format": "int32"
        }
      }
    },
    "PeriodResponse": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "description": "The id of the record<br/>Sortable",
          "format": "int64",
          "example": 228322
        },
        "sessionId": {
          "type": "integer",
          "format": "int64"
```

```
      },
      "startDate": {
        "type": "string",
        "format": "date"
      },
      "endDate": {
        "type": "string",
        "format": "date"
      },
      "plannedEventsCount": {
        "type": "integer",
        "format": "int32"
      },
      "unplannedEventsCount": {
        "type": "integer",
        "format": "int32"
      }
    }
  },
  "AuthenticationResponse": {
    "type": "object",
    "properties": {
      "accessToken": {
        "type": "string"
      },
      "refreshToken": {
        "type": "string"
      },
      "accessTokenLifetimeMinutes": {
        "type": "integer",
        "format": "int32"
      },
      "refreshTokenLifetimeMinutes": {
        "type": "integer",
        "format": "int32"
      }
    }
  },
  "AuthenticationRequest": {
    "type": "object",
    "properties": {
      "clientId": {
        "type": "string"
      },
      "clientSecret": {
        "type": "string"
      }
    }
  },
  "Sort": {
    "type": "object",
    "properties": {
      "sort": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    }
  },
  "PageResponseSessionResponse": {
```

```
      "type": "object",
      "properties": {
        "totalElements": {
          "type": "integer",
          "format": "int64"
        },
        "index": {
          "type": "integer",
          "format": "int64"
        },
        "size": {
          "type": "integer",
          "format": "int64"
        },
        "items": {
          "type": "array",
          "items": {
            "$ref": "#/components/schemas/SessionResponse"
          }
        }
      }
    },
    "PageResponsePeriodResponse": {
      "type": "object",
      "properties": {
        "totalElements": {
          "type": "integer",
          "format": "int64"
        },
        "index": {
          "type": "integer",
          "format": "int64"
        },
        "size": {
          "type": "integer",
          "format": "int64"
        },
        "items": {
          "type": "array",
          "items": {
            "$ref": "#/components/schemas/PeriodResponse"
          }
        }
      }
    }
  },
  "securitySchemes": {
    "Bearer Authentication": {
      "type": "http",
      "scheme": "bearer",
      "bearerFormat": "JWT"
    }
  }
}
```

Додаток В. Використані залежності Future-leave (pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.3</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.voltor</groupId>
  <artifactId>future-leave</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>future-leave</name>
  <description>Analytic module</description>
  <properties>
    <java.version>17</java.version>
    <dl4j.version>1.0.0-M2.1</dl4j.version>

  </properties>
  <dependencies>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-batch</artifactId>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-configuration-processor</artifactId>
      <scope>compile</scope>
```

```
</dependency>

<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.3.0</version>
</dependency>

<dependency>
  <groupId>com.auth0</groupId>
  <artifactId>java-jwt</artifactId>
  <version>4.0.0</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcprov-jdk15on</artifactId>
  <version>1.70</version>
</dependency>

<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>32.1.2-jre</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <scope>compile</scope>
</dependency>

<dependency>
  <groupId>org.deeplearning4j</groupId>
  <artifactId>deeplearning4j-core</artifactId>
  <version>${dl4j.version}</version>
</dependency>

<dependency>
  <groupId>org.deeplearning4j</groupId>
```

```
      <artifactId>deeplearning4j-nlp</artifactId>
      <version>${dl4j.version}</version>
    </dependency>
    <dependency>
      <groupId>org.nd4j</groupId>
      <artifactId>nd4j-native-platform</artifactId>
      <version>${dl4j.version}</version>
    </dependency>

    <dependency>
      <groupId>com.opencsv</groupId>
      <artifactId>opencsv</artifactId>
      <version>5.5</version> <!-- Replace with the latest version -->
    </dependency>

    <dependency>
      <groupId>commons-cli</groupId>
      <artifactId>commons-cli</artifactId>
      <version>1.5.0</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>


</project>
```

Додаток Г. Лістинг Future-leave

```
package com.voltor.futureleave.api.v1;
public class ApiConstants {
  private ApiConstants() {
  }
  // API
  public static final String API_PREFIX = "/api";
  public static final String API_VERSION_V1 = "/v1";
  // Endpoints
  public static final String AUTHENTICATION_ENDPOINT = "/login";
  public static final String REFRESH_TOKEN_ENDPOINT = "/refresh";
  public static final String V1_SESSION_ENDPOINT = API_PREFIX + API_VERSION_V1 +
"/sessions";
  public static final String V1_PERIOD_ENDPOINT = API_PREFIX + API_VERSION_V1 +
"/periods";
}


package com.voltor.futureleave.api.v1.authentication;

@Controller
@RequestMapping( AuthenticationController.API_URL )
@Tag( name = "Authentication" )
@Transactional
public class AuthenticationController {
  public static final String API_URL = ApiConstants.API_PREFIX;
  private final AuthenticationManager authenticationManager;
  private final JwtService jwtService;
  private final AuthenticatedUserService userAuthorizationService;
  private final UserDetailsServiceImpl userDetailsService;
  private final RefreshTokenService refreshTokenService;
  @Autowired
  public AuthenticationController(
      AuthenticationManager authenticationManager,
      JwtService jwtService,
      AuthenticatedUserService userAuthorizationService,
      UserDetailsServiceImpl userDetailsService,
      RefreshTokenService refreshTokenService) {
    this.authenticationManager = authenticationManager;
    this.jwtService = jwtService;
    this.userAuthorizationService = userAuthorizationService;
    this.userDetailsService = userDetailsService;
    this.refreshTokenService = refreshTokenService;
  }
  @PostMapping(path = "/login",
      consumes = MediaType.APPLICATION_JSON_VALUE,
      produces = MediaType.APPLICATION_JSON_VALUE)
  @ResponseBody
  public AuthenticationResponse authenticationRequest(@RequestBody
AuthenticationRequest request,
                            HttpServletResponse httpServletResponse) {
    String clientId = request.getClientId();
    String clientSecret = request.getClientSecret();
    Authentication authentication = this.authenticationManager
        .authenticate( new UsernamePasswordAuthenticationToken( clientId,
clientSecret ) );
    userAuthorizationService.authenticateUser( authentication );
    return buildAuthenticationResponse();
  }
```

```
  @PostMapping(path = "/logout")
  @ResponseBody
  public void logout(@RequestParam @NotEmpty String refreshToken) {
    refreshTokenService.delete(refreshToken);
  }
  @PostMapping(path = "/refresh")
  @ResponseBody
  public AuthenticationResponse refreshToken(@RequestParam @NotEmpty String
refreshToken) {
    validateRefreshToken(refreshToken);
    User user = refreshTokenService.getTokenData(refreshToken);
    userAuthorizationService.authenticateUser(
userDetailsService.authenticateUser( user ) );
    return buildAuthenticationResponse();
  }
  private void validateRefreshToken(String refreshToken){
    try {
      jwtService.verifyToken(refreshToken);
    } catch ( JwtExpirationException expiredException ){
      refreshTokenService.delete( refreshToken );
      throw expiredException;
    }
  }
  private AuthenticationResponse buildAuthenticationResponse() {
    return new AuthenticationResponse(
        jwtService.generateAccessToken(),
        jwtService.generateRefreshToken(),
        jwtService.getAccessTokenExpirationInMinutes(),
        jwtService.getRefreshTokenExpirationInMinutes() );
  }
}


package com.voltor.futureleave.api.v1.authentication;
public class AuthenticationRequest {
  private String clientId;
  private String clientSecret;

  public String getClientId() {
    return clientId;
  }
  public void setClientId( String clientId ) {
    this.clientId = clientId;
  }
  public String getClientSecret() {
    return clientSecret;
  }
  public void setClientSecret( String clientSecret ) {
    this.clientSecret = clientSecret;
  }
}
package com.voltor.futureleave.api.v1.authentication;
public class AuthenticationResponse {
  private String accessToken;
  private String refreshToken;
  private int accessTokenLifetimeMinutes;
  private int refreshTokenLifetimeMinutes;
  public AuthenticationResponse(String accessToken, String refreshToken, int
accessTokenLifetimeMinutes, int refreshTokenLifetimeMinutes) {
    this.accessToken = accessToken;
    this.refreshToken = refreshToken;
```

```
    this.accessTokenLifetimeMinutes = accessTokenLifetimeMinutes;
    this.refreshTokenLifetimeMinutes = refreshTokenLifetimeMinutes;
  }
  public int getAccessTokenLifetimeMinutes() {
    return accessTokenLifetimeMinutes;
  }
  public void setAccessTokenLifetimeMinutes( int accessTokenLifetimeMinutes ) {
    this.accessTokenLifetimeMinutes = accessTokenLifetimeMinutes;
  }
  public int getRefreshTokenLifetimeMinutes() {
    return refreshTokenLifetimeMinutes;
  }
  public void setRefreshTokenLifetimeMinutes( int refreshTokenLifetimeMinutes ) {
    this.refreshTokenLifetimeMinutes = refreshTokenLifetimeMinutes;
  }
  public String getAccessToken() {
    return accessToken;
  }
  public void setAccessToken(String accessToken) {
    this.accessToken = accessToken;
  }
  public String getRefreshToken() {
    return refreshToken;
  }
  public void setRefreshToken(String refreshToken) {
    this.refreshToken = refreshToken;
  }
}
package com.voltor.futureleave.api.v1;

public final class BaseControllerUtil {
  private static final Logger LOG =
LoggerFactory.getLogger(BaseControllerUtil.class);
  private BaseControllerUtil() {
  }
  public static <T, ID> T getObjectOrNotFound(T obj, ID id, Class<T> relClass) {
    if (obj == null) {
      throw LocalizedExceptionUtil.buildObjectNotFoundException(relClass, id);
    }
    return obj;
  }
  public static <T, ID> T getObjectOrNotFound(T obj, ID id, Class<T> relClass,
Class<?> onClass) {
    if (obj == null) {
      final String message = String.format("Failed to link %s (ID=%s) on %s",
relClass.getSimpleName(), id, onClass.getSimpleName());
      LOG.warn(message);
      throw LocalizedExceptionUtil.buildObjectNotFoundException(relClass, id,
message);
    }
    return obj;
  }

  public static List<String> parseExpandField(Optional<String> expandFields) {
    return expandFields.map(s ->
Arrays.asList(s.split(","))).orElse(Collections.emptyList());
  }
  public static void checkPatchRequest(Map<String, Object> request) {
    if (request.isEmpty()) {
      throw new MissingPropertyException(new LocalizedMessage(
          "Request object on PATCH is empty.",
```

```
          LocalizedExceptionCode.MISSING_PROPERTY_EXCEPTION_PATCH_BODY));
    }
  }
}
package com.voltor.futureleave.api.v1.common;

@Validated
@SecurityRequirement(name = "Bearer Authentication")
public abstract class AbstractController<
  T extends Identifiable,
  RequestType extends AbstractRequest<T>,
  ApiResponseType extends AbstractResponse>
    implements FilterableController<T> {

  public static final Integer DEFAULT_PAGE_SIZE = 25;
  public static final Integer DEFAULT_PAGE_INDEX = 0;
  private static final Logger LOGGER =
LoggerFactory.getLogger(AbstractController.class);
  private static final Pattern PATTERN =
Pattern.compile("(\\w+?)(:|!_=|[!<>_]=?|=)(.*)");
  public abstract AbstractService<T> getService();
  public abstract ApiResponseType convertEntityToResponse(T entity, List<String>
entitiesToExpand);

  @Autowired
  protected ConversionService conversionService;

  public abstract Class<T> getEntityClass();
  @Operation(summary = "Retrieve a list of all records")
  @GetMapping(produces = { MediaType.APPLICATION_JSON_VALUE })
  public ResponseEntity< PageResponse< ApiResponseType > >getAll(

      @Parameter( description = "Page number of the requested page. The default
value is 0", example = "0" )
      @RequestParam( required = false )
      Optional< @PositiveOrZero( message = "Page index must have a positive or
zero value" ) Integer> index,

      @Parameter( description = "The size of the requested page. The default value
is 25. Maximum size is 100", example = "25")
      @RequestParam( required = false )
      Optional< @Range( min = 1, max = 100, message = "Page size must be between 1
and 100 inclusive" ) Integer > size,

      @Parameter( description = "Search expression used to filter results. "
          + "To check fields which support filtering, please reffer to the
response model",
          example = "field1=value1,field2!=value2,field3:value3" )
      @RequestParam( required = false ) Optional<String> search,

      @Parameter( description = "Specify fields to sort. The default sorting is
ascending" )
      @PathParam( "sort" ) Sort sort,

      @Parameter(name = "expand_fields", description = "Specify fields which have
to be expanded in the response. "
            + "To check fields which support expanding, please refer to the
response model",
            example = "field1,field2" )
      @RequestParam(value = "expand_fields", required = false) Optional<String>
expand ) {
```

```
    int pageSize = size.orElse( DEFAULT_PAGE_SIZE );
    Specification<T> filteringSpecification =
buildDefaultGetAllFilteringSpec(search);
    sort = mapSortPropertities( sort );
    int pageIndex = index.orElse( DEFAULT_PAGE_INDEX );
    Pageable pageable = PageRequest.of(pageIndex, pageSize, sort);
    Page<T> entitiesPaged = getService().get(filteringSpecification, pageable);
    List<ApiResponseType> responses = entitiesPaged.getContent().stream()
        .map( item -> convertEntityToResponse(item,
BaseControllerUtil.parseExpandField(expand))).collect(Collectors.toList());
    PageResponse< ApiResponseType > apiResponseTypePageResponse =
        new PageResponse<>(responses, entitiesPaged.getTotalElements(),
entitiesPaged.getNumber(), entitiesPaged.getSize());
    return ResponseEntity.ok(apiResponseTypePageResponse);
  }

  protected Sort mapSortPropertities( Sort sort ) {
    if ( sort == null ) {
      return Sort.unsorted();
    }
    Iterator< Order > iterator = sort.iterator();
    List< Order > mappedOrders = new LinkedList<>();
    while ( iterator.hasNext() ) {
      Order order = iterator.next();
      Order mappedOrder = new Order(
          order.getDirection(),
          mapDBProperty( order.getProperty() ),
          order.getNullHandling() );
      mappedOrders.add( mappedOrder );
    }
    return Sort.by( mappedOrders );
  }
  protected String mapDBProperty( String property) {
    switch ( property ) {
        case "bcId":
          return "externalId";
        default:
          return property;
    }

  }
  @GetMapping(path = "/{id}", produces = {MediaType.APPLICATION_JSON_VALUE})
  @Operation(summary = "Retrieve a specific record by id",
      description = "This operation retrieves specific record of current entity
from database\n" +
          "- <b>expand_fields</b> - is used to specify the entity fields that need
to be included in the response.\n" +
          "- <b>id</b> - is used to specify the identification number of record
which must be returned.")
  @ResponseBody
  public ApiResponseType getRecord(@PathVariable(value = "id") Long id,
                   @Parameter(name = "expand_fields", description = "Specify
fields which has to be expanded " +
                       "in response. To check fields which support expanding,
please refer to response model")
                   @RequestParam(value = "expand_fields", required = false)
Optional<String> expand,
                   HttpServletResponse response,
                   HttpServletRequest servletRequest) {
    T entity = getService().getOne( id  );
    BaseControllerUtil.getObjectOrNotFound( entity, id, getEntityClass() );
```

```
    return convertEntityToResponse( entity, BaseControllerUtil.parseExpandField(
expand ) );
  }
  @PostMapping(
      consumes = {MediaType.APPLICATION_JSON_VALUE},
      produces = {MediaType.APPLICATION_JSON_VALUE})
  @Operation(summary = "Create record")
  @ResponseBody
  @ResponseStatus( HttpStatus.CREATED )
  public ApiResponseType createRecord(@Validated({CreateValidationGroup.class,
Default.class}) @RequestBody RequestType request,
                    HttpServletResponse response) {

    /* TODO: Why the create action is the responsibility of the request object? */
    T entity = request.createEntity();
    entity = executeEntityCreate(entity, request);
    return convertEntityToResponse(entity, expandFieldsOnCreateAndUpdate());
  }
  @PutMapping(path = "/{id}",
      consumes = {MediaType.APPLICATION_JSON_VALUE},
      produces = {MediaType.APPLICATION_JSON_VALUE})
  @Operation(summary = "Update a specific record, based on the id",
    description = "Update a specific record, based on the id<br>"
        + "All values will be replaced by the input data")
  @ApiResponses(value = {
      @ApiResponse(responseCode = "404", description = "The record to be updated
does not exist."),
      @ApiResponse(responseCode = "406", description = "Invalid property value")
  })
  @ResponseBody
  @ResponseStatus( HttpStatus.OK )
  public ApiResponseType updateRecord(@PathVariable(value = "id") Long id,
                    @Validated({UpdateValidationGroup.class, Default.class})
@RequestBody RequestType request,
                    HttpServletResponse response) {
    T entity = getByID( id );

    request.updateEntity( entity );
    entity = executeEntityUpdate( entity, request);

    return convertEntityToResponse(entity, expandFieldsOnCreateAndUpdate());
  }
  protected T executeEntityUpdate( T entity, RequestType request) {
    return getService().update(entity);
  }
  /* TODO: is the request needed? */
  protected T executeEntityCreate(T entity, RequestType request) {
    return getService().create(entity);
  }

  protected T executeEntityPatch( T entity, Map<String, Object> request) {
    return getService().update(entity);
  }

  @PatchMapping(path = "/{id}",
      consumes = {MediaType.APPLICATION_JSON_VALUE},
      produces = {MediaType.APPLICATION_JSON_VALUE})
  @Operation(summary = "Patch a specific record, based on the id")
  @ApiResponses(value = {
      @ApiResponse(responseCode = "404", description = "The request was empty"),
```

```
      @ApiResponse(responseCode = "404", description = "The record to be patched
does not exist."),
      @ApiResponse(responseCode = "405", description = "The PATCH method is not
implemented for this endpoint.")
  })
  @ResponseBody
  public ApiResponseType patchRecord(
      @PathVariable(value = "id") Long id, @RequestBody Map<String, Object>
request,
      HttpServletResponse response) {
    BaseControllerUtil.checkPatchRequest(request);
    T entity = getByID( id );

    patchFields(entity, request);
    entity = executeEntityPatch( entity, request );
    return convertEntityToResponse(entity, Collections.emptyList());
  }
  /**
   * Method to be overridden by implementing Controller if the Http PATCH method
is allowed.
   * <p>
   * Abstract implementation throws an unchecked {@link
HttpRequestMethodNotSupportedException} which results in Http status code 405
(Method Not Allowed).
   * It is only allowed to PATCH simple fields, that means no fields that are a
reference to a other entity.
   *
   * @param entity        The entity in which the updates need to be done.
   * @param fieldsToPatch A Map of the JSON fields in the request. These values
should be updated in the entity.
   */
  protected void patchFields(T entity, Map<String, Object> fieldsToPatch) {
    throw
LocalizedExceptionUtil.buildHttpRequestMethodNotSupportedException("PATCH");
  }
  @Operation( summary = "Delete record by id" )
  @DeleteMapping(path = "/{id}", produces = {MediaType.APPLICATION_JSON_VALUE})
  @ResponseBody
  @ResponseStatus(HttpStatus.NO_CONTENT)
  public void deleteRecord(@PathVariable(value = "id") Long id,
              HttpServletResponse response) {
    T entity = getByID( id );
    getService().delete(entity.getId());
  }

  public List<String> expandFieldsOnCreateAndUpdate() {
    return Collections.emptyList();
  }
  protected Specification<T> buildDefaultGetAllFilteringSpec(Optional<String>
search) {
    if (search.isPresent() && this.getFilterSpecificationsBuilder() != null) {
      List< FilterableProperty< T > > filterableProperties =
this.getFilterSpecificationsBuilder().getFilterableProperties();
      List< SearchCriteria > searchCriteria = parseSearchCriteria( search.get(),
filterableProperties );
      return
this.getFilterSpecificationsBuilder().buildSpecification(searchCriteria);
    }
    return null;
  }
```

```
  protected List< SearchCriteria > parseSearchCriteria( String searchQuery, List<
FilterableProperty< T > > filterableProperties ) {
    String[] searchParams = searchQuery.split(",");
    List<SearchCriteria> searchCriteria = new ArrayList<>();
    for (String searchParameter : searchParams) {
      Matcher matcher = PATTERN.matcher(searchParameter);
      while (matcher.find()) {
        String key = mapDBProperty( matcher.group(1) );
        String operationStr = matcher.group(2);
        FilteringOperation operation =
FilteringOperation.fromString(operationStr);
        String value = matcher.group(3);
        Optional< FilterableProperty< T > > filterableProperty =
filterableProperties.stream()
            .filter( property -> property.getPropertyName().equals( key )
).findFirst();
        if (filterableProperty.isPresent()) {

          Object convertedValue;
          if ("null".equals(value) || StringUtils.isEmpty(value)) {
            convertedValue = null;
          } else {
            convertedValue = convertValueForCriteria( key, operation, value,
filterableProperty.get() );
          }
          // check if a FilterableOperation is supported
          if (!filterableProperty.get().getOperators().contains(operation)) {
            throw new IllegalFilteringOperationException("Operation '" + operation
+ "' is not supported for property " + key);
          }
          searchCriteria.add(new SearchCriteria(key, operation, convertedValue));
        } else {
          LOGGER.warn("Filtering on property '{}' has been skipped because it's
absent in filterableProperties", key);
        }
      }
    }
    return searchCriteria;
  }

  protected Object convertValueForCriteria( String key, FilteringOperation
operation, String value, FilterableProperty< T > filterableProperty ) {
    return conversionService.convert( value, filterableProperty.getExpectedType()
);
  }
  private T getByID( Long id ) {
    return BaseControllerUtil.getObjectOrNotFound(getService().getOne( id ), id,
getEntityClass());
  }
}


package com.voltor.futureleave.api.v1.common;

public abstract class AbstractEntityResponseBuilder<T extends Identifiable, R
extends AbstractResponse> implements EntityResponseBuilder<T, R> {
  @Override
  public R convertEntityToResponse(T entity) {
    return convertEntityToResponse(entity, null);
  }
  @Override
```

```
  public R convertEntityToResponse(T entity, List<String> entitiesToExpand) {
    if (entity == null) return null;
    try {
      Class< R > responseTypeClass = getResponseTypeClass();
      R response = responseTypeClass.getConstructor().newInstance();
      mapFields(response, entity);
      expandEntities(response, entity, entitiesToExpand);
      return response;
    } catch ( IllegalArgumentException
        | InvocationTargetException
        | NoSuchMethodException
        | SecurityException e ) {
      throw new InternalServerException(new LocalizedMessage("Error creating
response instance of type " + entity.getClass(),
LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION));
    } catch ( InstantiationException | IllegalAccessException e ) {
      throw new InternalServerException(new LocalizedMessage("Error creating
response instance of type " + entity.getClass(),
LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION));
    }
  }
  protected abstract void mapFields(R response, T entity);
  protected abstract void buildLinks(R response, T entity);
  protected abstract void expandEntities(R response, T entity, List<String>
entitiesToExpand);
  public <S> void expandEntity(S entity, List<String> entitiesToExpand, String
expandFieldName, Consumer<List<String>> consumer) {
    if (entitiesToExpand == null || entitiesToExpand.isEmpty()) return;
    if (expandFieldName == null) return;
    if (entity == null) return;
    boolean found = entitiesToExpand.stream().anyMatch(s ->
s.startsWith(expandFieldName));
    if (found) {
      consumer.accept(stripBaseFromExpandFields(entitiesToExpand,
expandFieldName));
    }
  }
  private List<String> stripBaseFromExpandFields(List<String> expandFields, String
base) {
    if (expandFields == null) return null;
    if (StringUtils.isEmpty(base)) return expandFields;
    return expandFields.stream()
        .filter(s -> s.startsWith(base + "."))
        .map(s -> s.replaceFirst("^" + base + ".", ""))
        .filter(StringUtils::isNotEmpty)
        .collect(Collectors.toList());
  }
  @SuppressWarnings("unchecked")
  private Class<R> getResponseTypeClass() {
    Class<?>[] classes = GenericTypeResolver.resolveTypeArguments(getClass(),
AbstractEntityResponseBuilder.class);
    return (Class<R>) classes[1];
  }
}
package com.voltor.futureleave.api.v1.common;
public abstract class AbstractRequest<EntityType> {
  public abstract EntityType createEntity();
  public abstract EntityType updateEntity(EntityType entity);
}
package com.voltor.futureleave.api.v1.common;
public abstract class AbstractResponse {
```

```
  @Schema(
      description = "The id of the record<br/>"
          + "Sortable",
      example = "228322" )
  private Long id;
  public AbstractResponse() {
  }
  public AbstractResponse( Long id ) {
    this.id = id;
  }
  public void setId( final Long id ) {
    this.id = id;
  }
  public Long getId() {
    return this.id;
  }
  @Override
  public int hashCode() {
    return Objects.hash( id );
  }
  @Override
  public boolean equals( Object obj ) {
    if ( this == obj )
      return true;
    if ( obj == null )
      return false;
    if ( getClass() != obj.getClass() )
      return false;
    AbstractResponse other = (AbstractResponse) obj;
    return Objects.equals( id, other.id );
  }

}


package com.voltor.futureleave.api.v1.common;
public interface CreateValidationGroup {
}
package com.voltor.futureleave.api.v1.common;

public interface EntityResponseBuilder<T extends Identifiable, R extends
AbstractResponse> {
  R convertEntityToResponse(T entity);
  R convertEntityToResponse(T entity, List<String> entitiesToExpand);
}
package com.voltor.futureleave.api.v1.common;
public interface FilterableController<T> {
  default EntityFilterSpecificationsBuilder<T> getFilterSpecificationsBuilder() {
    return null;
  }
}
package com.voltor.futureleave.api.v1.common;
public class PageResponse<T> {
  private long totalElements;
  private long index;
  private long size;
  private List< T > items;
  public PageResponse() {
    //it is needed for JSON parsing
  }
```

```
  public PageResponse( List< T > items, long totalElements, long index, long size)
{
    this.totalElements = totalElements;
    this.items = items;
    this.index = index;
    this.size = size;
  }
  public void setTotalElements(long totalElements) {
    this.totalElements = totalElements;
  }
  public void setItems(List< T > items) {
    this.items = items;
  }
  public long getTotalElements() {
    return totalElements;
  }
  public List< T > getItems() {
    return items;
  }
  public long getIndex() {
    return index;
  }
  public void setIndex(long index) {
    this.index = index;
  }
  public long getSize() {
    return size;
  }
  public void setSize(long size) {
    this.size = size;
  }
}
package com.voltor.futureleave.api.v1.common;
public class UnpagedPage implements Pageable, Serializable {
  private static final long serialVersionUID = 1232825578694716871L;
  private final Sort sort;
  public UnpagedPage(Sort sort ) {
    Assert.notNull(sort, "Sort must not be null!");
    this.sort = sort;
  }
  public Sort getSort() {
    return this.sort;
  }
  @Override
  public boolean isPaged() {
    return false;
  }
  public Pageable previousOrFirst() {
    return this;
  }
  public Pageable next() {
    return this;
  }
  public boolean hasPrevious() {
    return false;
  }
  public int getPageSize() {
    throw new UnsupportedOperationException();
  }
  public int getPageNumber() {
    throw new UnsupportedOperationException();
```

```
  }
  public long getOffset() {
    throw new UnsupportedOperationException();
  }
  public Pageable first() {
    return this;
  }
  @Override
  public Pageable withPage(int pageNumber)  {
    throw new UnsupportedOperationException();
  }
  public boolean equals(@Nullable Object obj) {
    if (this == obj) {
      return true;
    } else if (!(obj instanceof UnpagedPage)) {
      return false;
    } else {
      UnpagedPage that = (UnpagedPage)obj;
      return super.equals(that) && this.sort.equals(that.sort);
    }
  }
  public int hashCode() {
    return 31 * super.hashCode() + this.sort.hashCode();
  }
  public String toString() {
    return String.format("Page request [sort: %s]", this.sort);
  }
}
package com.voltor.futureleave.api.v1.common;
public interface UpdateValidationGroup {
}
package com.voltor.futureleave.api.v1.exception;


@ControllerAdvice(basePackages = {
    "com.voltor.futureleave.api"
})
public class ExceptionsHandler extends ResponseEntityExceptionHandler {
  private static final Logger LOGGER =
LoggerFactory.getLogger(ExceptionsHandler.class);
  private static final Map<Class<? extends Exception>, HttpStatus> STATUSES = new
HashMap<>();
  /** In order to your class be handled, you should add it to STATUSES **/
  static {
    STATUSES.put(HttpRequestMethodNotSupportedException.class,
HttpStatus.METHOD_NOT_ALLOWED);
    STATUSES.put(UnsupportedActionException.class, HttpStatus.METHOD_NOT_ALLOWED);
    STATUSES.put(UserAlreadyExistsException.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(RetrievalNotAllowedException.class, HttpStatus.FORBIDDEN);
    STATUSES.put(ConstraintValidationException.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(ObjectNotFoundException.class, HttpStatus.NOT_FOUND);
    STATUSES.put(MaxCountValidationException.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(RemovalNotAllowedException.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(InvalidConfirmationToken.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(ConfirmationTokenExpired.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(ActionNotAllowedException.class, HttpStatus.NOT_ACCEPTABLE);
    STATUSES.put(RefreshTokenNotFoundException.class, HttpStatus.UNAUTHORIZED);
    STATUSES.put(UpdateNotAllowedException.class, HttpStatus.METHOD_NOT_ALLOWED);
    STATUSES.put(ParseDatePatternException.class, HttpStatus.NOT_ACCEPTABLE);
  }
```

```
  @ExceptionHandler(value = DuplicatedValueException.class)
  protected ResponseEntity<Object>
handleDuplicatedValueException(DuplicatedValueException ex, WebRequest request) {
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.BAD_REQUEST);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler(value = {IllegalFilteringOperationException.class})
  protected ResponseEntity<Object>
handleIllegalFilteringOperationException(IllegalFilteringOperationException ex,
WebRequest request) {
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.BAD_REQUEST);
    return handleExceptionInternal(ex, error, request);
  }

  @ExceptionHandler(value = {IOException.class})
  protected ResponseEntity<Object> handleIOException(IOException ex, WebRequest
request) {
    if (StringUtils.containsIgnoreCase(ExceptionUtils.getRootCauseMessage(ex),
"Broken pipe")) {
      return null;
    } else {
      final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.INTERNAL_SERVER_ERROR);
      return handleExceptionInternal(ex, error, request);
    }
  }
  @ExceptionHandler(value = {MultipartException.class})
  protected ResponseEntity<Object> handleMultipartException(MultipartException ex,
WebRequest request) {
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.INTERNAL_SERVER_ERROR);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler(value = {ConstraintViolationException.class})
  protected ResponseEntity<Object>
handleValidationFailedException(ConstraintViolationException ex, WebRequest
request) {
    final String details =
ex.getConstraintViolations().stream().map(ConstraintViolation::getMessage).collect
(Collectors.joining(","));
    final String defaultMessage = String.format("Validation failed, %s", details);
    final LocalizedMessage localizedMessage = new LocalizedMessage(defaultMessage,
LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION);
    final LocalizedApiError error = new LocalizedApiError(localizedMessage,
HttpStatus.NOT_ACCEPTABLE, ex.getClass().getSimpleName());
    return handleExceptionInternal(ex, error, new HttpHeaders(),
error.getStatus(), request);
  }
  @ExceptionHandler(value = {IllegalArgumentException.class})
  protected ResponseEntity<Object>
handleIllegalArgumentException(IllegalArgumentException ex, WebRequest request) {
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.INTERNAL_SERVER_ERROR);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler(value = {IllegalStateException.class})
  protected ResponseEntity<Object>
handleIllegalStateException(IllegalStateException ex, WebRequest request) {
```

```
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.INTERNAL_SERVER_ERROR);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler(value = {LocalizedException.class})
  protected ResponseEntity<Object> handleLocalizedException(LocalizedException ex,
WebRequest request) {
    final LocalizedApiError error = new LocalizedApiError(ex,
STATUSES.getOrDefault(ex.getClass(),
        HttpStatus.INTERNAL_SERVER_ERROR));
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler( value = {BadCredentialsException.class})
  protected void handleBadCredentialsException(BadCredentialsException ex,
HttpServletResponse response) throws IOException {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
  }
  @ExceptionHandler( value = {DateTimeParseException.class})
  protected ResponseEntity<Object>
handleDateTimeParseException(DateTimeParseException ex,  WebRequest request) {
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.UNPROCESSABLE_ENTITY);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler( value = {JwtExpirationException.class,
JwtBadSignatureException.class, MalformedJwtException.class})
  protected void handleJWTVerificationException(RuntimeException ex,
HttpServletResponse response) throws IOException {
     response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
  }
  @ExceptionHandler(value = {FileNotFoundException.class})
  protected void handleFileNotFoundException(FileNotFoundException ex,
HttpServletResponse response) throws IOException {
    response.sendError(HttpServletResponse.SC_NOT_FOUND, ex.getMessage());
  }
  @ExceptionHandler(value = {JobExecutionAlreadyRunningException.class})
  protected ResponseEntity<Object> handleJobExecutionAlreadyRunningException(
      JobExecutionAlreadyRunningException ex, WebRequest request) {
    final LocalizedApiError error = buildErrorOccurred(ex, HttpStatus.CONFLICT);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler(value = {Exception.class})
  protected ResponseEntity<Object> handleCommonException(Exception ex, WebRequest
request) {
    final LocalizedMessage localizedMessage = new LocalizedMessage("Error
occurred.", LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION);
    LocalizedApiError error = new LocalizedApiError(localizedMessage,
        HttpStatus.INTERNAL_SERVER_ERROR,
        ex.getClass().getSimpleName());
    return handleExceptionInternal(ex, error, request);
  }

  @ExceptionHandler(value = {PatchFieldConstraintViolationException.class})
  protected ResponseEntity<Object>
handlePatchFieldConstraintViolationException(PatchFieldConstraintViolationExceptio
n ex, WebRequest request) {
    final StringJoiner stringBuilder = new StringJoiner(". ");
    stringBuilder.add("Error occurred");
    for (ConstraintViolation<?> constraintViolation :
ex.getConstraintViolations()) {
```

```
      final String defaultMessage = String.format("%s: %s",
constraintViolation.getPropertyPath(), constraintViolation.getMessage());
      stringBuilder.add(defaultMessage);
    }
    final LocalizedMessage localizedMessage = new
LocalizedMessage(stringBuilder.toString(),
LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION);
    final LocalizedApiError error = new LocalizedApiError(localizedMessage,
HttpStatus.NOT_ACCEPTABLE, ex.getClass().getSimpleName());
    return handleExceptionInternal(ex, error, new HttpHeaders(),
error.getStatus(), request);
  }
  @Override
  protected ResponseEntity<Object>
handleHttpMessageNotReadable(HttpMessageNotReadableException ex,
                                    HttpHeaders headers, HttpStatusCode status,
WebRequest request) {
    LOGGER.error("Failed to process the HTTP request {}", ex.getMessage());
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.UNSUPPORTED_MEDIA_TYPE);
    return handleExceptionInternal(ex, error, request);
  }


  @Override
  protected ResponseEntity<Object>
handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
                                    HttpHeaders headers,
                                    HttpStatusCode status,
                                    WebRequest request) {
    final StringJoiner stringBuilder = new StringJoiner(". ");
    stringBuilder.add("Error occurred");
    for (FieldError error : ex.getBindingResult().getFieldErrors()) {
      final String defaultMessage = String.format("%s: %s", error.getField(),
error.getDefaultMessage());
      stringBuilder.add(defaultMessage);
    }
    for (ObjectError error : ex.getBindingResult().getGlobalErrors()) {
      final String defaultMessage = String.format("%s: %s", error.getObjectName(),
error.getDefaultMessage());
      stringBuilder.add(defaultMessage);
    }
    final LocalizedMessage localizedMessage = new
LocalizedMessage(stringBuilder.toString(),
LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION);
    final LocalizedApiError error = new LocalizedApiError(localizedMessage,
status, ex.getClass().getSimpleName());
    return handleExceptionInternal(ex, error, headers, status, request);
  }
  private ResponseEntity<Object> handleExceptionInternal(Exception ex,
LocalizedApiError error, WebRequest request) {
    final HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "application/json");
    return handleExceptionInternal(ex, error, headers, error.getStatus(),
request);
  }
  private LocalizedApiError buildErrorOccurred(Exception ex, HttpStatus status) {
    final String defaultDescription = String.format("Error occurred. %s",
ex.getMessage());
```

```
    final LocalizedMessage localizedMessage = new
LocalizedMessage(defaultDescription,
LocalizedExceptionCode.ERROR_OCCURRED_EXCEPTION);
    return new LocalizedApiError(localizedMessage, status,
ex.getClass().getSimpleName());
  }
  @Override
  protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
                                Object body, HttpHeaders headers, HttpStatusCode
status, WebRequest request) {
    LOGGER.warn("Exception occurred:", ex);
    return super.handleExceptionInternal(ex, body, headers, status, request);
  }
  @ExceptionHandler(value = {NoCurrentUserException.class})
  protected ResponseEntity<Object>
handleNoCurrentUserExceptionException(NoCurrentUserException ex, WebRequest
request) {
    final LocalizedApiError error = buildErrorOccurred(ex,
HttpStatus.UNAUTHORIZED);
    return handleExceptionInternal(ex, error, request);
  }
  @ExceptionHandler(value = {ClientAbortException.class})
  public void handleClientAbortException(ClientAbortException exception,
HttpServletRequest request) {
    final String message = "ClientAbortException generated by request {} {} from
remote address {} ";
    LOGGER.warn(message, request.getMethod(), request.getRequestURL(),
request.getRemoteAddr());
  }
}

  public static HttpRequestMethodNotSupportedException
buildHttpRequestMethodNotSupportedException(String method) {
    final String defaultErrorMessage = String.format("%s method is not
supported.", method);
    final HashMap<String, String> messageArguments = new HashMap<>();
    messageArguments.put("method", method);
    final LocalizedMessage message = new LocalizedMessage(defaultErrorMessage,
LocalizedExceptionCode.METHOD_NOT_SUPPORTED_MESSAGE);
    message.setMessageArguments(messageArguments);
    return new HttpRequestMethodNotSupportedException(message);
  }
  public static ObjectNotFoundException buildObjectsNotFoundException(
      Class<?> relClass, List<Long> values, String detailMessage) {
    String valueDetails = null;
    if( values != null  ){
      values.sort(Comparator.comparingLong( item -> item ));
      valueDetails =  StringUtils.join(values, ", ");
    }
    return buildObjectNotFoundException(relClass, valueDetails, detailMessage);
  }
  public static ObjectNotFoundException buildObjectNotFoundException(
      Class<?> relClass, Object id, String detailMessage) {
    String valueDetails =  id == null ? null : id.toString();
    return buildObjectNotFoundException(relClass, valueDetails, detailMessage);
  }
  public static ObjectNotFoundException buildObjectNotFoundException(
      Class<?> relClass, String valueDetails, String detailMessage) {
    final String defaultMessage = String.format("Requested object (%s) with
identifier (%s) was not found. %s",
```

```
        relClass.getSimpleName(), valueDetails, detailMessage != null ?
detailMessage : "");
    final HashMap<String, String> messageArguments = new HashMap<>();
    messageArguments.put("identifier", valueDetails);
    final HashMap<String, String> localizedMessageArguments = new HashMap<>();
    localizedMessageArguments.put("entity",
relClass.getSimpleName().toUpperCase());
    final LocalizedMessage localizedMessage = new LocalizedMessage(defaultMessage,
LocalizedExceptionCode.OBJECT_NOT_FOUND_EXCEPTION);
    localizedMessage.setLocalizedMessageArguments(localizedMessageArguments);
    localizedMessage.setMessageArguments(messageArguments);
    return new ObjectNotFoundException(localizedMessage);
  }
  public static ObjectNotFoundException buildObjectNotFoundException(Class<?>
relClass, Object id) {
    return buildObjectNotFoundException(relClass, id, null);
  }
}
package com.voltor.futureleave.api.v1.period;

@Tag( name = "Period" )
@RestController
@RequestMapping( ApiConstants.V1_PERIOD_ENDPOINT )
public class PeriodController extends AbstractController<Period, PeriodRequest,
PeriodResponse>  {

  @Autowired
  private PeriodSpecificationBuilder specificationBuilder;

  @Autowired
  private PeriodService service;

  @Autowired
  private Trainer trainer;

  private Validator validator;

  public PeriodController( ValidatorFactory validatorFactory ) {
    validator = validatorFactory.getValidator();
  }

  @Override
  public AbstractService<Period> getService() {
    return service;
  }
  @Override
  public Class<Period> getEntityClass() {
    return Period.class;
  }
  @Override
  public PeriodResponse convertEntityToResponse( Period period, List<String>
entitiesToExpand ) {
    PeriodResponse periodResponse = new PeriodResponse( period );
    periodResponse.setSessionId( period.getSessionId() );
    periodResponse.setStartDate( period.getStartDate() );
    periodResponse.setEndDate( period.getEndDate() );
    periodResponse.setPlannedEventsCount( period.getPlannedEventsCount() );
    periodResponse.setUnplannedEventsCount( period.getUnplannedEventsCount() );
    return periodResponse;
  }
```

```
  @Override
  public EntityFilterSpecificationsBuilder<Period>
getFilterSpecificationsBuilder() {
    return specificationBuilder;
  }

  @Override
  protected void patchFields( Period entity, Map< String, Object > request ) {
    request.entrySet().forEach( requestedField -> patchField( entity,
requestedField ) );
  }

  protected void patchField( Period entity, Entry< String, Object > requestedField
) {
    switch( requestedField.getKey() ) {
      case "startDate":
        LocalDate startDate = conversionService.convert(
requestedField.getValue(), LocalDate.class );
        validatePatchField( "startDate", startDate );
        entity.setStartDate(startDate);
        return;
      case "endDate":
        LocalDate endDate = conversionService.convert( requestedField.getValue(),
LocalDate.class );
        validatePatchField( "endDate", endDate );
        entity.setEndDate(endDate);
        return;
      case "plannedEventsCount":
        Integer plannedEventsCount = conversionService.convert(
requestedField.getValue(), Integer.class );
        validatePatchField( "plannedEventsCount", plannedEventsCount );
        entity.setPlannedEventsCount(plannedEventsCount);
        return;
      case "unplannedEventsCount":
        Integer unplannedEventsCount = conversionService.convert(
requestedField.getValue(), Integer.class );
        validatePatchField( "unplannedEventsCount", unplannedEventsCount );
        entity.setUnplannedEventsCount(unplannedEventsCount);
        return;
        default:
          return;
      }
  }

  private void validatePatchField( String propertyName, Object value ) {
    Set< ConstraintViolation< PeriodRequest > > validationResult = validator
        .validateValue( PeriodRequest.class, propertyName, value,
UpdateValidationGroup.class, Default.class );
    if(validationResult.isEmpty()) {
      return;
    }
    throw new PatchFieldConstraintViolationException( validationResult );
  }

  @PostMapping( path = "/predict",
      consumes = {MediaType.APPLICATION_JSON_VALUE},
      produces = {MediaType.APPLICATION_JSON_VALUE})
  @Operation(summary = "Create record")
  @ResponseBody
  @ResponseStatus( HttpStatus.OK )
  public PeriodRequest createRecord( @RequestBody PeriodRequest request ) {
```

```
    request.setUnplannedEventsCount(  Integer.valueOf(
trainer.getPredict(request.createEntity() )  ));
    return request;
  }
}package com.voltor.futureleave.api.v1.period;
public class PeriodRequest extends AbstractRequest<Period> {
  private long sessionId;
  private LocalDate startDate;
  private LocalDate endDate;
  private int plannedEventsCount;
  private int unplannedEventsCount;

  @Override
  public Period createEntity() {
    Period period = new Period();
    period.setSessionId(sessionId);
    return updateEntity(period);
  }

  @Override
  public Period updateEntity(Period entity) {
    entity.setStartDate(startDate);
    entity.setEndDate(endDate);
    entity.setPlannedEventsCount(plannedEventsCount);
    entity.setUnplannedEventsCount(unplannedEventsCount);
    return entity;
  }
  public long getSessionId() {
    return sessionId;
  }
  public void setSessionId(long sessionId) {
    this.sessionId = sessionId;
  }
  public LocalDate getStartDate() {
    return startDate;
  }
  public void setStartDate(LocalDate startDate) {
    this.startDate = startDate;
  }
  public LocalDate getEndDate() {
    return endDate;
  }
  public void setEndDate(LocalDate endDate) {
    this.endDate = endDate;
  }
  public int getPlannedEventsCount() {
    return plannedEventsCount;
  }
  public void setPlannedEventsCount(int plannedEventsCount) {
    this.plannedEventsCount = plannedEventsCount;
  }
  public int getUnplannedEventsCount() {
    return unplannedEventsCount;
  }
  public void setUnplannedEventsCount(int unplannedEventsCount) {
    this.unplannedEventsCount = unplannedEventsCount;
  }
}


package com.voltor.futureleave.api.v1.period;
```

```
public class PeriodResponse extends AbstractResponse {

  private long sessionId;
  private LocalDate startDate;
  private LocalDate endDate;
  private int plannedEventsCount;
  private int unplannedEventsCount;

  public PeriodResponse( Period entity) {
    super( entity.getId() );
  }

  public long getSessionId() {
    return sessionId;
  }
  public void setSessionId(long sessionId) {
    this.sessionId = sessionId;
  }
  public LocalDate getStartDate() {
    return startDate;
  }
  public void setStartDate(LocalDate startDate) {
    this.startDate = startDate;
  }
  public LocalDate getEndDate() {
    return endDate;
  }
  public void setEndDate(LocalDate endDate) {
    this.endDate = endDate;
  }
  public int getPlannedEventsCount() {
    return plannedEventsCount;
  }
  public void setPlannedEventsCount(int plannedEventsCount) {
    this.plannedEventsCount = plannedEventsCount;
  }
  public int getUnplannedEventsCount() {
    return unplannedEventsCount;
  }
  public void setUnplannedEventsCount(int unplannedEventsCount) {
    this.unplannedEventsCount = unplannedEventsCount;
  }

}


package com.voltor.futureleave.api.v1.session;

@Tag( name = "Session" )
@RestController
@RequestMapping( ApiConstants.V1_SESSION_ENDPOINT )
public class SessionController extends AbstractController<Session, SessionRequest,
SessionResponse>  {

  @Autowired
  private SessionSpecificationBuilder specificationBuilder;

  @Autowired
  private SessionService service;

  private Validator validator;
```

```
  public SessionController( ValidatorFactory validatorFactory ) {
    validator = validatorFactory.getValidator();
  }

  public SessionResponse create() {
    return null;
  }
  @Override
  public AbstractService<Session> getService() {
    return service;
  }
  @Override
  public Class<Session> getEntityClass() {
    return Session.class;
  }
  @Override
  public SessionResponse convertEntityToResponse( Session session, List<String>
entitiesToExpand ) {
    SessionResponse sessionResponse = new SessionResponse( session );
    sessionResponse.setName( session.getName() );
    return sessionResponse;
  }

  @Override
  public EntityFilterSpecificationsBuilder<Session>
getFilterSpecificationsBuilder() {
    return specificationBuilder;
  }


  @Override
  protected void patchFields( Session entity, Map< String, Object > request ) {
    request.entrySet().forEach( requestedField -> patchField( entity,
requestedField ) );
  }

  protected void patchField( Session entity, Entry< String, Object >
requestedField ) {
    switch( requestedField.getKey() ) {
      case "name":
        String name = conversionService.convert( requestedField.getValue(),
String.class );
        validatePatchField( "name", name );
        entity.setName( name );
        return;
        default:
          return;
      }
  }

  private void validatePatchField( String propertyName, Object value ) {
    Set< ConstraintViolation< SessionRequest > > validationResult = validator
        .validateValue( SessionRequest.class, propertyName, value,
UpdateValidationGroup.class, Default.class );
    if(validationResult.isEmpty()) {
      return;
    }
    throw new PatchFieldConstraintViolationException( validationResult );
  }
}package com.voltor.futureleave.api.v1.session;
```

```
public class SessionRequest extends AbstractRequest<Session> {
  private String name;

  @Override
  public Session createEntity() {
    return updateEntity( new Session() );
  }
  @Override
  public Session updateEntity(Session entity) {
    entity.setName(name);
    return entity;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }

}


package com.voltor.futureleave.api.v1.session;
public class SessionResponse extends AbstractResponse {

  private String name;

  public SessionResponse(Session entity) {
    super( entity.getId() );
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}


package com.voltor.futureleave.dao;
@Service
public class PeriodDao extends AbstractIdentifiableDao<Period> {
  private final PeriodRepository periodRepository;
  @Autowired
  public PeriodDao(AuthenticatedUserService userAuthorizationService,
PeriodRepository periodRepository) {
    super(userAuthorizationService);
    this.periodRepository = periodRepository;
  }
  @Override
  protected BaseCRUDRepository<Period> getRepository() {
    return periodRepository;
  }

}


package com.voltor.futureleave.dao;

@Service
public class RefreshTokenDao extends AbstractIdentifiableDao<RefreshToken> {
```

```
  private final RefreshTokenRepository refreshTokenRepository;
  public RefreshTokenDao(AuthenticatedUserService userAuthorizationService,
              RefreshTokenRepository refreshTokenRepository) {
    super(userAuthorizationService);
    this.refreshTokenRepository = refreshTokenRepository;
  }
  @Override
  public boolean isEditAllowed(RefreshToken item) {
    return true;
  }

  @Override
  protected PrimaryRepository<Long, RefreshToken> getRepository() {
    return refreshTokenRepository;
  }

  @Override
  protected boolean entityIsExtendedByUser() {
    return false;
  }
}
package com.voltor.futureleave.dao;
@Service
public class SessionDao extends AbstractIdentifiableDao<Session> {
  private final SessionRepository sessionRepository;
  @Autowired
  public SessionDao(AuthenticatedUserService userAuthorizationService,
SessionRepository sessionRepository) {
    super(userAuthorizationService);
    this.sessionRepository = sessionRepository;
  }
  @Override
  protected BaseCRUDRepository<Session> getRepository() {
    return sessionRepository;
  }
  @Override
  public boolean isEditAllowed( Session entity ) {
    return userAuthorizationService.isRoot();
  }

}


package com.voltor.futureleave.dao.specification;

public class EqualSpecification<T> implements Specification<T> {

  private static final long serialVersionUID = -2550145329349526320L;
  private final String fieldPath;
  private final Object value;
  public EqualSpecification( String fieldPath, Object value ) {
    this.fieldPath = fieldPath;
    this.value = value;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    Path< T > path = SpecificationUtil.buildPath( root, fieldPath );
    return cb.equal( path, value );
  }
  @Override
```

```
  public int hashCode() {
    return Objects.hash( fieldPath, value );
  }
  @SuppressWarnings("rawtypes")
  @Override
  public boolean equals( Object obj ) {
    if ( this == obj )
      return true;
    if ( obj == null )
      return false;
    if ( getClass() != obj.getClass() )
      return false;
    EqualSpecification other = (EqualSpecification) obj;
    return Objects.equals( fieldPath, other.fieldPath ) && Objects.equals( value,
other.value );
  }
}
package com.voltor.futureleave.dao.specification;

public class GreaterThenSpecification<T, V extends Comparable<V> > implements
Specification<T> {

  private static final long serialVersionUID = -2550145329349526320L;
  private final String fieldPath;
  private final V value;
  public GreaterThenSpecification( String fieldPath, V value ) {
    this.fieldPath = fieldPath;
    this.value = value;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    Expression< V > searchExpression = root.get( fieldPath );
    return cb.greaterThan( searchExpression, value );
  }
  @Override
  public int hashCode() {
    return Objects.hash( fieldPath, value );
  }
  @SuppressWarnings("rawtypes")
  @Override
  public boolean equals( Object obj ) {
    if ( this == obj )
      return true;
    if ( obj == null )
      return false;
    if ( getClass() != obj.getClass() )
      return false;
    GreaterThenSpecification other = (GreaterThenSpecification) obj;
    return Objects.equals( fieldPath, other.fieldPath ) && Objects.equals( value,
other.value );
  }
}
package com.voltor.futureleave.dao.specification;

public class InCollectionSpecification<T, R extends Identifiable> implements
Specification<T> {
  private static final long serialVersionUID = 3657277566518014718L;
  private final Class<T> entityClass;
  private final String fieldPath;
  private final R value;
```

```
  public InCollectionSpecification( Class< T > entityClass, String fieldPath, R
value ) {
    this.entityClass = entityClass;
    this.fieldPath = fieldPath;
    this.value = value;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    Subquery< T > subQuery = query.subquery( entityClass );
    Root< T > sqRoot = subQuery.from( entityClass );
    Join< R, T > sqJoin = SpecificationUtil.buildJoin( sqRoot, fieldPath );
    subQuery.select( sqRoot );
    subQuery.where( cb.equal( sqJoin, value ) );
    return cb.in( root ).value( subQuery );
  }
}
package com.voltor.futureleave.dao.specification;


public class InSpecification<T extends Identifiable, R extends Serializable>
implements Specification<T> {
  private static final long serialVersionUID = 7652231116221930648L;
  private final String fieldPath;
  private final Collection< R > values;
  public InSpecification( String fieldPath, Collection< R > values ) {
    this.fieldPath = fieldPath;
    this.values = values;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > cq,
CriteriaBuilder cb ) {
    Path< T > path = SpecificationUtil.buildPath( root, fieldPath );
    return path.in( values );
  }
}
package com.voltor.futureleave.dao.specification;

@SuppressWarnings("serial")
public class NotEqualSpecification< T extends Identifiable > implements
Specification< T > {
  private final String fieldPath;
  private final Object value;
  public NotEqualSpecification( String fieldPath, Object value ) {
    this.fieldPath = fieldPath;
    this.value = value;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    Path< T > path = SpecificationUtil.buildPath( root, fieldPath );
    return cb.notEqual( path, value );
  }
}
package com.voltor.futureleave.dao.specification;

public class NullSpecification<T extends Identifiable> implements Specification<T>
{
  private static final long serialVersionUID = 8379712258475972114L;
  private final String fieldPath;
  public NullSpecification(String fieldPath) {
```

```
    this.fieldPath = fieldPath;
  }
  @Override
  public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {
    Path<T> path = SpecificationUtil.buildPath(root, fieldPath);
    return cb.isNull(path);
  }
}
package com.voltor.futureleave.dao.specification;
public class SpecificationUtil {
  public static <T, R> Path<R> buildPath(Root<T> root, String fieldPath) {
    List<String> pathParts = Arrays.asList(fieldPath.split("\\."));
    Path<R> path = root.get(pathParts.get(0));
    for (String pathPart : pathParts.subList(1, pathParts.size())) {
      path = path.get(pathPart);
    }
    return path;
  }
  public static <T, R> Join<R, T> buildJoin(Root<T> root, String fieldPath) {
    List<String> joinParts = Arrays.asList(fieldPath.split("\\."));
    Join<R, T> join = root.joinSet(joinParts.get(0));
    for (String joinPart : joinParts.subList(1, joinParts.size())) {
      join = join.joinSet(joinPart);
    }
    return join;
  }
  public static <T> Specification<T> initEmptySpec() {
    return Specification.where(null);
  }
}
package com.voltor.futureleave.dao.specification;
public class SubclassEqualSpecification<T, S extends T> implements
Specification<T> {
  private static final long serialVersionUID = -1506428092338995164L;
  private final Class< S > subclassClass;
  private final String fieldPath;
  private final Object value;
  public SubclassEqualSpecification( String fieldPath, Object value, Class< S >
subclassClass ) {
    this.fieldPath = fieldPath;
    this.value = value;
    this.subclassClass = subclassClass;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    // A subquery is used because this should work with all InheritanceType
    // options (SINGLE_TABLE, TABLE_PER_CLASS, and JOINED).
    Subquery< S > subQuery = query.subquery( subclassClass );
    Root< S > sqRoot = subQuery.from( subclassClass );
    subQuery.select( sqRoot );
    Path< S > path = SpecificationUtil.buildPath( sqRoot, fieldPath );
    subQuery.where( cb.equal( path, value ) );
    return cb.in( root ).value( subQuery );
  }
}
package com.voltor.futureleave.dao;
@Service
public class UserDao extends AbstractIdentifiableDao< User > {
  private UserRepository repository;
```

```
  public UserDao( AuthenticatedUserService userAuthorizationService,
UserRepository repository ) {
    super( userAuthorizationService );
    this.repository = repository;
  }
  @Override
  protected PrimaryRepository< Long, User > getRepository() {
    return repository;
  }
  @Override
  protected Specification< User > addSpecifications( boolean
includeArchivedEntities ) {
    return SpecificationUtil.initEmptySpec();
  }

}


package com.voltor.futureleave;
@Profile ("h2")
@Component
public class DemoData {
  public DemoData(UserService userService) {
    User user = new User();
    user.setLogin("demo");
    user.setPassword("demo");
    user.setFirstName("demo");
    user.setLastName("demo");
    user.setUserRole(Role.SESSION_USER);
    userService.create(user);
  }

}


package com.voltor.futureleave.filtering;

/**
 * Define a entity property which support filtering
 */
public class FilterableProperty< EntityType > {
  /**
   * entity object property name
   */
  private final String propertyName;
  /**
   * property type
   */
  private final Class< ? > expectedType;
  /**
   * allowed operations for this property
   */
  private final List< FilteringOperation > operators;
  /**
   * helper class used to build {@link
org.springframework.data.jpa.domain.Specification} object for specified property
   */
  private final SpecificationBuilder< EntityType> specificationBuilder;
  public FilterableProperty( String propertyName,
      Class< ? > expectedType, List< FilteringOperation > operators,
      SpecificationBuilder< EntityType > specificationBuilder ) {
```

```
    this.propertyName = propertyName;
    this.expectedType = expectedType;
    this.operators = operators;
    this.specificationBuilder = specificationBuilder;
  }
  public String getPropertyName() {
    return propertyName;
  }
  public Class< ? > getExpectedType() {
    return expectedType;
  }
  public List< FilteringOperation > getOperators() {
    return operators;
  }
  public SpecificationBuilder< EntityType > getSpecificationBuilder() {
    return specificationBuilder;
  }
}
package com.voltor.futureleave.filtering;
public enum FilteringOperation {
  EQUAL("="),
  NOT_EQUAL("!="),
  CONTAIN(":"),
  GREATER_THEN(">"),
  GREATER_OR_EQUAL(">="),
  LESS_THEN("<"),
  LESS_OR_EQUAL("<="),
  IN("_="),
  NOT_IN("!_="),
  IS_NULL("IS NULL"),
  IS_NOT_NULL("IS NOT NULL");
  private final String code;
  FilteringOperation(String code) {
    this.code = code;
  }
  public String getCode() {
    return code;
  }
  public static FilteringOperation fromString(String text) {
    for (FilteringOperation b : FilteringOperation.values()) {
      if (b.code.equalsIgnoreCase(text)) {
        return b;
      }
    }
    throw new IllegalFilteringOperationException("Unknown filtering operation '" +
text + "'");
  }
}
package com.voltor.futureleave.filtering;
public class IllegalFilteringOperationException extends RuntimeException {
  private static final long serialVersionUID = 744571328521036919L;
  private String operation;
  public IllegalFilteringOperationException(String message) {
    super(message);
  }
  public IllegalFilteringOperationException(String operation, String message) {
    super(message);
    this.operation = operation;
  }
  public String getOperation() {
    return operation;
```

```
  }
}package com.voltor.futureleave.filtering.predicate;


public class BigDecimalComparisonSpecification<EntityType> implements
Specification<EntityType> {
  private static final long serialVersionUID = -4321756743799497958L;
  private final SearchCriteria searchCriteria;
    public BigDecimalComparisonSpecification(SearchCriteria searchCriteria) {
        this.searchCriteria = searchCriteria;
    }
    @Override
    public Predicate toPredicate(Root<EntityType> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {

        BigDecimal value = (BigDecimal) searchCriteria.getValue();
    Expression< BigDecimal > searchExpression = root.get( searchCriteria.getKey()
);

    switch ( searchCriteria.getOperation()  ) {
        case GREATER_THEN:
        return cb.greaterThan( searchExpression, value );
        case LESS_THEN:
          return cb.lessThan( searchExpression, value );
        case GREATER_OR_EQUAL:
          return cb.greaterThanOrEqualTo( searchExpression, value);
        case LESS_OR_EQUAL:
          return cb.lessThanOrEqualTo( searchExpression, value );
        case EQUAL:
          return cb.equal( searchExpression, value );
        case NOT_EQUAL:
          return cb.notEqual( searchExpression, value );
        default:
          return null;
    }
    }
}
package com.voltor.futureleave.filtering.predicate;

public class BigDecimalComparisonSpecificationBuilder implements
SpecificationBuilder<Object> {
  public static final List<FilteringOperation> SUPPORTED_OPERATORS =
Arrays.asList(
          FilteringOperation.GREATER_THEN, FilteringOperation.LESS_THEN,
FilteringOperation.GREATER_OR_EQUAL,
          FilteringOperation.LESS_OR_EQUAL, FilteringOperation.EQUAL,
FilteringOperation.NOT_EQUAL
      );
  @Override
  public Specification<Object> buildSpecification(SearchCriteria searchCriteria) {
    return new BigDecimalComparisonSpecification<Object>(searchCriteria);
  }
}
package com.voltor.futureleave.filtering.predicate;

public interface EntityFilterSpecificationsBuilder< EntityType > {
  Logger LOG = LoggerFactory.getLogger(EntityFilterSpecificationsBuilder.class);
  /**
   * Provide provide entity specific filtering configuration. It consist of
entityProperty, it's type, set of allowed operations.
   *
```

```
   * @return non-null list of FilterableProperty
   */
  List< FilterableProperty< EntityType > > getFilterableProperties();
  default Specification< EntityType > buildSpecification( List< SearchCriteria >
searchCriterias ) {
    Specification< EntityType > specification = null;
    for ( SearchCriteria searchCriteria : searchCriterias ) {
      Optional< FilterableProperty< EntityType > > filterableProperty =
          getFilterableProperties().stream()
          .filter( property -> property.getPropertyName().equals(
searchCriteria.getKey() ) ).findFirst();
      if ( filterableProperty.isPresent() ) {

        FilterableProperty< EntityType > filterablePropertyObject =
filterableProperty.get();
        Specification< EntityType > buildSpecification =
filterablePropertyObject.getSpecificationBuilder().buildSpecification(
searchCriteria );

        if ( specification == null ) {
          specification = Specification.where( buildSpecification );
        } else {
          specification = specification.and( buildSpecification );
        }
      } else {
        LOG.warn( "Filtering on property '{}' has been skipped because it's absent
in filterableProperties",
            searchCriteria.getKey() );
      }
    }
    return specification;
  }
  /**
   * Build a WHERE specification of the given searchCriteria
   *
   * @param searchCriteria
   * @return {@link org.springframework.data.jpa.domain.Specification
   */
  default Specification<EntityType> buildSpecification(SearchCriteria
...searchCriteria) {
    return buildSpecification(Arrays.asList(searchCriteria));
  }
}package com.voltor.futureleave.filtering.predicate;

public class EqualingOrNullSpecification< T > implements Specification< T > {
  private static final long serialVersionUID = 2206438670684039538L;
  private final SearchCriteria searchCriteria;
  public EqualingOrNullSpecification( SearchCriteria searchCriteria ) {
    this.searchCriteria = searchCriteria;
  }
  @Override
  public Predicate toPredicate( Root< T > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    final String[] parts = searchCriteria.getKey().split( "\\." );
    Path< ? > path;
    if ( parts.length == 2 ) {
      path = root.get( parts[0] ).get( parts[1] );
    } else {
      path = root.get( searchCriteria.getKey() );
    }
    if ( searchCriteria.getValue() != null ) {
```

```
      if ( FilteringOperation.EQUAL == searchCriteria.getOperation() ) {
        return cb.equal( path, searchCriteria.getValue() );
      } else if ( FilteringOperation.NOT_EQUAL == searchCriteria.getOperation() )
{
        return cb.notEqual( path, searchCriteria.getValue() );
      } else {
        return null;
      }
    } else {
      if ( FilteringOperation.EQUAL == searchCriteria.getOperation() ) {
        return cb.isNull( path );
      } else {
        return cb.isNotNull( path );
      }
    }
  }
  @Override
  public boolean equals( Object o ) {
    if ( this == o )
      return true;
    if ( !( o instanceof EqualingOrNullSpecification ) )
      return false;
    EqualingOrNullSpecification< ? > that = (EqualingOrNullSpecification< ? >) o;
    return new EqualsBuilder().append( searchCriteria, that.searchCriteria
).isEquals();
  }
  @Override
  public int hashCode() {
    return new HashCodeBuilder().append( searchCriteria ).toHashCode();
  }
  @Override
  public String toString() {
    return "EqualingOrNullSpecification{" + "searchCriteria=" + searchCriteria +
'}';
  }
}
package com.voltor.futureleave.filtering.predicate;

public class EqualingOrNullSpecificationBuilder< EntityType > implements
SpecificationBuilder< EntityType > {
  public static final List< FilteringOperation > SUPPORTED_OPERATORS =
Arrays.asList(
      FilteringOperation.EQUAL,
      FilteringOperation.NOT_EQUAL );
  @Override
  public Specification< EntityType > buildSpecification( SearchCriteria
searchCriteria ) {
    return new EqualingOrNullSpecification<>( searchCriteria );
  }
}
package com.voltor.futureleave.filtering.predicate;

public class EqualingSpecification< EntityType > implements Specification<
EntityType > {
  private static final long serialVersionUID = 800528251296820234L;
  private final SearchCriteria searchCriteria;
  public EqualingSpecification( SearchCriteria searchCriteria ) {
    this.searchCriteria = searchCriteria;
  }
  @Override
```

```
  public Predicate toPredicate( Root< EntityType > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    switch ( searchCriteria.getOperation()  ) {
        case EQUAL:
          return cb.equal( root.get( searchCriteria.getKey() ),
searchCriteria.getValue()  );
        case NOT_EQUAL:
          return cb.notEqual( root.get( searchCriteria.getKey() ),
searchCriteria.getValue()  );
        default:
          return null;
      }
  }
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof EqualingSpecification)) return false;
    EqualingSpecification< ? > that = (EqualingSpecification< ? >) o;
    return new EqualsBuilder()
        .append(searchCriteria, that.searchCriteria)
        .isEquals();
  }
  @Override
  public int hashCode() {
    return new HashCodeBuilder()
        .append(searchCriteria)
        .toHashCode();
  }
  @Override
  public String toString() {
    return "EqualingSpecification{" +
        "searchCriteria=" + searchCriteria +
        '}';
  }
}
package com.voltor.futureleave.filtering.predicate;

public class EqualingSpecificationBuilder< EntityType >  implements
SpecificationBuilder< EntityType >  {
  public static final List< FilteringOperation > SUPPORTED_OPERATORS =
Arrays.asList(
      FilteringOperation.EQUAL,
      FilteringOperation.NOT_EQUAL );
  @Override
  public Specification< EntityType >  buildSpecification( SearchCriteria
searchCriteria ) {
    return new EqualingSpecification<>( searchCriteria );
  }
}
package com.voltor.futureleave.filtering.predicate;
public class EqualStringSpecification<T> implements Specification<T> {
  private static final long serialVersionUID = -3614399089131893893L;

  private final String field;
  private final String value;
  private final boolean ignoreCase;
  public EqualStringSpecification(String field, String value, boolean ignoreCase)
{
    this.field = field;
    this.value = value;
    this.ignoreCase = ignoreCase;
```

```
  }
  @Override
  public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {
    final Path<String> path = root.get(field);
    if (value == null) {
      return cb.isNull(path);
    }
    if (ignoreCase) {
      return cb.equal(cb.lower(path), value.toLowerCase());
    }
    return cb.equal(path, value);
  }
}
package com.voltor.futureleave.filtering.predicate;

public class InIdsSpecification< EntityType extends Identifiable > implements
Specification< EntityType > {
  private static final long serialVersionUID = -2185458578685293412L;
  private final SearchCriteria searchCriteria;
  public InIdsSpecification(SearchCriteria searchCriteria) {
    this.searchCriteria = searchCriteria;
  }
  @Override
  public Predicate toPredicate(Root<EntityType> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {
    if (searchCriteria.getValue() == null) {
      return null;
    }
    if (FilteringOperation.IN != searchCriteria.getOperation() &&
FilteringOperation.NOT_IN != searchCriteria.getOperation()) {
      return null;
    }
    Path<?> path;
    if (searchCriteria.getKey().equals("id")) {
      path = root.get("id");
    } else {
      path = SpecificationUtil.buildPath(root, searchCriteria.getKey());
    }
    String[] parts = ((String) searchCriteria.getValue()).split("_");
    // If we have a single value for IN operator, it would perform faster if we
convert it to EQUAL
    boolean include = searchCriteria.getOperation() == FilteringOperation.IN;
    if (parts.length == 1 && !include) {
      return cb.equal(path, Long.valueOf(parts[0])).not();
    } else if (parts.length == 1 && include) {
      return cb.equal(path, Long.valueOf(parts[0]));
    }
    if (!include) {
      return path.in(Arrays.stream(parts).map(Long::valueOf).toArray()).not();
    }
    return path.in(Arrays.stream(parts).map(Long::valueOf).toArray());
  }
}
package com.voltor.futureleave.filtering.predicate;

public class InIdsSpecificationBuilder< EntityType extends Identifiable >
implements SpecificationBuilder< EntityType >  {
  public static final List<FilteringOperation> SUPPORTED_OPERATORS =
Arrays.asList(FilteringOperation.IN, FilteringOperation.NOT_IN);
  @Override
```

```
  public Specification< EntityType > buildSpecification(SearchCriteria
searchCriteria) {
    return new InIdsSpecification<>(searchCriteria);
  }
}
package com.voltor.futureleave.filtering.predicate;

@Component
public class PeriodSpecificationBuilder implements
EntityFilterSpecificationsBuilder<Period> {
  private static final List<FilterableProperty<Period>> FILTERABLE_PROPERTIES =
List
      .of(new FilterableProperty<>("sessionId", Long.class,
EqualingOrNullSpecificationBuilder.SUPPORTED_OPERATORS,
          new EqualingOrNullSpecificationBuilder<>()));
  @Override
  public List<FilterableProperty<Period>> getFilterableProperties() {
    return FILTERABLE_PROPERTIES;
  }
}


package com.voltor.futureleave.filtering.predicate;

@Component
public class SessionSpecificationBuilder implements
EntityFilterSpecificationsBuilder<Session> {
  private static final List<FilterableProperty<Session>> FILTERABLE_PROPERTIES =
List
      .of(new FilterableProperty<>("name", String.class,
EqualingOrNullSpecificationBuilder.SUPPORTED_OPERATORS,
          new EqualingOrNullSpecificationBuilder<>()));
  @Override
  public List<FilterableProperty<Session>> getFilterableProperties() {
    return FILTERABLE_PROPERTIES;
  }
}


package com.voltor.futureleave.filtering.predicate;

public interface SpecificationBuilder< EntityType > {
  Specification< EntityType > buildSpecification( SearchCriteria searchCriteria );
}
package com.voltor.futureleave.filtering.predicate;

public class StringComparisonSpecification< EntityType > implements Specification<
EntityType > {

  private static final long serialVersionUID = 4568825149040657679L;

  private final SearchCriteria searchCriteria;
  public StringComparisonSpecification(SearchCriteria searchCriteria) {
    this.searchCriteria = searchCriteria;
  }
  @Override
  public Predicate toPredicate( Root< EntityType > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {

    String searchValueLowerCase = StringUtils.lowerCase( (String)
searchCriteria.getValue() );
```

```
    Expression< String > searchExpression = cb.lower( root.get(
searchCriteria.getKey() ) );

    switch ( searchCriteria.getOperation()  ) {
        case EQUAL:
          return cb.equal( searchExpression, searchValueLowerCase );
        case NOT_EQUAL:
          return cb.notEqual( searchExpression, searchValueLowerCase );
        case CONTAIN:
          return cb.like( searchExpression, "%" + searchValueLowerCase + "%" );
        default:
          return null;
    }
  }
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof StringComparisonSpecification)) return false;
    StringComparisonSpecification< ? > that = (StringComparisonSpecification< ? >)
o;
    return new EqualsBuilder()
        .append(searchCriteria, that.searchCriteria)
        .isEquals();
  }
  @Override
  public int hashCode() {
    return new HashCodeBuilder()
        .append(searchCriteria)
        .toHashCode();
  }
  @Override
  public String toString() {
    return "StringComparisonSpecification{" +
        "searchCriteria=" + searchCriteria +
        '}';
  }
}
package com.voltor.futureleave.filtering.predicate;

public class StringComparisonSpecificationBuilder< EntityType > implements
SpecificationBuilder< EntityType > {
  public static final List< FilteringOperation > SUPPORTED_OPERATORS =
Arrays.asList(
      FilteringOperation.EQUAL,
      FilteringOperation.NOT_EQUAL,
      FilteringOperation.CONTAIN );
  @Override
  public Specification< EntityType > buildSpecification(SearchCriteria
searchCriteria) {
    return new StringComparisonSpecification<>(searchCriteria);
  }
}
package com.voltor.futureleave.filtering.predicate;


public class ZonedDateTimeComparisonSpecification< EntityType > implements
Specification<EntityType> {
  private static final long serialVersionUID = -8876125803915202920L;

  private final SearchCriteria searchCriteria;
  public ZonedDateTimeComparisonSpecification(SearchCriteria searchCriteria) {
```

```
    this.searchCriteria = searchCriteria;
  }
  @Override
  public Predicate toPredicate(Root<EntityType> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {

    ZonedDateTime value =
ZonedDateTime.parse(searchCriteria.getValue().toString());
    Expression< ZonedDateTime > searchExpression = root.get(
searchCriteria.getKey() );

    switch ( searchCriteria.getOperation()  ) {
        case GREATER_THEN:
        return cb.greaterThan( searchExpression, value );
        case LESS_THEN:
          return cb.lessThan( searchExpression, value );
        case GREATER_OR_EQUAL:
          return cb.greaterThanOrEqualTo( searchExpression, value);
        case LESS_OR_EQUAL:
          return cb.lessThanOrEqualTo( searchExpression, value );
        case EQUAL:
          return cb.equal( searchExpression, value );
        case NOT_EQUAL:
          return cb.notEqual( searchExpression, value );
        default:
          return null;
    }
  }
}
package com.voltor.futureleave.filtering.predicate;

public class ZonedDateTimeComparisonSpecificationBuilder< EntityType > implements
SpecificationBuilder< EntityType > {
  public static final List< FilteringOperation > SUPPORTED_OPERATORS =
Arrays.asList(
      FilteringOperation.GREATER_THEN,
      FilteringOperation.LESS_THEN, FilteringOperation.GREATER_OR_EQUAL,
      FilteringOperation.LESS_OR_EQUAL,
      FilteringOperation.EQUAL,
      FilteringOperation.NOT_EQUAL );
  @Override
  public Specification< EntityType > buildSpecification( SearchCriteria
searchCriteria ) {
    return new ZonedDateTimeComparisonSpecification<>( searchCriteria );
  }
}
package com.voltor.futureleave.filtering.searchcriteria;

public class SearchCriteria implements Serializable {

  private static final long serialVersionUID = -4654470036504265513L;

  private final String key;
  private final FilteringOperation operation;
  private final Object value;
  public SearchCriteria(String key, FilteringOperation operation, Object value) {
    this.key = key;
    this.operation = operation;
    this.value = value;
  }
  public String getKey() {
```

```
    return key;
  }
  public FilteringOperation getOperation() {
    return operation;
  }
  public Object getValue() {
    return value;
  }
  public Class<?> getType() {
    return value == null ? null : value.getClass();
  }
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    SearchCriteria that = (SearchCriteria) o;
    if (getKey() != null ? !getKey().equals(that.getKey()) : that.getKey() !=
null) return false;
    if (getOperation() != null ? !getOperation().equals(that.getOperation()) :
that.getOperation() != null)
      return false;
    return getValue() != null ? getValue().equals(that.getValue()) :
that.getValue() == null;
  }
  @Override
  public int hashCode() {
    int result = getKey() != null ? getKey().hashCode() : 0;
    result = 31 * result + (getOperation() != null ? getOperation().hashCode() :
0);
    result = 31 * result + (getValue() != null ? getValue().hashCode() : 0);
    return result;
  }
  @Override
  public String toString() {
    return "SearchCriteria[" + "key='" + key + '\'' + ", operation='" + operation
+ '\'' + ", value=" + value + ']';
  }
}
package com.voltor.futureleave.filtering.session;
public class ArchivedSpecification< EntityType > implements Specification<
EntityType > {
  private static final long serialVersionUID = -7315580228476708139L;
  @Override
  public Predicate toPredicate( Root< EntityType > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    return cb.equal( root.get( "archived" ), false );
  }
}package com.voltor.futureleave.filtering.session;

public class IdSpecification< IdType, EntityType extends PrimaryEntity< IdType > >
    implements Specification< EntityType > {

  private static final long serialVersionUID = -3617213012291647979L;

  final IdType id;
  boolean inverse = false;
  public IdSpecification( IdType id ) {
    this.id = id;
  }
  public IdSpecification( IdType id, boolean inverse ) {
    this.id = id;
```

```
    this.inverse = inverse;
  }
  @Override
  public Predicate toPredicate( Root< EntityType > root, CriteriaQuery< ? > query,
CriteriaBuilder cb ) {
    if ( !inverse ) {
      return cb.equal( root.get( "id" ), ( id ) );
    }
    return cb.notEqual( root.get( "id" ), ( id ) );
  }
}
package com.voltor.futureleave.filtering.session;

public class SessionSpecification<EntityType> implements Specification<EntityType>
{
  private static final long serialVersionUID = -8146867805381784111L;
  private final Long sessionId;
  public SessionSpecification(Long sessionId) {
    this.sessionId = sessionId;
  }
  @Override
  public Predicate toPredicate(Root<EntityType> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {
    return cb.equal(root.get("sessionId"), this.sessionId);
  }
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof SessionSpecification)) return false;
    SessionSpecification< ? > that = (SessionSpecification< ? >) o;
    return new EqualsBuilder()
        .append(sessionId, that.sessionId)
        .isEquals();
  }
  @Override
  public int hashCode() {
    return new HashCodeBuilder()
        .append(sessionId)
        .toHashCode();
  }
}package com.voltor.futureleave.filtering.user;
public class UserSpecification<EntityType> implements Specification<EntityType> {
  private static final long serialVersionUID = -8146867805381784111L;
  private final Long userId;
  public UserSpecification(Long userId) {
    this.userId = userId;
  }
  @Override
  public Predicate toPredicate(Root<EntityType> root, CriteriaQuery<?> query,
CriteriaBuilder cb) {
    return cb.equal(root.get("user").get("id"), this.userId);
  }
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof UserSpecification)) return false;
    UserSpecification< ? > that = (UserSpecification< ? >) o;
    return new EqualsBuilder()
        .append(userId, that.userId)
        .isEquals();
  }
```

```
  @Override
  public int hashCode() {
    return new HashCodeBuilder()
        .append(userId)
        .toHashCode();
  }
}package com.voltor.futureleave;
@SpringBootApplication
@EntityScan(basePackageClasses = {FutureLeaveApplication.class,
Jsr310JpaConverters.class})
@EnableAutoConfiguration(exclude = RepositoryRestMvcAutoConfiguration.class)
@EnableSpringDataWebSupport
@EnableScheduling
@EnableCaching
public class FutureLeaveApplication extends SpringBootServletInitializer {
  public static void main(final String[] args) {
    SpringApplication.run(FutureLeaveApplication.class, args);
  }
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(FutureLeaveApplication.class);
    }

}
package com.voltor.futureleave.jpa;

@NoRepositoryBean
public interface BaseCRUDRepository<EntityType extends Identifiable> extends
PrimaryRepository<Long, EntityType> {
}
package com.voltor.futureleave.jpa;
public interface PeriodRepository extends BaseCRUDRepository<Period> {
}


package com.voltor.futureleave.jpa;

@NoRepositoryBean
public interface PrimaryRepository<IdType, EntityType extends
PrimaryEntity<IdType>> extends CrudRepository<EntityType, IdType>,
    JpaSpecificationExecutor<EntityType>, PagingAndSortingRepository<EntityType,
IdType> {

}
package com.voltor.futureleave.jpa;
public interface RefreshTokenRepository extends BaseCRUDRepository<RefreshToken> {
}
package com.voltor.futureleave.jpa;

public interface SessionRepository extends BaseCRUDRepository<Session> {
  Optional<Session> findFirstByName(String name);
}
package com.voltor.futureleave.jpa;
public interface UserRepository extends BaseCRUDRepository< User > {
}
package com.voltor.futureleave.localization;
public interface ILocalizedExceptionCode {
  String getCode();
}
package com.voltor.futureleave.localization;
```

```
/**
 * Contains all localization keys used to i18n server-side-generated messages.
 */
public final class LocalizationKeys {
  private LocalizationKeys() {}

    public static final String DRIVER_CREATED = "driver_was_created";
    public static final String DRIVER_GROUP_CREATED = "driver_group_was_created";
    public static final String ACTIVITY_TYPE_CREATED =
"activity_type_was_created";
    public static final String SALARY_VARIABLE_GROUP_CREATED =
"salary_variable_group_was_created";
    public static final String VEHICLE_GROUP_CREATED =
"vehicle_group_was_created";
    public static final String VEHICLE_CREATED = "vehicle_was_created";
    public static final String TRIP_CREATED = "trip_was_created";
    public static final String CURRENCY_CREATED = "currency_was_created";

    public static final String DRIVER_DELETED = "driver_was_deleted";
  public static final String VEHICLE_DELETED = "vehicle_was_deleted";
  public static final String ACTIVITY_TYPE_DELETED = "activity_type_was_deleted";

    public static final String FAILED_PROCESS_ACTIVITY =
"failed_process_activity";
    public static final String FAILED_PROCESS_DRIVER = "failed_process_driver";
    public static final String FAILED_PROCESS_VEHICLE = "failed_process_vehicle";
    public static final String FAILED_PROCESS_ACTIVITY_TYPE =
"failed_process_activity_type";
    public static final String FAILED_PROCESSING_ACTIVITIES =
"failed_processing_activities";

    public static final String IMPORT_STOPPED_FAILURE = "import_stopped_failure";
    public static final String EXPORT_STOPPED_FAILURE = "export_stopped_failure";

    public static final String FAILED_TO_CREATE_ACTIVITIES_FOR_DRIVER =
"failed_to_create_activities_for_driver";
    public static final String FAILED_TO_CREATE_ACTIVITY_FROM_IMPORTED_ACTIVITY =
"failed_to_create_activity_from_imported_activity";

    public static final String STARTING_PROCESSING_IMPORTED_ACTIVITIES_FOR_DRIVER
= "starting_processing_imported_activities_for_driver";
    public static final String FINISHED_PROCESSING_IMPORTED_ACTIVITIES_FOR_DRIVER
= "finished_processing_imported_activities_for_driver";


    public static final String ACTIVITY_INCORRECT_ODOMETER =
"activity_incorrect_odometer";
  public static final String ACTIVITIES_PAIR_INCORRECT_ODOMETER =
"activities_pair_incorrect_odometer";
  public static final String ACTIVITY_INCORRECT_DURATION =
"activity_incorrect_duration";

  public static final String STARTING_TRIP_GENERATION =
"starting_trip_generation";
  public static final String FINISHED_TRIP_GENERATION =
"finished_trip_generation";

  public static final String STARTING_AUTOMATIC_POI_ASSIGN =
"starting_automatic_poi_assign";
  public static final String FINISHED_AUTOMATIC_POI_ASSIGN =
"finished_automatic_poi_assign";
```

```
  public static final String STARTING_MANUAL_POI_ASSIGN =
"starting_manual_poi_assign";
  public static final String FINISHED_MANUAL_POI_ASSIGN =
"finished_manual_poi_assign";

  public static final String FAILED_AUTOMATIC_POI_ASSIGN =
"failed_automatic_poi_assign";
  public static final String FAILED_MANUAL_POI_ASSIGN =
"failed_manual_poi_assign";

  public static final String STARTING_IMPORT = "starting_import";
  public static final String FINISHED_IMPORT = "finished_import";
  public static final String STARTING_EXPORT = "starting_export";
  public static final String FINISHED_EXPORT = "finished_export";

  public static final String STOPPING_IMPORT_ITERATION_LIMIT_EXCEEDED =
"stopping_import_iteration_limit_exceeded";

  public static final String TRIP_GENERATION_ERROR_CODE =
"trip_generation_error_code";

  public static final String POI_ASSIGN_ERROR_CODE = "poi_assign_error_code";

  public static final String ACTIVITY_VALIDATION_ERROR_CODE =
"activity_validation_error_code";

  public static final String IMPORT_ERROR_CODE = "import_error_code";
  public static final String EXPORT_CODE = "export_code";

  public static final String DEPENDENT_OBJECT_CREATION_ERROR_CODE =
"dependent_object_creation_error_code";
  public static final String DEPENDENT_OBJECT_DELETE_ERROR_CODE =
"dependent_object_delete_error_code";

  public static final String NON_TRIP_ACTIVITY_OVERLAP
="non_trip_activity_overlap";

  public static final String ACTIVITY_OVERLAP ="activity_overlap";

  public static final String SBB_PHASE2_EMPTY_RESPONSE =
"sbb_phase2_empty_response";

  /**
   * Indicate error situation - when marker in response is equal to request
   */
  public static final String CC_MARKER_IN_RESPONSE_EQUAL_REQUEST =
"cc_marker_in_response_equal_request";

  public static final String SAVED_IMPORTED_ACTIVITY_FOR_OUT_OF_DATE_DRIVER =
"saved_imported_activity_for_out_of_date_driver";
  public static final String SAVED_IMPORTED_ACTIVITY_FOR_OUT_OF_DATE_VEHICLE =
"saved_imported_activity_for_out_of_date_vehicle";

  public static final String EASY_TRACK_SUCCESSFULLY_DOWNLOADED_FILES_QUANTITY =
"EasyTrackImportHandler.SuccessfullyDownloadedFilesQuantity";
  public static final String EASY_TRACK_NO_FILES_TO_DOWNLOAD =
"EasyTrackImportHandler.NoFilesToDownloaded";
  public static final String EASY_TRACK_FTP_ERROR =
"EasyTrackImportHandler.FTPError";
```

```
  public static final String EASY_TRACK_FTP_CONNECTION_FAILURE =
"EasyTrackImportHandler.FTPConnectionFailure";

  public static final String LOGICWAY_SBB_CALCULATION_DRIVER_IS_ABSENT =
"LogicWaySbb.SalaryCalculation.AbsentDriver";
  public static final String LOGICWAY_SBB_CALCULATION_SALARY_PERIOD_IS_ABSENT =
"LogicWaySbb.SalaryCalculation.AbsentSalaryPeriod";
  public static final String LOGICWAY_SBB_CALCULATION_INTERNAL_ERROR =
"LogicWaySbb.SalaryCalculation.InternalError";
  public static final String LOGICWAY_SBB_CALCULATION_STOPPING_CALCULATION =
"LogicWaySbb.SalaryCalculation.StoppingCalculation";

}
package com.voltor.futureleave.localization;
public class LocalizedMessage {
  private String defaultDescription;
  private String titleKey;
  private String descriptionKey;
  private List<String> detailKeys;
  private Map<String, String> messageArguments;
  private Map<String, String> localizedMessageArguments;
  public LocalizedMessage(String defaultDescription, ILocalizedExceptionCode
titleKey) {
    this.titleKey = Objects.requireNonNull(titleKey.getCode());
    this.defaultDescription = Objects.requireNonNull(defaultDescription);
    this.detailKeys = new ArrayList<>();
  }
  public String getDefaultDescription() {
    return defaultDescription;
  }
  public void setDefaultDescription(String defaultDescription) {
    this.defaultDescription = defaultDescription;
  }
  public String getTitleKey() {
    return titleKey;
  }
  public void setTitleKey(String titleKey) {
    this.titleKey = titleKey;
  }
  public String getDescriptionKey() {
    return descriptionKey;
  }
  public void setDescriptionKey(String descriptionKey) {
    this.descriptionKey = descriptionKey;
  }
  public List<String> getDetailKeys() {
    return detailKeys;
  }
  public void setDetailKeys(List<String> detailKeys) {
    this.detailKeys = detailKeys;
  }
  public Map<String, String> getMessageArguments() {
    return messageArguments;
  }
  public void setMessageArguments(Map<String, String> messageArguments) {
    this.messageArguments = messageArguments;
  }
  public Map<String, String> getLocalizedMessageArguments() {
    return localizedMessageArguments;
  }
```

```
  public void setLocalizedMessageArguments(Map<String, String>
localizedMessageArguments) {
    this.localizedMessageArguments = localizedMessageArguments;
  }
}
package com.voltor.futureleave.model;
public interface Archivable {
  void setArchived(boolean archived);
  boolean isArchived();
}
package com.voltor.futureleave.model;
public class CustomToString extends ToStringStyle {

  private static final long serialVersionUID = 4447419889961907063L;
  public CustomToString() {
    super();
    this.setContentStart("[");
    this.setFieldSeparator(System.lineSeparator() + "  ");
    this.setFieldSeparatorAtStart(true);
    this.setContentEnd(System.lineSeparator() + "]");
  }
  @Override
  protected void appendDetail(StringBuffer buffer, String fieldName, Object value)
{
    if (value instanceof Identifiable) {
      value = ((Identifiable) value).getId();
    }
    buffer.append(value);
  }
}
package com.voltor.futureleave.model;
public interface Identifiable extends PrimaryEntity<Long> {
}
package com.voltor.futureleave.model;
@Entity
@Table(name = "fl_period")
public class Period extends SessionTenencyEntity implements Identifiable{
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column
  private LocalDate startDate;

  @Column
  private LocalDate endDate;
  @Column
  private int plannedEventsCount;

  @Column
  private int unplannedEventsCount;

  public Long getId() {
    return id;
  }
  public void setId(Long id) {
    this.id = id;
  }
  public LocalDate getStartDate() {
    return startDate;
  }
```

```
  public void setStartDate(LocalDate startDate) {
    this.startDate = startDate;
  }
  public LocalDate getEndDate() {
    return endDate;
  }
  public void setEndDate(LocalDate endDate) {
    this.endDate = endDate;
  }

  public int getPlannedEventsCount() {
    return plannedEventsCount;
  }
  public void setPlannedEventsCount(int plannedEventsCount) {
    this.plannedEventsCount = plannedEventsCount;
  }
  public int getUnplannedEventsCount() {
    return unplannedEventsCount;
  }
  public void setUnplannedEventsCount(int unplannedEventsCount) {
    this.unplannedEventsCount = unplannedEventsCount;
  }
  @Override
  public boolean equals( Object o ) {
    if ( this == o )
      return true;
    if ( o == null || getClass() != o.getClass() )
      return false;
    Period other = (Period) o;
    return Objects.equals( id, other.id );
  }
  @Override
  public int hashCode() {
    return Objects.hash( id );
  }
}


package com.voltor.futureleave.model;
public interface PrimaryEntity<T> {

  T getId();
  void setId(T id);
}
package com.voltor.futureleave.model;
@Entity
@Table(name = "refresh_token")
public class RefreshToken extends UserTenencyEntity implements Identifiable{
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column(name = "token", nullable = false, unique = true)
  @NotBlank
  private String token;
  @Column(name = "expiring_date", columnDefinition = "TIMESTAMP WITH TIME ZONE",
nullable = false)
  private ZonedDateTime expiringDate;
  public Long getId() {
    return id;
  }
  public void setId(Long id) {
```

```
    this.id = id;
  }

  public String getToken() {
    return token;
  }
  public void setToken(String token) {
    this.token = token;
  }
  public ZonedDateTime getExpiringDate() {
    return expiringDate;
  }
  public void setExpiringDate(ZonedDateTime expiringDate) {
    this.expiringDate = expiringDate;
  }
  @Override
  public boolean equals( Object o ) {
    if ( this == o )
      return true;
    if ( o == null || getClass() != o.getClass() )
      return false;
    RefreshToken other = (RefreshToken) o;
    return Objects.equals( id, other.id );
  }
  @Override
  public int hashCode() {
    return Objects.hash( id );
  }
}
```

```
package com.voltor.futureleave.model;
public enum Role {

  ROOT,
  SESSION_USER;

  public int getId() {
    return this.ordinal();
  }

}
```

```
package com.voltor.futureleave.model;
@Entity
@Table(name = "fl_session")
public class Session extends SessionTenencyEntity implements Identifiable {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column(nullable = true)
  private String name;

  @Override
  public Long getId() {
    return id;
  }
  @Override
  public void setId(Long id) {
    this.id = id;
```

```
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  @Override
  public boolean equals( Object o ) {
    if ( this == o )
      return true;
    if ( o == null || getClass() != o.getClass() )
      return false;
    Session other = (Session) o;
    return Objects.equals( id, other.id );
  }
  @Override
  public int hashCode() {
    return Objects.hash( id );
  }

}


package com.voltor.futureleave.model;
@MappedSuperclass
public abstract class SessionTenencyEntity extends UserTenencyEntity {

  @Column(name = "session_id")
  protected Long sessionId;
  public Long getSessionId() {
    return sessionId;
  }
  public void setSessionId( Long sessionId ) {
    this.sessionId = sessionId;
  }
}


package com.voltor.futureleave.model;
public enum SupportedLocale {
  EN("en", Locale.ENGLISH),
  NL("nl", new Locale( "nl" ) ),
  DE("de", Locale.GERMAN),
  ES("es", new Locale( "es" ) );

  private final String name;
  private final Locale locale;
  private SupportedLocale(String name, Locale locale) {
    this.name = name;
    this.locale = locale;
  }
  @Override
  public String toString() {
    return this.name;
  }
  public static SupportedLocale getValueByName( String name ) {
    for ( SupportedLocale locale : SupportedLocale.values() ) {
      if ( locale.name.equals( name ) ) {
        return locale;
      }
```

```
    }
    return getDefault();
  }

  private static SupportedLocale getDefault() {
    return NL;
  }

  public static String getDefaultName() {
    return getDefault().name;
  }
  public Locale getLocale() {
    return locale;
  }
}
package com.voltor.futureleave.model;

@Entity
@Table(name="fl_user", uniqueConstraints = {
    @UniqueConstraint( columnNames = "login" )
  }
)
public class User implements Identifiable{
    @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false)
    private Long id;

    @Column
  private String email;

    @Column( name="first_name", nullable = false )
    private String firstName;

    @Column( name="last_name", nullable = false )
    private String lastName;

    @Column(nullable = false)
    private String login;

    @Column(nullable = false)
    private String password;

  //NOTE: we use this complex name because 'role' is reserved for our sh..t
parser!
  @Enumerated(EnumType.STRING)
  @Column(name = "role", nullable = false)
  private Role userRole;
  @Override
  public Long getId() {
    return id;
  }
  @Override
  public void setId( Long id ) {
    this.id = id;
  }
  public String getEmail() {
    return email;
  }
  public void setEmail( String email ) {
    this.email = email;
```

```
  }
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName( String firstName ) {
    this.firstName = firstName;
  }
  public String getLastName() {
    return lastName;
  }
  public void setLastName( String lastName ) {
    this.lastName = lastName;
  }
  public String getLogin() {
    return login;
  }
  public void setLogin( String login ) {
    this.login = login;
  }
  public String getPassword() {
    return password;
  }
  public void setPassword( String password ) {
    this.password = password;
  }
  public Role getUserRole() {
    return userRole;
  }
  public void setUserRole( Role userRole ) {
    this.userRole = userRole;
  }
  @Override
  public int hashCode() {
    return Objects.hash( id );
  }
  @Override
  public boolean equals( Object obj ) {
    if ( this == obj )
      return true;
    if ( !( obj instanceof User ) )
      return false;
    User other = (User) obj;
    return Objects.equals( id, other.id );
  }
}
package com.voltor.futureleave.model;
@MappedSuperclass
public abstract class UserTenencyEntity implements Identifiable {

  @ManyToOne(cascade = CascadeType.ALL, fetch =  FetchType.EAGER)
  protected User user;
  public User getUser() {
    return user;
  }
  public void setUser(User user) {
    this.user = user;
  }
}


package com.voltor.futureleave.security;
```

```
public class AuthenticatedUser extends User {
  private static final long serialVersionUID = 6700782328491796936L;
  private final com.voltor.futureleave.model.User user;
  private final Role role;
  public AuthenticatedUser( com.voltor.futureleave.model.User user, Role role,
boolean enabled, boolean accountNonExpired,
               boolean credentialsNonExpired, boolean accountNonLocked,
               List<GrantedAuthority> authorityList) {
    super( user.getLogin(), user.getPassword(), enabled, accountNonExpired,
credentialsNonExpired,
        accountNonLocked, authorityList);
    this.user = user;
    this.role = role;
  }

  public com.voltor.futureleave.model.User getUser() {
    return user;
  }
  public Role getRole() {
    return role;
  }
}

package com.voltor.futureleave.service;


public abstract class AbstractService<T extends Identifiable> {
  protected abstract AbstractIdentifiableDao<T> getDao();
  public Page<T> get(Pageable pageable) {
    return getDao().get(pageable, false);
  }
  public List<T> get() {
    return getDao().get();
  }
  public List<T> get(Sort sort) {
    return getDao().get(sort, false);
  }
  public Page<T> get(Specification<T> filter, Pageable pageable) {
    return getDao().get(filter, pageable, false);
  }
  public List<T> get(Specification<T> filter) {
    return getDao().get(filter);
  }
  public List<T> get(Specification<T> filter, Sort sort) {
    return getDao().get(filter, sort, false);
  }

  public List<T> getByIds(Collection<Long> ids) {
    return getDao().getByIds(ids);
  }
  public T getOne(Long id) {
    return getDao().getOne(id);
  }

  public T getOne(Specification<T> filter) {
    return getDao().getOne(filter);
  }
  @Transactional
  public T create(T entity) {
    if (entity == null) {
```

```
      throw new IllegalArgumentException(format("Can not create Null %s entity",
getEntityTypeClass().getSimpleName()));
    }
    beforeCreate(entity);
    entity = getDao().create(entity);
    afterCreate(entity);
    return entity;
  }
  /**
   * Method to override when additional create logic is needed
   *
   * @param entity the entity to be created
   */
  protected void beforeCreate(T entity) {
  }
  /**
   * Method to override when additional create logic is needed
   *
   * @param entity the entity to be created
   */
  protected void afterCreate(T entity) {
  }
  @Transactional
  public T update( T entity ) {
    return this.update( entity, false);
  }
  @Transactional
  public T update( T entity, boolean ignorePermissions ) {
    if ( entity == null ) {
      throw new IllegalArgumentException( String.format( "Can not update Null %s
entity", getEntityTypeClass().getSimpleName() ) );
    }
    beforeUpdate( entity );
    entity = getDao().update( entity, ignorePermissions );
    afterUpdate( entity );
    return entity;
  }
  /**
   * Method to override when additional update logic is needed
   *
   * @param entity the entity to be updated
   */
  protected void beforeUpdate(T entity) {
  }
  /**
   * Method to override when additional update logic is needed
   *
   * @param entity the entity to be updated
   */
  protected void afterUpdate(T entity) {
  }
  @Transactional
  public void delete(long id) {
    this.delete(id, false);
  }
  @Transactional
  public void delete(long id, boolean ignorePermissions) {
    T entity = getDao().getOne(id);
    if (entity == null) {
      throw new IllegalArgumentException(format("Can not delete. Record with ID:
%d for class %s not found. ", id, getEntityTypeClass().getSimpleName()));
```

```
    }
    delete(entity, ignorePermissions);
  }
  @Transactional
  protected void delete(Specification<T> specification, boolean ignorePermissions)
{
    T entity = getDao().getOne(specification);
    if (entity == null) {
      throw new IllegalArgumentException(
          format("Can not delete. Record by specification for class %s not found.
", getEntityTypeClass().getSimpleName())
      );
    }
    delete(entity, ignorePermissions);
  }
  private void delete(T entity, boolean ignorePermissions) {
    beforeDelete(entity);
    getDao().delete(entity, ignorePermissions);
    afterDelete(entity);
  }
  /**
   * Method to override when additional deletion logic is needed
   *
   * @param entity the entity to be deleted
   */
  protected void beforeDelete(T entity) {
  }
  /**
   * Method to override when additional delete logic is needed
   *
   * @param entity the entity to be deleted
   */
  protected void afterDelete(T entity) {
  }
  private Class<T> getEntityTypeClass() {
    @SuppressWarnings("unchecked")
    Class<T>[] classes = (Class<T>[])
GenericTypeResolver.resolveTypeArguments(getClass(), AbstractService.class);
    return classes[0];
  }
  public boolean editingIsAllowed(T entity) {
    return getDao().isEditAllowed(entity);
  }
}
package com.voltor.futureleave.service;
public abstract class AbstractUserTenencyService<T extends UserTenencyEntity>
extends AbstractService<T> {
  @Autowired
  private AuthenticatedUserService userAuthorizationService;
  @Autowired
  private UserService userService;

  @Override
  protected void beforeCreate( T entity ) {
    entity.setUser( userService.getOne(
userAuthorizationService.getCurrentUserId() ) );
    super.beforeCreate( entity );
  }

}
```

```
package com.voltor.futureleave.service.ai;

@Component
public class Trainer {
  private static Logger log = LoggerFactory.getLogger(Trainer.class);
  private final static String MODEL_PATH = "model.json";
  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");
  private double MAX_VALUE =  100;
  private int NUM_OUTPUTS = 10;
  private int PART = (int) (MAX_VALUE / NUM_OUTPUTS);

  public void learnBase() throws Exception {
    String csvFilePath = "data-y-1w.csv";
    URL resource = getClass().getClassLoader().getResource(csvFilePath);
    try (CSVReader reader = new CSVReader(new FileReader(resource.getFile()))) {
      // Read all rows at once
      List<String[]> rows = reader.readAll();
      // Convert each row to a Person object
      List<Period> persons = mapRowsToPersons(rows);
      learn(persons);
    }
  }
  private List<Period> mapRowsToPersons(List<String[]> rows) {
    // Assuming the CSV structure is: name,age,email
    // You may need to adapt this part based on your CSV structure
    return rows.stream().skip(1) // Skip the header row if it exists
        .map(row -> {
          Period s = new Period();
          Integer total = Integer.valueOf(row[0]);
          Integer unplanned = Integer.valueOf(row[1]);
          s.setPlannedEventsCount(total - unplanned);
          s.setUnplannedEventsCount(unplanned);
          s.setStartDate(LocalDateTime.parse(row[2], formatter).toLocalDate());
          s.setEndDate(LocalDateTime.parse(row[3], formatter).toLocalDate());
          return s;
        }).toList();
  }

  public void learn(List<Period> dataToLearn) throws Exception {
    List<INDArray> inputList = new ArrayList<>();
    List<INDArray> outputList = new ArrayList<>();
    for (Period period : dataToLearn) {
      inputList.add( periodToLearn(period) );
      outputList.add( Nd4j.create( toBinary( period.getUnplannedEventsCount() ) )
);
    }

    // Задаємо параметри мережі
        int numInput = 3; // Кількість вхідних значень
        int numOutput = NUM_OUTPUTS; // Кількість вихідних значень
        int numHidden = 15; // Кількість прихованих нейронів


        INDArray input = Nd4j.vstack(inputList.toArray(new INDArray[0]));
        INDArray labels = Nd4j.vstack(outputList.toArray(new INDArray[0]));

    System.out.println("Build model....");
    // Fit the model
    MultiLayerNetwork model = null;
```

```
    boolean exists = isNetwork();
    if(exists) {
      model = loadMultiLayerNetwork();
    } else {
      model = createConfig( numInput, numOutput, numHidden);
    }

    model.init();
    model.setListeners(new ScoreIterationListener(100));
    SplitTestAndTrain testAndTrain = new DataSet(input,
labels).splitTestAndTrain(0.99);

    DataSet trainingData = testAndTrain.getTrain();
    DataSet testData = trainingData;

    DataNormalization normalizer = new NormalizerStandardize();
    normalizer.fit(trainingData);
    normalizer.transform(trainingData);
    normalizer.transform(testData);


    for (int i = 0; i < 10000; i++) {
      model.fit(trainingData.getFeatures(), trainingData.getLabels());
    }
    ModelSerializer.writeModel(model, MODEL_PATH, true);
    // Evaluate the model on the test set
    Evaluation eval = new Evaluation();
    INDArray output = model.output(testData.getFeatures());
    eval.eval(testData.getLabels(), output);
    log.info(eval.stats());
  }
  public boolean isNetwork() {
    return new File(MODEL_PATH).exists();
  }

  public MultiLayerNetwork createConfig(int numInputs, int numOutputs, int
numHiddenUnits) {
      // Створюємо конфігурацію мережі
      NeuralNetConfiguration.ListBuilder builder = new
NeuralNetConfiguration.Builder()
               .seed(123)
                 .updater(new org.nd4j.linalg.learning.config.Adam(0.00001))
                 .activation(Activation.TANH)
                 .weightInit(WeightInit.XAVIER)
                 .l2(1e-4)
                 .list()
                 .layer(new DenseLayer.Builder().nIn(numInputs).nOut(numInputs*2)
                     .build())
                 .layer(new
DenseLayer.Builder().nIn(numInputs*2).nOut(numOutputs*2)
                     .activation(Activation.CUBE)
                     .build())
                 .layer(new
DenseLayer.Builder().nIn(numOutputs*2).nOut(numOutputs*2)
                         .build())
                 .layer(new
DenseLayer.Builder().nIn(numOutputs*2).nOut(numOutputs*2)
                         .build())
                 .layer(new
DenseLayer.Builder().nIn(numOutputs*2).nOut(numOutputs*2)
```

```
                        .build())
                .layer(new
DenseLayer.Builder().nIn(numOutputs*2).nOut(numOutputs*2)
                        .build())
                .layer( new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                    .activation(Activation.SOFTMAX) //Override the global TANH
activation with softmax for this layer
                    .nIn(numOutputs*2).nOut(numOutputs).build());
      return new MultiLayerNetwork(builder.build());
    }

  private MultiLayerNetwork loadMultiLayerNetwork() {
    try {
      return ModelSerializer.restoreMultiLayerNetwork(MODEL_PATH, true);
    }  catch (IOException e) {
      log.error(" getForecast error", e);
    }
    return null;
  }
  public int getPredict( Period period ) {
    MultiLayerNetwork layerNetwork = loadMultiLayerNetwork();
    if(layerNetwork==null) {
      return -1;
    }

    INDArray input = periodToLearn(period);


    INDArray inputMatrix = input.reshape(1, input.length());

    int[] result = layerNetwork.predict( inputMatrix );
    return result[0]*PART;
  }

  private static double getValueFromDate( LocalDate date ) {

    return  (double) buildBase(date);
  }

  private static double buildBase( LocalDate date ) {

    return (double) date.getDayOfYear() ;
  }

  public INDArray periodToLearn( Period period ) {
    return Nd4j.create(  new double[] { period.getPlannedEventsCount() ,
          getValueFromDate( period.getStartDate() ),
            getValueFromDate( period.getEndDate() )  } );
  }

  public void removeNetwork() {
    File file = new File(MODEL_PATH);
        if( file.delete() ) {
            System.out.println("Network deleted successfully");
        }
  }

  public double[] toBinary( int num ) {
    double[] result = new double[ NUM_OUTPUTS ];// { num/MAX_VALUE};
    int index = num/PART;
```

```
    index = index >= NUM_OUTPUTS ? (int) NUM_OUTPUTS - 1 : index;
    result [index] = 1;

    return result;

  }
}


package com.voltor.futureleave.service;
@Service
public class AuthenticatedUserService {

  public boolean isRoot() {
    return Role.ROOT.equals( getCurrentAuthUser().getRole() );
  }
  public boolean isSupport() {
    return Role.SESSION_USER.equals( getCurrentAuthUser().getRole() );
  }

  public User getCurrentUser() {
    return getCurrentAuthUser().getUser();
  }

  public Long getCurrentUserId() {
    return getCurrentUser().getId();
  }
  private AuthenticatedUser getCurrentAuthUser() throws NoCurrentUserException {
    final Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication == null) {
      throw new NoCurrentUserException();
    }
    if (!(authentication.getPrincipal() instanceof AuthenticatedUser)) {
      throw new NoCurrentUserException();
    }
    final AuthenticatedUser authenticatedUser = (AuthenticatedUser)
authentication.getPrincipal();
    if (authenticatedUser == null ) {
      throw new NoCurrentUserException();
    }
    return authenticatedUser;
  }

  public void authenticateUser(UserDetails userDetails) {
    UsernamePasswordAuthenticationToken authentication =
        new UsernamePasswordAuthenticationToken(userDetails,
userDetails.getUsername());
    authenticateUser(authentication);
  }
  public void authenticateUser(Authentication authentication) {
    SecurityContextHolder.getContext().setAuthentication(authentication);
  }
}

  public static UserAlreadyExistsException buildUserAlreadyExistsException(String
emailAddress) {
    final String defaultErrorMessage = String.format("User with email address (%s)
already exists.", emailAddress);
    final Map<String, String> messageArguments = new HashMap<>();
    messageArguments.put("email", emailAddress);
```

```
    final LocalizedMessage message = new LocalizedMessage(defaultErrorMessage,
LocalizedExceptionCode.USER_EXISTS_EXCEPTION);
    message.setMessageArguments(messageArguments);
    return new UserAlreadyExistsException(message);
  }
  public static RetrievalNotAllowedException buildRetrievalNotAllowedException(
      Class<?> entityClass, LocalizedMessage... details) {
    final String defaultErrorMessage = String.format("Current user is not allowed
to retrieve %s.",
        entityClass.getSimpleName());
    final Map<String, String> localizedArguments = new HashMap<>();
    localizedArguments.put("entity", entityClass.getSimpleName().toUpperCase());
    final LocalizedMessage message = new LocalizedMessage(defaultErrorMessage,
        LocalizedExceptionCode.RETRIEVAL_NOT_ALLOWED_EXCEPTION_MESSAGE);
    message.setLocalizedMessageArguments(localizedArguments);
    return new RetrievalNotAllowedException(message, details == null ? null :
Arrays.asList(details));
  }
  public static ConstraintValidationException
buildConstraintValidationExceptionMissingRequiredField(
      Class<?> entityClass, String fieldName, LocalizedMessage... details) {
    final String defaultErrorMessage = String.format("[%s] field is mandatory for
%s.", fieldName,
        entityClass.getSimpleName());
    final Map<String, String> messageArguments = new HashMap<>();
    messageArguments.put("fieldName", fieldName);
    final Map<String, String> localizedArguments = new HashMap<>();
    localizedArguments.put("entity", entityClass.getSimpleName().toUpperCase());
    final LocalizedMessage message = new LocalizedMessage(defaultErrorMessage,

LocalizedExceptionCode.CONSTRAINT_VALIDATION_EXCEPTION_MISSING_REQUIRED_FIELD);
    message.setLocalizedMessageArguments(localizedArguments);
    message.setMessageArguments(messageArguments);
    return new ConstraintValidationException(message);
  }
  public static String getEntityNameOrId(Identifiable entity) {
    Method m = ReflectionUtils.findMethod(entity.getClass(), "getName");
    if (m != null) {
      return (String) ReflectionUtils.invokeMethod(m, entity);
    } else {
      return entity.getId() == null ? "-" : entity.getId().toString();
    }
  }
}
package com.voltor.futureleave.service;
@Service
public class PeriodService extends AbstractUserTenencyService<Period> {
  private final PeriodDao periodDao;
  @Autowired
  public PeriodService(PeriodDao periodDao) {
    this.periodDao = periodDao;
  }
  @Override
  protected AbstractIdentifiableDao<Period> getDao() {
    return periodDao;
  }
}


package com.voltor.futureleave.service;
```

```
@Service
public class RefreshTokenService extends AbstractUserTenencyService<RefreshToken>
{
  @Autowired
  private RefreshTokenDao refreshTokenDao;


  @Override
  protected AbstractIdentifiableDao<RefreshToken> getDao() {
    return refreshTokenDao;
  }
  public RefreshToken create( String token, ZonedDateTime expiringDate ) {
    RefreshToken refreshToken = new RefreshToken();
    refreshToken.setToken( token );
    refreshToken.setExpiringDate( expiringDate );
    return super.create( refreshToken );
  }

  public User getTokenData(String refreshToken) {
    RefreshToken token = getOne(new EqualSpecification<>("token", refreshToken));
    if (token == null) {
      throw new RefreshTokenNotFoundException();
    }
    delete( token.getId() );
    return token.getUser();
  }
  public void delete(String refreshToken) {
    delete(new EqualSpecification<>("token", refreshToken), true);
  }
}


package com.voltor.futureleave.service;

@Service
public class SessionService extends AbstractService<Session> {
  private final SessionDao sessionDao;
  @Autowired
  public SessionService(SessionDao sessionDao) {
    this.sessionDao = sessionDao;
  }
  @Override
  protected AbstractIdentifiableDao<Session> getDao() {
    return sessionDao;
  }

  public Session getCurrentSession() {
    return sessionDao.get().iterator().next();
  }
  @Override
  public void delete(long id, boolean ignorePermissions) {
    String msg = "Removing session is not allowed";
    throw new ActionNotAllowedException(new LocalizedMessage(msg,
LocalizedExceptionCode.REMOVAL_NOT_ALLOWED_EXCEPTION_MESSAGE));
  }
}
package com.voltor.futureleave.service;
@Service
public class UserService extends AbstractService< User > {

  @Autowired private UserDao dao;
```

```
  @Autowired private PasswordEncoder encoder;
  @Override
  protected AbstractIdentifiableDao< User > getDao() {
    return dao;
  }

  public User createOrUpdate( User user ) {
    if( user.getId() == null ) {
      return create( user );
    }
    return update( user );
  }

  @Override
  protected void beforeCreate(User entity) {
    entity.setPassword( encoder.encode( entity.getPassword() ) );
  }

  @Override
  protected void beforeUpdate(User entity) {
    User oldEntity = dao.getOneArchived( entity.getId(), true );
    if( !oldEntity.getPassword().equals( entity.getPassword() ) ) {
      entity.setPassword( encoder.encode( entity.getPassword() ) );
    }
  }
  public boolean isLoginUnique( User user ) {
    Specification< User > idFilter = new NotEqualSpecification<>( "id",
user.getId() );
    Specification< User > loginFilter = new EqualSpecification<>( "login",
user.getLogin() );
    return !getDao().existsIncludingArchived( loginFilter.and( idFilter ), true );
  }

  public User findByLogin( String login ) {
    Specification< User > specification = new EqualSpecification<>( "login", login
);
    return dao.getOne( specification );
  }
}


package com.voltor.futureleave.api.v1.controller;

public abstract class AbstractCRUDControllerTest<
    Entity extends Identifiable,
    Request extends AbstractRequest< Entity >,
    Response extends AbstractResponse>
    extends SpringBasedControllerTest {
  protected static final Logger LOGGER = LoggerFactory.getLogger(
AbstractCRUDControllerTest.class );
  protected ResultCaptor< Entity > resultCaptor;

  @BeforeEach
  public void setup(){
    resultCaptor = new ResultCaptor<>();
  }

  @Test
  public void testItemCreation() throws Exception {
```

```
    mockServiceCreateOrUpdateMethod( resultCaptor, whenCreateInService( any(
getEntityClass() ) ) );

    ResultActions resultAction = mvc.perform( post( getControllerPath() )
        .content( getJson( getNewRequestBean() ) )
        .contentType( MediaType.APPLICATION_JSON )
        .accept( MediaType.APPLICATION_JSON ) )
      .andExpect( status().isCreated() );

    assertNotNull( resultCaptor.getResult() );

    Response expectedResponse = convertToResponse( resultCaptor.getResult() );
    resultAction.andExpect( content().json( getJson( expectedResponse ) ) );
  }

  protected <T> void mockServiceCreateOrUpdateMethod( ResultCaptor< Entity >
resultCaptor, OngoingStubbing<T> ongoingStubbing){
    ongoingStubbing.thenAnswer( invocation -> {
      Entity entity = invocation.getArgument( 0, getEntityClass() );
      entity.setId( getRandomId() );
      resultCaptor.setResult(entity);
      return entity;});
  }

  @Test
  public void testItemCreationOptionalFields() throws Exception {
    String path = getControllerPath();
    mockServiceCreateOrUpdateMethod( resultCaptor, whenCreateInService( any(
getEntityClass() ) ) );
    getCreateOptionalFieldsTestParameters().forEach( ( requestedMap, valueProvider
) -> doOptionalFieldsCreationTest( path, requestedMap, valueProvider ) );
  }

  private void doOptionalFieldsCreationTest( String path,
      Consumer< Request > setter, Pair< Function< Entity, Object >,  Function<
Entity, Object > > valueProvider ) {
    Entity entity = getNewEntity();
    Request request = convertToRequest( entity );
    setter.accept( request );

    try {
      mvc.perform( post( path, entity.getId() )
          .contentType( MediaType.APPLICATION_JSON )
          .content( getJson( request ) ) )
        .andExpect( status().is2xxSuccessful() );
    } catch( Exception e ) {
      LOGGER.error( "Failed OptionalFieldsCreation test because of an exception",
e );
      fail();
    }

    Entity result = resultCaptor.getResult();
    assertEquals( valueProvider.getRight().apply( result ),
valueProvider.getLeft().apply( result ) );
  }

  @Test
  public void testItemUpdating() throws Exception {
    Entity entity = getNewEntity();

    Entity entityWithNewValues = getNewEntity();
```

```
    entityWithNewValues.setId( entity.getId() );

    Request request = convertToRequest( entityWithNewValues );

    List< Function< Entity, Object >  > valuesGetters = getValueToBeUpdated(
request );
    mockServiceGetEntity( entity );

    mockServiceCreateOrUpdateMethod( resultCaptor, whenUpdateInService( any(
getEntityClass() ) ) );

    ResultActions resultAction = mvc.perform( put( getControllerPath() + "/{id}",
entity.getId() )
        .contentType( MediaType.APPLICATION_JSON )
        .content( getJson( request ) ) )
      .andExpect( status().isOk() );

    Entity updatedEntity = resultCaptor.getResult();

    Response expectedResponse = convertToResponse( updatedEntity );
    resultAction.andExpect( content().json( getJson( expectedResponse ) ) );

    for ( Function< Entity, Object > getter : valuesGetters ) {
      assertEquals( getter.apply( entityWithNewValues ), getter.apply(
updatedEntity ) );
    }

  }

  @Test
  public void testPatching() throws Exception {
    String path = getControllerPath() + "/{id}";
    getPatchValuesTestParameters().forEach( ( requestedMap, valueProvider ) ->
doPatchTest( path, requestedMap, valueProvider ) );
  }

  private void doPatchTest( String path, Map< String, Object > requestedMap, Pair<
Function< Entity, Object >, Object > valueProvider ) {

    Entity entity = getNewEntity();

    mockServiceGetEntity( entity );
    mockServiceCreateOrUpdateMethod( resultCaptor, whenUpdateInService( any(
getEntityClass() ) ) );

    try {
      mvc.perform( patch( path, entity.getId() )
          .contentType( MediaType.APPLICATION_JSON )
          .content( getJson( requestedMap ) ) )
        .andExpect( status().isOk() );
    } catch( Exception e ) {
      LOGGER.error( "Failed patch test because of an exception", e );
      fail();
    }
    Entity updatedEntity = resultCaptor.getResult();
    assertEquals( valueProvider.getRight(), valueProvider.getLeft().apply(
updatedEntity ) );
  }

  @Test
  public void testPatchingExceptions() throws Exception {
```

```
    String path = getControllerPath() + "/{id}";
    getPatchExceptionsValuesTestParameters().forEach( ( requestedMap,
reasonMessage ) -> doPatchExceptionTest( path, requestedMap, reasonMessage ) );
  }

  private void doPatchExceptionTest( String path, Map< String, Supplier< Object >
> requestedMap, String reasonMessage ) {
    Entity entity = getNewEntity();
    mockServiceGetEntity( entity );

    Map< String, Object > queryMap = new HashMap<>();
    for( Entry< String, Supplier< Object > > entry : requestedMap.entrySet() ) {
      queryMap.put( entry.getKey(), entry.getValue().get() );
    }

    try {
      mvc.perform( patch( path, entity.getId() )
          .contentType( MediaType.APPLICATION_JSON )
          .content( getJson( queryMap ) ) )
        .andExpect( status().is4xxClientError() )
        .andExpect( jsonPath( "$.title.defaultDescription" ).value( reasonMessage
) );
    } catch( Exception e ) {
      LOGGER.error( "Failed patch test because of an exception", e );
      fail();
    }
  }

  @Test
  public void testSuccessOnFindById() throws Exception {
    Entity entity = getNewEntity();
    mockServiceGetEntity( entity );

    Response expectedResponse = convertToResponse( entity );
    mvc.perform( get( getControllerPath() + "/{id}", entity.getId() )
        .accept( MediaType.APPLICATION_JSON ) )
      .andDo( MockMvcResultHandlers.print() )
      .andExpect( status().isOk() )
      .andExpect( content().json( getJson( expectedResponse ) ) );
  }
  @Test
  public void testNotFoundOnFindById() throws Exception {
    Long missingId = 10000L;
    when( getService().getOne( anyLong() ) ).thenReturn( null );
    ObjectNotFoundException notFoundException = LocalizedExceptionUtil
        .buildObjectNotFoundException( getEntityClass(), missingId);
    mvc.perform( get( getControllerPath() + "/{id}", missingId )
        .accept( MediaType.APPLICATION_JSON ) )
      .andExpect( status().isNotFound() )
      .andExpect( jsonPath( "$.status" ).value("NOT_FOUND"))
      .andExpect( jsonPath( "$.title.defaultDescription" )
          .value( notFoundException.getLocalizedMessage() )
      );
  }

  @Test
  public void testCreatingWithWrongParameter() {
    whenCreateInService( any( getEntityClass() ) ).thenAnswer( invocation -> {
      Entity entity = spy( invocation.getArgument( 0, getEntityClass() ) );
      when( entity.getId() ).thenReturn( getRandomId() );
      return entity;
```

```
    });

    getCreateWithWrongValuesTestParameters().forEach( this ::
doCreatingWithWrongValuesTest );
  }

  protected void doCreatingWithWrongValuesTest( Consumer< Request >
missingValueSetter, String exceptionMessage ) {
    Request request = getNewRequestBean();
    missingValueSetter.accept( request );

    try {
      mvc.perform( post( getControllerPath() )
          .contentType( MediaType.APPLICATION_JSON )
          .content( getJson( request ) ) )
        .andExpect( status().is4xxClientError() )
        .andExpect( jsonPath( "$.title.defaultDescription" ).value(
exceptionMessage ) );
    } catch( Exception e  ) {
      LOGGER.error( "Failed creation test because of an exception", e );
      fail();
    }
  }

  protected void testServiceExceptionDuringCreation( LocalizedException
describedServiceException ) {
    whenCreateInService( any( getEntityClass() ) ).thenThrow(
describedServiceException );
    doCreatingWithWrongValuesTest( request -> {},
describedServiceException.getMessage() );
  }

  @Test
  public void testUpdatingWithMissingParameter() {
    getUpdateWithWrongValuesTestParameters().forEach( this ::
doUpdatingWithMissingValuesTest );
  }

  protected void doUpdatingWithMissingValuesTest( Consumer< Request >
missingValueSetter, String exceptionMessage ) {
    Entity entity = getNewEntity();
    Request request = convertToRequest( entity );
    missingValueSetter.accept( request );

    mockServiceGetEntity( entity );

    try {
      mvc.perform( put( getControllerPath() + "/{id}", entity.getId() )
          .contentType( MediaType.APPLICATION_JSON )
          .content( getJson( request ) ) )
        .andExpect( status().is4xxClientError() )
        .andExpect( jsonPath( "$.title.defaultDescription" ).value(
exceptionMessage ) );
    } catch( Exception e  ) {
      LOGGER.error( "Failed updating test because of an exception", e );
      fail();
    }
  }

  @Test
  public void testFindAll() throws Exception {
```

```
    List< Entity > entities = new LinkedList<>( Arrays.asList( getNewEntity(),
getNewEntity() ) );
    Page< Entity > page = mockPage( entities );

    when( getService().get( any(), any(Pageable.class) ) ).thenReturn(page);

    List< Response > expectedResponse = entities.stream().map( this ::
convertToResponse ).collect( Collectors.toList() );

    PageResponse< Response > expectedResponses = new
PageResponse<>(expectedResponse, page.getTotalElements(), page.getNumber(),
page.getSize());
    mvc.perform( get( getControllerPath() )
        .accept( MediaType.APPLICATION_JSON_VALUE ) )
      .andExpect( status().isOk() )
      .andExpect( content().json( getJson( expectedResponses ) ) );
  }

  @Test
  public void testDefaultPageValuesForFindAll() throws Exception {
    Page< Entity > page = getMockedServicePage();
    when( getService().get( any(), any(Pageable.class) ) ).thenReturn( page );
    mvc.perform(get( getControllerPath())
        .accept(MediaType.APPLICATION_JSON_VALUE)
        .param("index", "1"))
        .andExpect(status().isOk());
    assertEquals( 25, getLastPageRequest().getPageSize());
  }
  @Test
  public void testCustomPageValuesForFindAll() throws Exception {
    Page< Entity > page = getMockedServicePage();
    when( getService().get( any(), any(Pageable.class) ) ).thenReturn( page );
    mvc.perform(get( getControllerPath())
        .accept(MediaType.APPLICATION_JSON_VALUE)
        .param("index", "8")
        .param("size", "52"))
        .andExpect(status().isOk());
    Pageable lastPageRequest = getLastPageRequest();
    assertEquals( 52, lastPageRequest.getPageSize());
    assertEquals( 8, lastPageRequest.getPageNumber());
  }

  @Test
  public void testPageSizeValidationForFindAll() throws Exception {
    mvc.perform( get( getControllerPath() )
        .param( "size", "101" )
        .accept( MediaType.APPLICATION_JSON_VALUE ) )
      .andExpect( status().is4xxClientError() )
      .andExpect( jsonPath( "$.title.defaultDescription" ).value( "Validation
failed, Page size must be between 1 and 100 inclusive" ) );

    mvc.perform( get( getControllerPath() )
        .param( "size", "-1" )
        .accept( MediaType.APPLICATION_JSON_VALUE ) )
      .andExpect( status().is4xxClientError() )
      .andExpect( jsonPath( "$.title.defaultDescription" ).value( "Validation
failed, Page size must be between 1 and 100 inclusive" ) );
  }

  @Test
  public void testPageIndexValidationForFindAll() throws Exception {
```

```
    mvc.perform( get( getControllerPath() )
        .param( "index", "-1" )
        .accept( MediaType.APPLICATION_JSON_VALUE ) )
      .andExpect( status().is4xxClientError() )
      .andExpect( jsonPath( "$.title.defaultDescription" ).value( "Validation
failed, Page index must have a positive or zero value" ) );
  }
  @Test
  public void findAllSortingTest() {
    Page< Entity > page = getMockedServicePage();
    when( getService().get( any(), any(Pageable.class) ) ).thenReturn(page);
    Map< List< String >, Sort > sortingTestParameters =
getSortingTestParameters();
    sortingTestParameters.forEach( this :: doSortingTest );
  }
  protected void doSortingTest( List< String > sortingValues, Sort expectedResult
) {
    MockHttpServletRequestBuilder mockHttpServletRequestBuilder = get(
getControllerPath() );
    sortingValues.forEach( sortingValue -> mockHttpServletRequestBuilder.param(
"sort", sortingValue  ) );
    try {
      mvc.perform( mockHttpServletRequestBuilder )
          .andExpect( status().isOk() );
    } catch( Exception e ) {
      LOGGER.error( "Failed sorting test because of an exception", e );
      fail();
    }
    assertEquals( expectedResult, getLastPageRequest().getSort() );
  }

  @Test
  public void testFilteringForFindAll() {
    Page< Entity > page = getMockedServicePage();
    when( getService().get( any(), any(Pageable.class) ) ).thenReturn( page );
    Map< List< String >, List< SearchCriteria > > filteringTestParameters =
getFilteringTestParameters();
    filteringTestParameters.forEach( this :: doFilterTest );
  }

  @SuppressWarnings("unchecked")
  protected void doFilterTest( List< String > filteringValues,  List<
SearchCriteria >  expectedSearchCriterias ) {
    MockHttpServletRequestBuilder mockHttpServletRequestBuilder = get(
getControllerPath() );
    filteringValues.forEach( filteringValue ->
mockHttpServletRequestBuilder.param( "search", filteringValue ) );
    ArrayList< SearchCriteria > listClassObjsct = new ArrayList<>();
    ArgumentCaptor< List< SearchCriteria > > createriasCaptor =
ArgumentCaptor.forClass( listClassObjsct.getClass() );
    try {
      mvc.perform( mockHttpServletRequestBuilder )
          .andExpect( status().isOk() );
    } catch( Exception e ) {
      LOGGER.error( "Failed filtering test because of an exception", e );
      fail();
    }
    verify( getSpecificationsBuilder(), atLeastOnce() ).buildSpecification(
createriasCaptor.capture() );
    Specification< Entity > specification = getLastCalledSpecification();
    assertNotNull( specification, expectedSearchCriterias.toString() );
```

```
    assertEquals( expectedSearchCriterias,  createriasCaptor.getValue() );
  }

  @Test
  public void findAllFilteringWrongValueTest() throws Exception {
    Page< Entity > page = getMockedServicePage();
    when( getService().get( any(), any(Pageable.class) ) ).thenReturn( page );
    mvc.perform( get( getControllerPath() )
        .param( "search", "bla bla" ) )
            .andExpect( status().isOk() );
    assertNull( getLastCalledSpecification() );
  }
  @Test
  public void testDelete() throws Exception {
    Entity entity = getNewEntity();

    mockServiceGetEntity( entity );

    mvc.perform( delete( getControllerPath() + "/{id}", entity.getId() ) )
      .andExpect( status().isNoContent() );
    verify( getService(), times( 1 ) ).delete( eq( entity.getId() ) );
  }

  @Test
  public void testCanNotDelete() throws Exception {
    Entity entity = getNewEntity();

    mockServiceGetEntity( entity );

    Mockito.doThrow(RemovalNotAllowedException.class).when(getService()).delete(
entity.getId() );
    mvc.perform( delete( getControllerPath() + "/{id}", entity.getId() ) ).andDo(
MockMvcResultHandlers.print() )
        .andExpect( status().isNotAcceptable() );
  }
  @Test
  public void testCanNotUpdate() throws Exception {
    Entity entity = getNewEntity();
    Request request = convertToRequest( entity );
    mockServiceGetEntity( entity );
    whenUpdateInService( any( getEntityClass() ) ).thenThrow(
UpdateNotAllowedException.class );
    mvc.perform( put( getControllerPath() + "/{id}", entity.getId() )
            .contentType( MediaType.APPLICATION_JSON )
            .content( getJson( request ) ) )
        .andExpect( status().isMethodNotAllowed() );
  }

  protected OngoingStubbing< Entity > whenCreateInService( Entity entity ) {
    return when( getService().create( entity ) );
  }

  protected OngoingStubbing< Entity > whenUpdateInService( Entity entity ) {
    return when( getService().update( entity ) );
  }

  protected void mockServiceGetEntity( Entity entity ) {
    when( getService().getOne( entity.getId() ) ).thenReturn( entity );
  }

  protected abstract Class< Entity > getEntityClass();
```

```
  protected abstract AbstractService< Entity > getService();

  protected abstract Entity getNewEntity();

  protected abstract Request getNewRequestBean();

  protected abstract Map< List< String >, Sort> getSortingTestParameters();

  /* TODO: Is the parameter needed here? */
  protected abstract List< Function< Entity, Object > > getValueToBeUpdated(
Request request );
  protected abstract Map< List< String >, List< SearchCriteria > >
getFilteringTestParameters();

  protected abstract EntityFilterSpecificationsBuilder<Entity>
getSpecificationsBuilder();

  protected abstract HashMap< Consumer< Request >, Pair< Function< Entity, Object
>, Function< Entity, Object > > > getCreateOptionalFieldsTestParameters();

  protected abstract Map< Map< String, Supplier< Object > >, String >
getPatchExceptionsValuesTestParameters();

  protected abstract Map< Map< String, Object >, Pair< Function< Entity, Object >,
Object > > getPatchValuesTestParameters();

  protected abstract Request convertToRequest( Entity entity );

  protected abstract Response convertToResponse( Entity entity );

  protected abstract Map< Consumer< Request >, String >
getCreateWithWrongValuesTestParameters();

  protected Map< Consumer< Request >, String >
getUpdateWithWrongValuesTestParameters(){
    return getCreateWithWrongValuesTestParameters();
  }

  protected abstract String getControllerPath();

  protected String getControllerImportPath() {
    return getControllerPath() + "/import";
  }

  protected Pageable getLastPageRequest() {
    ArgumentCaptor< Pageable > captor = ArgumentCaptor.forClass(Pageable.class);
    verify( getService(), atLeastOnce() ).get( any(), captor.capture() ) ;
    return captor.getValue();
  }

  @SuppressWarnings("unchecked")
  protected Specification< Entity > getLastCalledSpecification() {
    ArgumentCaptor< Specification< Entity > > captor = ArgumentCaptor.forClass(
Specification.class );
    verify( getService(), atLeastOnce() ).get( captor.capture(), any(
Pageable.class ) );
    return captor.getValue();
  }

  protected Entity getLastCreatedEntity() {
```

```
    ArgumentCaptor< Entity > captor = ArgumentCaptor.forClass( getEntityClass() );
    whenCreateInService( captor.capture() );
    return captor.getValue();
  }

  protected Entity getLastUpdatedEntity() {
    ArgumentCaptor< Entity > captor = ArgumentCaptor.forClass( getEntityClass() );
    whenUpdateInService( captor.capture() );
    return captor.getValue();
  }

  private Page< Entity > getMockedServicePage() {
    return mockPage( new ArrayList<>() );
  }

  @SuppressWarnings("unchecked")
  private Page< Entity > mockPage( List< Entity > entities ) {
    Page< Entity > page = mock(Page.class);
    when( page.getContent()).thenReturn(entities);
    when( page.getTotalElements() ).thenReturn( 765L );
    when( page.getNumber() ).thenReturn( 1 );
    when( page.getSize() ).thenReturn( 5 );
    when( page.iterator() ).thenReturn( entities.iterator() );
    when( getService().get( any(), any(Pageable.class) ) ).thenReturn(page);
    return page;
  }

  protected Long getRandomId() {
    return ThreadLocalRandom.current().nextLong( Long.MAX_VALUE - 1 );
  }
}


package com.voltor.futureleave.api.v1.controller.authentication;

@WebMvcTest( AuthenticationController.class )
@ActiveProfiles(profiles = { "develop" })
@Import( SpringSecurityConfig.class  )
public class AuthenticationControllerTest {
  private static final String API_URL = AuthenticationController.API_URL;
  private HttpMessageConverter< Object > mappingJackson2HttpMessageConverter;
  @MockBean
  private JwtService jwtService;
  @MockBean
  private AuthenticationManager authenticationManager;
  @MockBean
  private UserDetailsServiceImpl userDetailsService;
  @MockBean
  private RefreshTokenService refreshTokenService;
  @MockBean
  private AuthenticatedUserService userAuthorizationService;
  @Autowired
  private AuthenticationController authenticationController;
  protected MockMvc mvc;
  @Autowired
  protected void setConverters( HttpMessageConverter< Object >[] converters ) {
    this.mappingJackson2HttpMessageConverter = Stream.of( converters )
        .filter( hmc -> hmc instanceof MappingJackson2HttpMessageConverter
).findAny().get();
  }
  @BeforeEach
```

```
  public void setup() {
    this.mvc = MockMvcBuilders.standaloneSetup( authenticationController )
        .setControllerAdvice( new ExceptionsHandler() ).build();
  }
  @Test
  public void loginWithWrongCredentialsTest() throws Exception {
    given( authenticationManager.authenticate( any() ) ).willThrow(
BadCredentialsException.class );
    this.mvc.perform( post( API_URL + "/login" )
            .contentType( MediaType.APPLICATION_JSON )
            .accept( MediaType.APPLICATION_JSON )
            .content( "{\"clientId\": \"gebruiker\", \"clientSecret\":
\"wachtwoord\"}" ) )
        .andExpect( status().isUnauthorized() );
  }

  @Test
  public void authenticationProcessTest() throws Exception {
    Authentication authentication = new UsernamePasswordAuthenticationToken(
"gebruiker", "wachtwoord");
    given( authenticationManager.authenticate( any() ) ).willReturn(
authentication );

    this.mvc.perform( post( API_URL + "/login" )
            .contentType( MediaType.APPLICATION_JSON )
            .accept( MediaType.APPLICATION_JSON )
            .content( "{\"clientId\": \"gebruiker\", \"clientSecret\":
\"wachtwoord\"}" ) )
        .andExpect( status().isOk() );

    verify( authenticationManager, times( 1 ) ).authenticate( authentication );
    verify( userAuthorizationService, times( 1 ) ).authenticateUser(
authentication );
  }
  @Test
  public void shouldReturnToken() throws Exception {
    AuthenticatedUser authenticatedUser =
AuthenticatedUserBuilder.start().build();
    given( authenticationManager.authenticate( any() ) )
        .willReturn( new UsernamePasswordAuthenticationToken(
authenticatedUser.getUsername(), "bla" ) );
    AuthenticationResponse response = mockAuthenticationResponse();
    this.mvc.perform( post( API_URL + "/login" )
          .contentType( MediaType.APPLICATION_JSON )
            .accept( MediaType.APPLICATION_JSON )
            .content( "{\"clientId\": \"gebruiker\", \"clientSecret\":
\"wachtwoord\"}" ) )
        .andExpect( status().isOk() )
        .andExpect( content().json( getJson( response ) ) );
  }
  @Test
  public void shouldRefreshToken() throws Exception {
    String refreshToken = "t63567otyj&&&&keneshterstan";
    User user = UserBuilder.start().build();
    when( refreshTokenService.getTokenData( "someToken" ) ).thenReturn( user );
    List< GrantedAuthority > authorityList = new ArrayList<>();
    authorityList.add( new SimpleGrantedAuthority( "ROLE_SUPPORT" ) );
    when( userDetailsService.authenticateUser( user ) )
        .thenReturn( AuthenticatedUserBuilder.start().role( Role.SESSION_USER
).build() );
    AuthenticationResponse response = mockAuthenticationResponse();
```

```
    this.mvc.perform( post( API_URL + "/refresh" )
            .contentType( MediaType.APPLICATION_JSON )
            .param( "refreshToken", refreshToken )
            .accept( MediaType.APPLICATION_JSON ) )
        .andExpect( status().isOk() )
        .andExpect( content().json( getJson( response ) ) );
  }
  private AuthenticationResponse mockAuthenticationResponse() {
    given( jwtService.generateAccessToken() ).willReturn( "testToken" );
    given( jwtService.generateRefreshToken() ).willReturn( "testRefreshToken" );
    given( jwtService.getAccessTokenExpirationInMinutes() ).willReturn( 5 );
    given( jwtService.getRefreshTokenExpirationInMinutes() ).willReturn( 8 );
    return new AuthenticationResponse( "testToken", "testRefreshToken", 5, 8 );
  }
  @Test
  public void shouldLogoutByToken() throws Exception {
    String token = "tokenedfs";
    this.mvc.perform( post( API_URL + "/logout" ).contentType(
MediaType.APPLICATION_JSON )
        .param( "refreshToken", token ).accept( MediaType.APPLICATION_JSON )
).andExpect( status().isOk() );
    verify( refreshTokenService ).delete( token );
  }
  @Test
  public void wrongRefreshTokenTest() throws Exception {
    doWrongRefreshTokenTest( new JwtExpirationException( "" ) );
    doWrongRefreshTokenTest( new JwtBadSignatureException( "" ) );
    doWrongRefreshTokenTest( new MalformedJwtException( "" ) );
  }
  public void doWrongRefreshTokenTest( RuntimeException jwtException ) throws
Exception {
    String refreshToken = "t63567otyj&&&&keneshterstan";
    willThrow( jwtException ).willDoNothing().given( jwtService ).verifyToken(
refreshToken );
    this.mvc.perform( post( API_URL + "/refresh" ).contentType(
MediaType.APPLICATION_JSON ).param( "refreshToken",
        refreshToken ) ).andExpect( status().isUnauthorized() );
    verify( refreshTokenService, never() ).getTokenData( refreshToken );
  }
  @Test
  public void removeRefreshTokenOnTokenExpired() throws Exception {
    String refreshToken = "xcv";
    willThrow( new JwtExpirationException( "" ) ).willDoNothing().given(
jwtService ).verifyToken( refreshToken );
    this.mvc.perform( post( API_URL + "/refresh" )
        .contentType( MediaType.APPLICATION_JSON )
        .param( "refreshToken", refreshToken ) )
    .andExpect( status().isUnauthorized() );
    verify( refreshTokenService, never() ).getTokenData( refreshToken );
    verify( refreshTokenService, times( 1 ) ).delete( refreshToken );
  }
  protected String getJson( Object object ) throws IOException {
    MockHttpOutputMessage mockHttpOutputMessage = new MockHttpOutputMessage();
    this.mappingJackson2HttpMessageConverter.write( object,
MediaType.APPLICATION_JSON, mockHttpOutputMessage );
    return mockHttpOutputMessage.getBodyAsString();
  }
}


package com.voltor.futureleave.api.v1.controller;
```

```
public class ResultCaptor<T> implements Answer<T> {
  private T result = null;
  public T getResult() {
    return result;
  }
  public void setResult(T result) {
    this.result = result;
  }
  @SuppressWarnings("unchecked")
  @Override
  public T answer(InvocationOnMock invocationOnMock) throws Throwable {
    result = (T) invocationOnMock.callRealMethod();
    return result;
  }
}
package com.voltor.futureleave.api.v1.controller;

@WithMockUser(roles = { "SESSION_USER" })
@ActiveProfiles(profiles = { "develop" })
@Import({ ValidatorConfig.class, DummySecurityConfiguration.class,
SpringSecurityConfig.class,
    MappingJackson2HttpMessageConverter.class })
public abstract class SpringBasedControllerTest {
  private HttpMessageConverter<Object> mappingJackson2HttpMessageConverter;
  @Autowired
  protected MockMvc mvc;
  @Autowired
  protected void setConverters(HttpMessageConverter<Object>[] converters) {
    this.mappingJackson2HttpMessageConverter = Stream.of(converters)
        .filter(hmc -> hmc instanceof MappingJackson2HttpMessageConverter)
        .findAny().get();
  }
  /* util */
  protected String getJson(Object object) throws IOException {
    MockHttpOutputMessage mockHttpOutputMessage = new MockHttpOutputMessage();
    this.mappingJackson2HttpMessageConverter.write(object,
MediaType.APPLICATION_JSON, mockHttpOutputMessage);
    return mockHttpOutputMessage.getBodyAsString();
  }
}


package com.voltor.futureleave.api.v1.period;

@WebMvcTest( PeriodController.class )
public class PeriodControllerTest extends AbstractCRUDControllerTest<Period,
PeriodRequest, PeriodResponse>{

  @MockBean
  private PeriodService periodService;

  @SpyBean
  private PeriodSpecificationBuilder specificationBuilder;

  @Override
  protected Class<Period> getEntityClass() {
    return Period.class;
  }
  @Override
  protected AbstractService<Period> getService() {
    return this.periodService;
```

```
  }
  @Override
  protected Period getNewEntity() {
    return PeriodBuilder.start().build();
  }
  @Override
  protected PeriodRequest getNewRequestBean() {
    return convertToRequest( getNewEntity() );
  }
  @Override
  protected Map<List<String>, Sort> getSortingTestParameters() {
    Map< List< String >, Sort > parameters = new HashMap<>();
    parameters.put( List.of( "startDate" ), Sort.by( Sort.Direction.ASC,
"startDate") );
    parameters.put( List.of( "startDate,DESC" ), Sort.by( Sort.Direction.DESC,
"startDate") );
    parameters.put( List.of( "endDate" ), Sort.by( Sort.Direction.ASC, "endDate")
);
    parameters.put( List.of( "endDate,DESC" ), Sort.by( Sort.Direction.DESC,
"endDate") );

    parameters.put( List.of( "plannedEventsCount" ), Sort.by( Sort.Direction.ASC,
"plannedEventsCount") );
    parameters.put( List.of( "plannedEventsCount,DESC" ), Sort.by(
Sort.Direction.DESC, "plannedEventsCount") );

    parameters.put( List.of( "unplannedEventsCount" ), Sort.by(
Sort.Direction.ASC, "unplannedEventsCount") );
    parameters.put( List.of( "unplannedEventsCount,DESC" ), Sort.by(
Sort.Direction.DESC, "unplannedEventsCount") );
    return parameters;
  }
  @Override
  protected List<Function< Period, Object >> getValueToBeUpdated( PeriodRequest
request ) {
    List< Function< Period, Object >  > parameters = new LinkedList<>();
    parameters.add( Period::getStartDate );
    parameters.add( Period::getEndDate );
    parameters.add( Period::getPlannedEventsCount );
    parameters.add( Period::getUnplannedEventsCount );
    return parameters;
  }
  @Override
  protected Map< List< String >, List< SearchCriteria > >
getFilteringTestParameters() {
    Map<List<String>, List<SearchCriteria>>  filteringTestParameters = new
HashMap<>();
//    filteringTestParameters.put(
//        List.of( "sessionId=9556","sessionId!=65885" ),
//        List.of( new SearchCriteria("sessionId", FilteringOperation.EQUAL,
"9556"),
//            new SearchCriteria("sessionId", FilteringOperation.NOT_EQUAL,
"65885") ) );
    return filteringTestParameters;
  }
  @Override
  protected EntityFilterSpecificationsBuilder<Period> getSpecificationsBuilder() {
    return specificationBuilder;
  }
  @Override
```

```
  protected HashMap<Consumer<PeriodRequest>, Pair<Function<Period, Object>,
Function<Period, Object>>> getCreateOptionalFieldsTestParameters() {
    return new HashMap<>();
  }
  @Override
  protected Map<Map<String, Supplier<Object>>, String>
getPatchExceptionsValuesTestParameters() {
    Map< Map< String, Supplier< Object > >, String > parameters = new HashMap<>();
    return parameters;
  }
  @Override
  protected Map<Map<String, Object>, Pair<Function<Period, Object>, Object>>
getPatchValuesTestParameters() {
    Map< Map< String, Object >, Pair< Function< Period, Object >, Object > >
parameters = new HashMap<>();
    parameters.put(
        Map.of( "startDate", "2022-01-01" ),
        Pair.of( Period :: getStartDate, LocalDate.of(2022, 1, 1) ) );

    parameters.put(
        Map.of( "endDate", "2022-01-01" ),
        Pair.of( Period :: getEndDate, LocalDate.of(2022, 1, 1) ) );

    parameters.put(
        Map.of( "plannedEventsCount", "21" ),
        Pair.of( Period :: getPlannedEventsCount, 21 ) );

    parameters.put(
        Map.of( "unplannedEventsCount", "23" ),
        Pair.of( Period :: getUnplannedEventsCount, 23 ) );
    return parameters;
  }
  @Override
  protected PeriodRequest convertToRequest( Period period ) {
    PeriodRequest periodRequest = new PeriodRequest();
    periodRequest.setSessionId( period.getSessionId() );
    periodRequest.setStartDate( period.getStartDate() );
    periodRequest.setEndDate( period.getEndDate() );
    periodRequest.setPlannedEventsCount( period.getPlannedEventsCount() );
    periodRequest.setUnplannedEventsCount( period.getUnplannedEventsCount() );
    return periodRequest;
  }
  @Override
  protected PeriodResponse convertToResponse( Period period ) {
    PeriodResponse periodResponse = new PeriodResponse( period );
    periodResponse.setSessionId( period.getSessionId() );
    periodResponse.setStartDate( period.getStartDate() );
    periodResponse.setEndDate( period.getEndDate() );
    periodResponse.setPlannedEventsCount( period.getPlannedEventsCount() );
    periodResponse.setUnplannedEventsCount( period.getUnplannedEventsCount() );
    return periodResponse;
  }
  @Override
  protected Map<Consumer<PeriodRequest>, String>
getCreateWithWrongValuesTestParameters() {
    return  getGeneralWrongValuesTestParameters();
  }

  protected Map< Consumer< PeriodRequest >, String >
getGeneralWrongValuesTestParameters() {
    return new HashMap<>();
```

```
  }
  @Override
  protected String getControllerPath() {
    return ApiConstants.V1_PERIOD_ENDPOINT;
  }
}


package com.voltor.futureleave.api.v1.session;

@WebMvcTest( SessionController.class )
public class SessionControllerTest extends AbstractCRUDControllerTest<Session,
SessionRequest, SessionResponse>{

  @MockBean
  private SessionService sessionService;

  @SpyBean
  private SessionSpecificationBuilder specificationBuilder;

  @Override
  protected Class<Session> getEntityClass() {
    return Session.class;
  }
  @Override
  protected AbstractService<Session> getService() {
    return this.sessionService;
  }
  @Override
  protected Session getNewEntity() {
    return SessionBuilder.start().build();
  }
  @Override
  protected SessionRequest getNewRequestBean() {
    return convertToRequest( getNewEntity() );
  }
  @Override
  protected Map<List<String>, Sort> getSortingTestParameters() {
    Map< List< String >, Sort > parameters = new HashMap<>();
    parameters.put( List.of( "name" ), Sort.by( Sort.Direction.ASC, "name") );
    parameters.put( List.of( "name,DESC" ), Sort.by( Sort.Direction.DESC, "name")
);
    return parameters;
  }
  @Override
  protected List<Function< Session, Object >> getValueToBeUpdated( SessionRequest
request ) {
    List< Function< Session, Object >  > parameters = new LinkedList<>();
    parameters.add( Session::getName );
    return parameters;
  }
  @Override
  protected Map< List< String >, List< SearchCriteria > >
getFilteringTestParameters() {
    Map<List<String>, List<SearchCriteria>>  filteringTestParameters = new
HashMap<>();
    filteringTestParameters.put(
        List.of( "name=9556","name!=65885" ),
        List.of( new SearchCriteria("name", FilteringOperation.EQUAL, "9556"),
            new SearchCriteria("name", FilteringOperation.NOT_EQUAL, "65885") ) );
    return filteringTestParameters;
```

```
  }
  @Override
  protected EntityFilterSpecificationsBuilder<Session> getSpecificationsBuilder()
{
    return specificationBuilder;
  }
  @Override
  protected HashMap<Consumer<SessionRequest>, Pair<Function<Session, Object>,
Function<Session, Object>>> getCreateOptionalFieldsTestParameters() {
    return new HashMap<>();
  }
  @Override
  protected Map<Map<String, Supplier<Object>>, String>
getPatchExceptionsValuesTestParameters() {
    Map< Map< String, Supplier< Object > >, String > parameters = new HashMap<>();
    return parameters;
  }
  @Override
  protected Map<Map<String, Object>, Pair<Function<Session, Object>, Object>>
getPatchValuesTestParameters() {
    Map< Map< String, Object >, Pair< Function< Session, Object >, Object > >
parameters = new HashMap<>();
    parameters.put(
        Map.of( "name", "test2mers" ),
        Pair.of( Session :: getName, "test2mers" ) );
    return parameters;
  }
  @Override
  protected SessionRequest convertToRequest( Session session ) {
    SessionRequest sessionRequest = new SessionRequest();
    sessionRequest.setName( session.getName() );
    return sessionRequest;
  }
  @Override
  protected SessionResponse convertToResponse( Session session ) {
    SessionResponse sessionResponse = new SessionResponse( session );
    sessionResponse.setName( session.getName() );
    return sessionResponse;
  }
  @Override
  protected Map<Consumer<SessionRequest>, String>
getCreateWithWrongValuesTestParameters() {
    return  getGeneralWrongValuesTestParameters();
  }

  protected Map< Consumer< SessionRequest >, String >
getGeneralWrongValuesTestParameters() {
    return new HashMap<>();
  }
  @Override
  protected String getControllerPath() {
    return ApiConstants.V1_SESSION_ENDPOINT;
  }
}


package com.voltor.futureleave.builder;

public class AuthenticatedUserBuilder {
  private User user;
  private Role role;
```

```
  public AuthenticatedUserBuilder() {
    initDefaultData();
  }

  public AuthenticatedUser build() {
      User user = this.user;
      Role role = this.role;
      user.setUserRole(role);
    List<GrantedAuthority> authorityList = List.of( new
SimpleGrantedAuthority("ROLE_" + role) );
    AuthenticatedUser authenticatedUser = new AuthenticatedUser( user, role, true,
true, true, true, authorityList );
    initDefaultData();
    return authenticatedUser;
  }

  public AuthenticatedUser mock( AuthenticatedUserService userAuthorizationService
) {
    AuthenticatedUser authUser = build();
    given( userAuthorizationService.isRoot() ).willReturn( Role.ROOT.equals( role
) );
    given( userAuthorizationService.isSupport() ).willReturn(
Role.SESSION_USER.equals( role ) );
    given( userAuthorizationService.getCurrentUser() ).willReturn( user );
    given( userAuthorizationService.getCurrentUserId() ).willReturn( user.getId()
);
    return authUser;
  }

  public void login() {
    AuthenticatedUser user = build();
    UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken( user,
        user.getUsername() );
    SecurityContextHolder.getContext().setAuthentication( authentication );
  }

  public static AuthenticatedUserBuilder start() {
      return new AuthenticatedUserBuilder();
  }
  private void initDefaultData() {
    this.user = UserBuilder.start().build();
    this.role = Role.ROOT;
  }

  public AuthenticatedUserBuilder role( Role role ) {
    this.role = role;
    return this;
  }

  public AuthenticatedUserBuilder user( User user ) {
    this.user = user;
    return this;
  }

}


package com.voltor.futureleave.builder;

@Component
```

```
public class PeriodBuilder {
  private Period period;
  public PeriodBuilder() {
    initDefaultData();
  }
  public Period build() {
    Period token = this.period;
    initDefaultData();
    return token;
  }

  public Period buildNew() {
    return setUser(null).setId(null).build();
  }
  private void initDefaultData() {
    period = new Period();
    Long randomValue = ThreadLocalRandom.current().nextLong(1, 999999);
    int randomSmallValue = ThreadLocalRandom.current().nextInt(1, 99);
    setId( randomValue );
    setUser( UserBuilder.start().build() );
    setSessionId( randomValue );
    setStartDate(LocalDate.now().plusDays(randomSmallValue));
    setEndDate(period.getStartDate().plusMonths(1));
    setPlannedEventsCount(randomSmallValue);
    setUnplannedEventsCount(randomSmallValue + 10);
  }

  public static PeriodBuilder start() {
      return new PeriodBuilder();
  }

  public PeriodBuilder setId( Long id ) {
    period.setId( id );
    return this;
  }

  public PeriodBuilder setUser( User user ) {
    period.setUser( user );
    return this;
  }

  public PeriodBuilder setStartDate(LocalDate startDate) {
    period.setStartDate( startDate );
    return this;
  }

  public PeriodBuilder setEndDate(LocalDate endDate) {
    period.setEndDate( endDate );
    return this;
  }
  public PeriodBuilder setPlannedEventsCount(int plannedEventsCount) {
    period.setPlannedEventsCount( plannedEventsCount );
    return this;
  }

  public PeriodBuilder setUnplannedEventsCount(int unplannedEventsCount) {
    period.setUnplannedEventsCount( unplannedEventsCount );
    return this;
  }
  public PeriodBuilder setSessionId(Long sessionId) {
    period.setSessionId( sessionId );
```

```
    return this;
  }
}
```

```
package com.voltor.futureleave.builder;

@Component
public class RefreshTokenBuilder {
  private RefreshToken refreshToken;
  public RefreshTokenBuilder() {
    initDefaultData();
  }
  public RefreshToken build() {
    RefreshToken token = this.refreshToken;
    initDefaultData();
    return token;
  }

  public RefreshToken buildNew() {
    return setId(null).build();
  }
  private void initDefaultData() {
    refreshToken = new RefreshToken();
    Long randomValue = ThreadLocalRandom.current().nextLong(1, 999999);
    refreshToken.setId( randomValue );
    refreshToken.setToken( "SomeToken" + randomValue );
    refreshToken.setExpiringDate( DateTimeService.now().plusDays(2) );
    refreshToken.setUser( UserBuilder.start().build() );
  }

  public static RefreshTokenBuilder start() {
      return new RefreshTokenBuilder();
  }

  public RefreshTokenBuilder setId( Long id ) {
    refreshToken.setId( id );
    return this;
  }

  public RefreshTokenBuilder setUser( User user ) {
    refreshToken.setUser( user );
    return this;
  }
  public RefreshTokenBuilder setRefreshToken(String tokenValue) {
    refreshToken.setToken( tokenValue );
    return this;
  }
  public RefreshTokenBuilder setExpiringDate(ZonedDateTime expiringDate) {
    refreshToken.setExpiringDate( expiringDate );
    return this;
  }
}
```

```
package com.voltor.futureleave.builder;

@Component
public class SessionBuilder {

  @Autowired protected SessionService sessionService;
```

```
  private Session session = new Session();
  public SessionBuilder() {
    initDefaultData();
  }

  // Spring based
  public Session toDB() {
    Session session = buildNew();
    return sessionService.create( session );
  }
  // Spring based
  public SessionBuilder initDefaultDBRelations() {

    return this;
  }
  public Session build() {
    Session session = this.session;
    initDefaultData();
    return session;
  }

  public Session buildNew() {
    Session session = build();
    session.setId( null );
    return session;
  }
  private void initDefaultData() {
    Long randomValue = ThreadLocalRandom.current().nextLong(1, 999999);
    session = new Session();
    setId( randomValue );
    setName( "someName-" + randomValue );
  }

  public static SessionBuilder start() {
      return new SessionBuilder();
  }

  public SessionBuilder setId(Long id) {
    session.setId( id );
    return this;
  }
  public SessionBuilder setName( String name ) {
    session.setName( name );
    return this;
  }
}
package com.voltor.futureleave.builder;

@Component
public class UserBuilder {
  @Autowired private UserService userService;

  private User user;
  public UserBuilder() {
    initDefaultData();
  }

  public static UserBuilder start() {
      return new UserBuilder();
  }
```

```
  public User build() {
    User entity = this.user;
    initDefaultData();
    return entity;
  }
  public User buildNew() {
    return setId(null).build();
  }

  //Spring based
  public User toDB() {
    User user = buildNew();
    return userService.create( user );
  }
  private void initDefaultData() {
    Long randomValue = ThreadLocalRandom.current().nextLong(1, 999999);
    user = new User();
    setId( randomValue );
    setLastName( "LastName_" + randomValue );
    setEmail( "Email_" + randomValue + "@test.test" );
    setFirstName( "FirstName_" + randomValue );
    setLogin( "Login_" + randomValue );
    setPassword( "Password_" + randomValue );
    setRole( Role.SESSION_USER );
  }
  public UserBuilder setId( Long id ) {
    user.setId( id );
    return this;
  }
  public UserBuilder setLastName( String lastName ) {
    user.setLastName( lastName );
    return this;
  }

  public UserBuilder setFirstName( String firstName ) {
    user.setFirstName( firstName );
    return this;
  }
  public UserBuilder setEmail( String email ) {
    user.setEmail( email );
    return this;
  }

  public UserBuilder setLogin( String login ) {
    user.setLogin( login );
    return this;
  }

  public UserBuilder setRole( Role role ) {
    user.setUserRole( role );
    return this;
  }

  public UserBuilder setPassword( String password ) {
    user.setPassword( password );
    return this;
  }
}
package com.voltor.futureleave.config;

public class SpringBootTestClassOrderer implements ClassOrderer {
```

```
  @Override
  public void orderClasses( ClassOrdererContext classOrdererContext ) {
    classOrdererContext.getClassDescriptors()
        .sort( Comparator.comparingInt( SpringBootTestClassOrderer::getOrder ) );
  }
  private static int getOrder( ClassDescriptor classDescriptor ) {
    if ( classDescriptor.findAnnotation( SpringBootTest.class ).isPresent() ) {
      return 4;
    }
    return 1;
  }
}
package com.voltor.futureleave.dao;

public class RefreshTokenDAOTest extends AbstractIdentifiableDaoTest<RefreshToken,
RefreshTokenDao, RefreshTokenRepository> {
  @Mock
  private RefreshTokenRepository refreshTokenRepository;
  private RefreshTokenBuilder refreshTokenBuilder = new RefreshTokenBuilder();
  @Override
  public RefreshToken createEntity() {
    return refreshTokenBuilder.build();
  }
  @Override
  public RefreshTokenDao createDao() {
    return new RefreshTokenDao(userAuthorizationService, refreshTokenRepository);
  }
  @Override
  public RefreshTokenRepository createRepository() {
    return refreshTokenRepository;
  }
  @Disabled("Not supported")
  @Test
  @Override
  public void testRootAbleToEdit() {
    fail();
  }
}
package com.voltor.futureleave.dao;


public class SessionDaoTest extends AbstractIdentifiableDaoTest< Session,
SessionDao, SessionRepository > {
  @Mock
  private SessionRepository sessionRepository;
  @Override
  public Session createEntity() {
    return SessionBuilder.start().build();
  }
  @Override
  public SessionDao createDao() {
    return new SessionDao( userAuthorizationService, sessionRepository );
  }
  @Override
  public SessionRepository createRepository() {
    return sessionRepository;
  }

  @Override
  @SuppressWarnings("unchecked")
```

```
  public void shouldReturnPageableRecordsIncludeArchived() {
    AuthenticatedUserBuilder.start().role( Role.ROOT ).mock(
userAuthorizationService );
    given( createRepository().findAll( any( Specification.class ), any(
Pageable.class ) ) )
        .willReturn( new PageImpl<>( Collections.singletonList( entity ) ) );
    ArchivedSpecification< Session > archivedSpecification = mock(
ArchivedSpecification.class );
    PageRequest request = PageRequest.of( 0, 1, Sort.unsorted() );
    daoService.get( request, true );
    verify( createRepository() ).findAll( any( Specification.class ), any(
Pageable.class ) );
    verifyNoInteractions( archivedSpecification );
  }
  @Override
  public void shouldReturnAllSortedRecordsIncludeArchived() {
    AuthenticatedUserBuilder.start().role( Role.ROOT ).mock(
userAuthorizationService );
    super.shouldReturnAllSortedRecordsIncludeArchived();
  }
  @Override
  public void shouldReturnAllFilteredSortedRecordsWithIncludeArchived() {
    AuthenticatedUserBuilder.start().role( Role.ROOT ).mock(
userAuthorizationService );
    super.shouldReturnAllFilteredSortedRecordsWithIncludeArchived();
  }
  @Override
  public void shouldReturnPageableFilteredRecordsWithIncludeArchived() {
    AuthenticatedUserBuilder.start().role( Role.ROOT ).mock(
userAuthorizationService );
    super.shouldReturnPageableFilteredRecordsWithIncludeArchived();
  }
}
package com.voltor.futureleave.dao.shared;


/**
 * Use this as the base class for Data DAO JPA Tests when testing commonly used
implementations
 */
@DataJpaTest
@ActiveProfiles(profiles = {"develop", "h2"})
public abstract class AbstractIdentifiableDaoDataJpaTest<T extends Identifiable> {
  @Autowired
  private EntityManager entityManager;
  @Autowired
  private SessionRepository sessionRepository;
  /** Mockito Bean Mocks */
  @MockBean
  protected AuthenticatedUserService userAuthorizationService;
  @Mock
  private PasswordEncoder passwordEncoder;

  /**
   * Returns the JPA EntityManager
   * @return {@link jakarta.persistence.EntityManager}
   */
  protected EntityManager getEntityManager(){
    return this.entityManager;
  }
  /**
```

```
   * Returns the SessionRepository
   * @return {@link SessionRepository}
   */
  protected SessionRepository getSessionRepository() {
    return sessionRepository;
  }
  /**
   * A mock of the UserAuthorizationService
   * @return {@link AuthenticatedUserService}
   */
  protected AuthenticatedUserService getUserAuthorizationService() {
    return this.userAuthorizationService;
  }
  /**
   * The actual Spring Managed Entity Repository Bean
   * @return {@link BaseCRUDRepository}
   */
  protected abstract BaseCRUDRepository<T> getRepository();
  /**  Start {@link EntityType} Common tests */
  /**
   * Create a new entity to validate.
   * Important! Make sure there is at least one property that can break Bean
Validation.
   * @return
   */
  protected abstract T createInvalidEntity();
  /**
   * Creates a new Valid entity to persist before breaking constraint violation
step
   * @return entity
   */
  protected abstract T createValidEntity();
  /**
   * Returns an invalid entity that should break one or more constraint violations
   * @return entity
   */
  protected abstract T updateInvalidEntity(T createdEntity);
  @Test
  public void onCreateShouldThrowConstraintViolationException() {
    AuthenticatedUserBuilder.start().role( Role.ROOT ).mock(
userAuthorizationService );
    T createInvalidEntity = createInvalidEntity();
    assertThrows( ConstraintViolationException.class, () ->
getRepository().save(createInvalidEntity) );
  }
  @Test
  public void onUpdateShouldThrowConstraintViolationException() {
    AuthenticatedUserBuilder.start().role( Role.ROOT ).mock(
userAuthorizationService );
    T entity = createValidEntity();
    getRepository().save(entity);
    T persistedEntity = getRepository().findById(entity.getId()).get();
    T updateInvalidEntity = updateInvalidEntity(persistedEntity);
    assertThrows( ConstraintViolationException.class, () -> {
      getRepository().save(updateInvalidEntity);
      entityManager.flush();
    } );
  }
}
package com.voltor.futureleave.dao.shared;
```

```
@SuppressWarnings("unchecked")
public abstract class AbstractIdentifiableDaoTest<EntityType extends Identifiable,
  DaoType extends AbstractIdentifiableDao<EntityType>,
  RepositoryType extends BaseCRUDRepository<EntityType>> {

  @Mock
  protected AuthenticatedUserService userAuthorizationService;
  protected DaoType daoService;
  protected EntityType entity;
  public abstract EntityType createEntity();
  public abstract DaoType createDao();
  public abstract RepositoryType createRepository();
  @BeforeEach
  public void setup() throws Exception {
    MockitoAnnotations.openMocks(this).close();
    daoService = createDao();
    entity = createEntity();
    entity.setId( 1L );
    AuthenticatedUserBuilder.start().role( Role.SESSION_USER ).mock(
userAuthorizationService );
  }
  @Test
  public void testRootAbleToEdit() {
    assertTrue( daoService.isEditAllowed( entity ) );
  }
  @Test
  public void shouldReturnPageableRecordsIncludeArchived() {
    given( createRepository().findAll( any( Specification.class ), any(
Pageable.class ) ) )
        .willReturn( new PageImpl<>( Collections.singletonList( entity ) ) );
    ArchivedSpecification< EntityType > archivedSpecification = mock(
ArchivedSpecification.class );
    PageRequest request = PageRequest.of( 0, 1, Sort.unsorted() );
    daoService.get( request, true );
    verify( createRepository() ).findAll( any( Specification.class ), any(
Pageable.class ) );
    verifyNoInteractions( archivedSpecification );
  }
  @Test
  public void shouldReturnAllSortedRecordsIncludeArchived() {
    given( createRepository().findAll( any( Specification.class ), any( Sort.class
) ) )
        .willReturn( Collections.singletonList( entity ) );
    ArchivedSpecification< EntityType > archivedSpecification = mock(
ArchivedSpecification.class );
    daoService.get( Sort.unsorted(), true );
    verify( createRepository() ).findAll( any( Specification.class ), any(
Sort.class ) );
    verifyNoInteractions(archivedSpecification);
  }
  @Test
  public void shouldReturnAllFilteredSortedRecordsWithIncludeArchived() {
    given( createRepository().findAll( any( Specification.class ), any( Sort.class
) ) )
        .willReturn( Collections.singletonList( entity ) );
    ArchivedSpecification< EntityType > archivedSpecification = mock(
ArchivedSpecification.class );
    Specification< EntityType > specification = new EqualingSpecification<>(
        new SearchCriteria( "name", FilteringOperation.EQUAL, "entity" ) );
    daoService.get( specification, Sort.unsorted(), true );
```

```
    verify( createRepository() ).findAll( any( Specification.class ), any(
Sort.class ) );
    verifyNoInteractions( archivedSpecification );
  }
  @Test
  public void shouldReturnPageableFilteredRecordsWithIncludeArchived() {
    given( createRepository().findAll( any( Specification.class ), any(
Pageable.class ) ) )
        .willReturn( new PageImpl<>( Collections.singletonList( entity ) ) );
    ArchivedSpecification< EntityType > archivedSpecification = mock(
ArchivedSpecification.class );
    Specification< EntityType > specification = new EqualingSpecification<>(
        new SearchCriteria( "name", FilteringOperation.EQUAL, "entity" ) );
    PageRequest request = PageRequest.of( 0, 1, Sort.unsorted() );
    daoService.get( specification, request, true );
    verify( createRepository() ).findAll( any( Specification.class ), any(
Pageable.class ) );
    verifyNoInteractions( archivedSpecification );
  }
}
package com.voltor.futureleave.security;

@TestConfiguration
public class DummySecurityConfiguration {
  @Bean
  public UserDetailsService userDetailsService() {
    return Mockito.mock(UserDetailsServiceImpl.class);
  }
  @Bean
  public JwtService jwtService() {
    return Mockito.mock(JwtService.class);
  }
  @Bean
  public SessionDao sessionDao() {
    return Mockito.mock(SessionDao.class);
  }
}
package com.voltor.futureleave.security.jwt;


@SpringBootTest
@ActiveProfiles(profiles = {"develop", "h2"})
public class JwtServiceTest {
  @MockBean
  private RefreshTokenService refreshTokenService;
  @Autowired
  private JwtService jwtService;
  @Test
  public void shouldSaveRefreshToken() {
    String token = jwtService.generateRefreshToken();
    assertNotNull(token);
    verify(refreshTokenService).create(any(String.class),
any(ZonedDateTime.class));
  }
}
package com.voltor.futureleave.service;
@SpringBootTest
@ActiveProfiles(profiles = {"develop", "h2"})
public class AbstractServiceDBTest {

  @Autowired protected UserBuilder userBuilder;
```

```
  @Autowired protected AuthenticatedUserService authenticatedUserService;
  @BeforeEach
  void createUser() {
    loginInUser();
  }

  void loginInUser() {
    loginInUser( Role.SESSION_USER  );
  }

  void loginInUser( Role role ) {
    AuthenticatedUserBuilder.start().role( role).login();
  }

  void logoff() {
    SecurityContextHolder.clearContext();
  }
}


package com.voltor.futureleave.service;

/** Base class for most recent Service (DAO) tests */
public abstract class BaseDaoServiceTest<
  T extends Identifiable,
  Service extends AbstractService< T >,
  Dao extends AbstractIdentifiableDao< T > > {

  @Mock
  AuthenticatedUserService userAuthorizationService;

  protected T entity;
  protected Class<T> entityClass;
  abstract void additionalSetup();
  abstract Service getService();
  abstract Dao getDao();
  abstract T createEntity();
  @SuppressWarnings("unchecked")
  @BeforeEach
  void setup() throws Exception {
    MockitoAnnotations.openMocks( this ).close();
    AuthenticatedUserBuilder.start().role( Role.SESSION_USER ).mock(
userAuthorizationService );
    additionalSetup();
    entity = createEntity();
    entityClass = (Class< T >) entity.getClass();
  }
  @Test
  void shouldCreateEntity() {
    given( getDao().create( any( entityClass ) ) ).willReturn( entity );
    ArgumentCaptor< T > argumentCaptor = ArgumentCaptor.forClass( entityClass );
    getService().create( entity );
    verify( getDao() ).create( argumentCaptor.capture() );
    assertEquals( entity, argumentCaptor.getValue() );
  }
  @Test
  void shouldUpdateEntity() {
    given( getDao().update( any( entityClass ) ) ).willReturn( entity );
    ArgumentCaptor< T > argumentCaptor = ArgumentCaptor.forClass( entityClass );
    getService().update( entity );
    verify( getDao() ).update( argumentCaptor.capture(), anyBoolean() );
```

```
    assertEquals( entity, argumentCaptor.getValue() );
  }
  @Test
  void shouldDeleteEntity() {
    given( getDao().getOne( entity.getId() ) ).willReturn( entity );
    ArgumentCaptor< T > argumentCaptor = ArgumentCaptor.forClass( entityClass );
    getService().delete( entity.getId() );
    verify( getDao() ).delete( argumentCaptor.capture(), anyBoolean() );
    assertEquals( entity, argumentCaptor.getValue() );
  }
}
package com.voltor.futureleave.service;
@AutoConfigureMockMvc
@SpringBootTest
@ActiveProfiles(profiles = {"develop"})
public abstract class BaseSpringServiceTest {
}
package com.voltor.futureleave.service;

public class DateTimeServiceTest {
  @Test
  public void shouldReturnNowWithUTC() {
    assertEquals(ZoneOffset.UTC, DateTimeService.now().getOffset());
  }
  @Test
  public void shouldReturnNowWithOffset() {
    assertEquals(ZoneOffset.MIN, DateTimeService.now(ZoneOffset.MIN).getOffset());
  }
  @Test
  public void shouldConvertToUTC() {
    ZonedDateTime dateTimeWithDifferentOffset = ZonedDateTime.now(ZoneOffset.MIN);
    assertEquals(ZoneOffset.UTC,
DateTimeService.toUTC(dateTimeWithDifferentOffset).getOffset());
  }
}
package com.voltor.futureleave.service.djl;

class TrainModelServiceTest {
  Trainer trainModelService = new Trainer();
  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");

  @Test
  void learnModelSimpleTest() throws Exception {
    trainModelService.learnBase();

    Period s = new Period();
    Integer total = 124;
    Integer unplanned = 105;
    s.setPlannedEventsCount(total - unplanned);
    s.setUnplannedEventsCount(unplanned);
    s.setStartDate(LocalDateTime.parse("2023-08-21 00:00:00",
formatter).toLocalDate());
    s.setEndDate(LocalDateTime.parse("2023-09-18 00:00:00",
formatter).toLocalDate());

    System.out.println( trainModelService.getPredict(s) );
  }
}
```

```
package com.voltor.futureleave.service;
class PeriodServiceDBTest extends AbstractServiceDBTest{

  @Autowired private PeriodService periodService;
  @Autowired private PeriodBuilder builder;

  @Test
  void simpleCreateTest() {
    Period period = builder.buildNew();
    Period result =  periodService.create(period);
    assertNotNull(result.getId());
    assertNotEquals( 0L, result.getId());
  }
}
```

```
package com.voltor.futureleave.service;
class RefreshTokenDBTest extends AbstractServiceDBTest{

  @Autowired
  private RefreshTokenService refreshTokenService;

  @Autowired
  private RefreshTokenBuilder refreshTokenBuilder;
  @Test
  public void shouldThrowExceptionDueToConstraint() {
    RefreshToken token = refreshTokenBuilder.buildNew();
    refreshTokenService.create(token);
    RefreshToken secondToken = refreshTokenBuilder.setRefreshToken(
token.getToken() ).buildNew();
    Exception exception = assertThrows(DataIntegrityViolationException.class,
        () -> refreshTokenService.create(secondToken));
    assertNotNull(exception);
  }
}
```

```
package com.voltor.futureleave.service;
public class RefreshTokenServiceTest extends BaseDaoServiceTest< RefreshToken,
RefreshTokenService, RefreshTokenDao > {
  @InjectMocks
  private RefreshTokenService refreshTokenService;
  @Mock
  private RefreshTokenDao refreshTokenDao;

  @Mock
  private UserService userService;
  @Override
  public void additionalSetup() {
    User currentUser = userAuthorizationService.getCurrentUser();
    when( userService.getOne( currentUser.getId() ) ).thenReturn( currentUser );
  }
  @Override
  public RefreshTokenService getService() {
    return refreshTokenService;
  }
  @Override
  public RefreshTokenDao getDao() {
    return refreshTokenDao;
  }
  @Override
```

```
  public RefreshToken createEntity() {
    return RefreshTokenBuilder.start().setUser(
userAuthorizationService.getCurrentUser() ).build();
  }
  @Test
  public void setCurrentSessionTest() {
    given( getDao().create( any() ) ).willAnswer( i -> i.getArguments()[0] );
    RefreshToken refreshToken = getService().create( createEntity() );
    assertEquals( userAuthorizationService.getCurrentUser(),
refreshToken.getUser() );
  }
  @Test
  @SuppressWarnings("unchecked")
  public void returnAuthDatAndDeleteTest() {
    User user = UserBuilder.start().build();
    AuthenticatedUserBuilder.start().role( Role.SESSION_USER ).user( user ).mock(
userAuthorizationService );
    when( userService.findByLogin(
userAuthorizationService.getCurrentUser().getLogin() ) )
      .thenReturn( user );

    RefreshToken refreshToken = createEntity();

    when( getDao().getOne( any( Specification.class ) ) ).thenReturn( refreshToken
);
    when( getDao().getOne( refreshToken.getId() ) ).thenReturn( refreshToken );

    User result = getService().getTokenData( "someToken" );
    assertEquals( refreshToken.getUser(), result );
    verify( getDao() ).delete( eq( refreshToken), anyBoolean() );
  }
  @Test
  public void shouldThrowException() {
    Exception exception = assertThrows( RefreshTokenNotFoundException.class,
        () -> getService().getTokenData( "UnexistingToken" ) );
    assertNotNull( exception );
  }
  @Test
  @SuppressWarnings("unchecked")
  public void shouldDeleteByToken() {
    RefreshToken refreshToken = createEntity();
    when( getDao().getOne( any( Specification.class ) ) ).thenReturn( refreshToken
);
    getService().delete( refreshToken.getToken() );
    verify( getDao(), times( 1 ) ).delete( refreshToken, true );
  }
}


package com.voltor.futureleave.service;
class SessionServiceDBTest extends AbstractServiceDBTest{

  @Autowired
  private SessionService sessionService;

  @Autowired
  private SessionBuilder sessionBuilder;

  @Test
  public void shouldNotAllowEditingIfCurrentUserIsNotRoot() {
    Session session = sessionBuilder.toDB();
```

```
    loginInUser( Role.SESSION_USER  );
    assertFalse( sessionService.editingIsAllowed( session ) );
  }
  @Test
  public void shouldNotAllowDeleteIfCurrentUserIsNotRoot() {
    loginInUser( Role.SESSION_USER  );
    assertThrows( ActionNotAllowedException.class, () -> sessionService.delete(
1L, true ) );
  }

}


package com.voltor.futureleave.service.user;
@SpringBootTest
@ActiveProfiles(profiles = { "develop" })
public class UserAuthorizationServiceTest {
  @Autowired
  private AuthenticatedUserService userAuthorizationService;
  @MockBean
  private SecurityContext securityContext;
  @MockBean
  private PermissionEvaluator permissionEvaluator;
  @MockBean
  private Authentication authentication;

  @BeforeEach
  public void setup() {
    SecurityContextHolder.setContext( securityContext );
    given( securityContext.getAuthentication() ).willReturn( authentication );
  }
  @Test
  public void noAuthenticationTest() {
    given( securityContext.getAuthentication() ).willReturn( null );
    assertThrows( NoCurrentUserException.class, () ->
userAuthorizationService.getCurrentUser() );
  }
  @Test
  public void wrongAuthenticationTest() {
    Object objectWithWrongPrincipalType = new Object();
    given( securityContext.getAuthentication().getPrincipal() ).willReturn(
objectWithWrongPrincipalType );
    assertThrows( NoCurrentUserException.class, () ->
userAuthorizationService.getCurrentUser() );
  }
  @Test
  public void rightAuthenticationTest() {
    AuthenticatedUser build = AuthenticatedUserBuilder.start().build();
    given( authentication.getPrincipal() ).willReturn( build );
    assertNotNull( userAuthorizationService.getCurrentUser() );
  }
  @Test
  public void rootRoleTest() {
    AuthenticatedUser build = AuthenticatedUserBuilder.start().role( Role.ROOT
).build();
    given( authentication.getPrincipal() ).willReturn( build );
    assertTrue( userAuthorizationService.isRoot() );
  }
  @Test
  public void supportRoleTest() {
```

```
    AuthenticatedUser build = AuthenticatedUserBuilder.start().role(
Role.SESSION_USER ).build();
    given( authentication.getPrincipal() ).willReturn( build );
    assertTrue( userAuthorizationService.isSupport() );
  }
}


package com.voltor.futureleave.service.user;
public class UserDetailsServiceTest {

  @Mock
  private UserService userService;

  @InjectMocks
  private UserDetailsServiceImpl userDetailsService;

  @BeforeEach
  public void setup() throws Exception {
    MockitoAnnotations.openMocks(this).close();
    userDetailsService = new UserDetailsServiceImpl( userService );
  }
  @Test
  public void testSessionUserRoleAuthority() {
    User user = UserBuilder.start().build();
    given( userService.findByLogin( anyString() ) ).willReturn( user );
    AuthenticatedUser ud = (AuthenticatedUser)
userDetailsService.loadUserByUsername( user.getLogin() );


    assertEquals( Role.SESSION_USER, ud.getRole() );
    assertEquals( user, ud.getUser() );
    assertTrue( ud.getAuthorities().contains( new SimpleGrantedAuthority(
"ROLE_SESSION_USER" ) ) );
  }
}


package com.voltor.futureleave.service;
class UserServiceDBTest extends AbstractServiceDBTest{

  @Autowired private UserService userService;
  @Autowired private UserBuilder userBuilder;
  @Autowired private PasswordEncoder encoder;

  @Test
  void encodePasswordOnCreate() {
    User user = userBuilder.buildNew();
    String password = user.getPassword();
    user = userService.create( user );
    assertTrue(  encoder.matches(  password, user.getPassword() ) );
  }

  @Test
  void encodePasswordOnUndate() {
    User user = userBuilder.toDB();
    String oldEncriptedPassword = user.getPassword();
    String newPassword = "bala bla test";
    user.setPassword( newPassword );
    user = userService.update( user );
    assertFalse(  encoder.matches(   newPassword, oldEncriptedPassword ) );
```

```
    assertTrue(  encoder.matches(  newPassword, user.getPassword() ) );
  }

  @Test
  void dontEncodePasswordOnUndate() {
    User user = userBuilder.buildNew();
    String password = user.getPassword();
    user = userService.create( user );
    String oldEncriptedPassword = user.getPassword();
    user = userService.update( user );
    assertEquals( oldEncriptedPassword, user.getPassword() );
    assertTrue(  encoder.matches(  password, oldEncriptedPassword ) );
    assertTrue( encoder.matches(  password, user.getPassword() ) );
  }

  @Test
  void checkloginUnique() {
    User user = userBuilder.toDB();
    assertTrue( userService.isLoginUnique( user ) );
    user.setId( 0L );
    assertFalse( userService.isLoginUnique( user ) );
  }

  @Test
  void findByLogin() {
    User user = userBuilder.toDB();
    userBuilder.toDB();
    userBuilder.toDB();
    logoff();
    User result = userService.findByLogin( user.getLogin() );
    assertEquals(user, result);
  }

}


package com.voltor.futureleave.service;

public class UserServiceTest {

  @Mock private AuthenticatedUserService userAuthorizationService;
  @Mock private UserDao dao;
  @Mock private SessionService sessionService;
  @InjectMocks private UserService service;

  @BeforeEach
  void setup() throws Exception {
    MockitoAnnotations.openMocks( this ).close();
    AuthenticatedUserBuilder.start().role( Role.SESSION_USER ).mock(
userAuthorizationService );
  }

  @Test
  void createOrUpdate() {
    User user = UserBuilder.start().buildNew();
    service.createOrUpdate( user );
    verify( dao, times( 1 ) ).create( user );
    verify( dao, times( 0 ) ).update( eq( user ), anyBoolean() );

    user = UserBuilder.start().build();
```

```
    when( dao.getOneArchived( eq( user.getId() ), anyBoolean() ) ).thenReturn(
user );
    service.createOrUpdate( user );
    verify( dao, times( 0 ) ).create( user );
    verify( dao, times( 1 ) ).update( eq( user ), anyBoolean() );
  }

}
```