math_2021_038

**Міністерство освіти і науки України**

**Чернівецький національний університет**

**імені Юрія Федьковича**

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

# Створення веб-додатку для проведення голосувань та опитувань на факультеті

**Дипломна робота**

**Рівень вищої освіти – другий (магістерський)**

Виконав:
студент 6 курсу, групи 607
спеціальності 124 – Системний аналіз
(назва спеціальності)
**Малярчук Михайло Михайлович**
(прізвище, ім'я, по-батькові)
Керівник: **доц. Сопронюк Т.М.**
(науковий ступінь, вчене звання, призвіще та ініціали)

До захисту допущено:

Протокол засідання кафедри № 6

від "7" грудня 2021 р.

Завідувач. кафедри _____ проф. Черевко І.М.

Чернівці - 2021

# Анотація

У магістерській роботі розглянуто сучасні веб-системи, технології та процеси їх розробки. Детально розглянуто з теоретичної та практичної сторін їх застосування для створення веб-додатку для проведення опитувань.

Здійснено розробку веб-додатку для проведення опитувань та голосувань з використанням технологій Kotlin, Spring, TypeScript та Angular. Додаток призначений для застосування в навчальному процесі в межах факультету.

# Зміст

# Вступ

**Актуальність роботи.** В наш час все більш поширеними стають різноманітні соціальні мережі та додатки всеосяжного призначення, які замінюють або автоматизують нецифрові процеси. Інформаційні технології застосовуються наразі практично у всіх сферах. Наприклад, одним з таких процесів є проведення опитування, котре зазвичай проводиться в усному або писемному форматі, а вже існуючі додатки часто мають платний функціонал, або є недоступними в українському секторі. Також слід зазначити, що не всі існуючі засоби для проведення онлайн опитувань та голосувань дозволяють з легкістю коригувати доступні варіанти, рівень анонімності опитування тощо.

**Метою** дипломної роботи є створення веб-додатку для проведення опитувань та голосувань з використанням мов програмування Kotlin та TypeScript, технологій фреймворків Spring та Angular відповідно.

**Завдання роботи** - розробка веб-додатоку для проведення різноманітних опитувань в межах факультету або навіть і поза ним. Основними вимогами до додатку була можливість обирати теми курсових робіт, варіанти лабораторних завдань тощо студентами, а також можливість проведення різноманітних опитувань серед студентів.

В ході дипломної роботи для досягнення мети розв’язано наступні задачі:

- проаналізовано можливості та необхідності застосування даного додатку в межах факультету;
- створено веб-додаток для проведення опитувань та голосувань засобами мов програмування Kotlin та TypeScript;
- протестовано розроблений веб-додаток.

Розроблений додаток складається з серверної та клієнтської частин, для зберігання даних було обрано реляційну базу даних. В ході роботи було

використано такі мови програмування як Kotlin, Java та TypeScript, мову структурованих запитів PostgreSQL, фреймворки Spring та Angular.

**Об’єктом дослідження** є сучасні веб-системи та технології їх розробки.

**Наукова новизна** роботи полягає в тому, що розроблений додаток створений з використанням сучасних технології розробки програмного забезпечення, включає різноманітні варіанти створення опитувань такі як можливість генерації доступних опцій, визначення рівня анонімності опитування тощо. **Практичне значення** полягає в тому, що створений веб-додаток може використовуватися на факультеті для проведення опитуваня для вибору варіантів лабораторних та курсових робіт, проведення голосувань серед студентів, або поза його межами.

Дипломна робота складається з вступу, трьох розділів, висновків, переліку посилань та додатків.

У першому розділі детальніше розглянуто використані при розробці додатку технології та їх можливості. У другому розділі детально описано етапи розробки веб-додатку, надано та описано фрагменти коду, описано проблеми з якими довелось стикнутись в процесі виконання роботи. В третьому розділі описано функціонал розробленого додатку, його особливості в порівнянні з існуючими аналогами та можливості практичного застосування.

В ході розробки додатку та написання дипломної роботи використано літературу та інтернет ресурси за посиланнями [1-7], написаний код зберігається в сервісі Github за посиланнями [8-9].

# Розділ 1. Огляд технологій сучасної веб-розробки

## 1.1. Сучасні веб-системи та технології

Розробка веб-додатків в наші дні містить величезний набір різних правил та прийомів, про які необхідно знати, якщо ви хочете щоб ваш веб-сайт виглядав і функціонував згідно вимог та побажань замовника. Для цього необхідно ознайомитись з веб-технологіями, які допоможуть в досягненні цієї мети. Веб-технології – це програмні засоби, мультимедійні пакети, мови програмування та розмітки, що використовуються комп'ютерами для спілкуванні в Інтернет-мережі.

Одними з таких програмних засобів є браузери – додатки, призначені для запиту необхідної інформації із серверів та відображення їх в доступному та зрозумілому вигляді. Найпоширенішими серед них є Google Chrome (наразі найпопулярніший браузер, що надається компанією Google), Safari (веб-браузер від компанії Apple), Firefox (браузер з відкритим вихідним кодом, що підтримується компанією Mozilla Foundation) та Edge (браузер від компанії Microsoft, що прийшов на заміну Internet Explorer).

Основними програмними засобами для розробки навіть найпростіших веб-сайтів є мова розмітки гіпертексту HTML (використовується для розмітки відображення інформації, що надсилається із серверу) та мова каскадних таблиць стилів CSS (визначає як саме елементи HTML будуть відображатися у браузері). При необхідності додавання певної логіки для опрацювання інформації чи відображення компонентів використовується мова програмування JavaScript (або її типізована версія TypeScript), яка за деякими підрахунками є найбільш популярною мовою програмування. Слід також зазначити, що в залежності від типу браузера та його версії деякі компоненти можуть працювати та відображатись по різному, тому якщо додаток має підтримувати різні браузери, або наприклад деякі їх застарілі версії, це становить певну проблему для веб-розробників.

В сучасній розробці програмного забезпечення (не тільки веб-додатків) широко застосовуються різні фреймворки (англ. *framework* – каркас, інфраструктура, платформа) – це певна сукупність, інфраструктура програмних рішень, що полегшує розробку складних програм та систем. На відміну від бібліотеки, з якою фреймворк дуже схожий, останній має ряд правил та обмежень. Також фреймворк може базуватись на інших програмних засобах або бібліотеках.

У сучасній веб-розробці широко застосовуються такі фреймворки як Angular (одна з найновіших веб-технологій, призначена для розробки динамічних веб-додатків, за допомогою якої можна з легкістю розробляти інтерфейсні програми без використанні інших фреймворків, бібліотек та плагінів), Ruby on Rails (технологія розробки веб-сайтів на серверній стороні з використанням мови програмування Ruby, яка надає широкий ряд переваг, який робить розробку та написання коду простішими та швидшими; до списку популярних веб-сайтів створених за допомогою цього фреймворку входять Basecamp, GitHub, Ask.fm тощо), Meteor JS (фреймворк з використанням Node.js, що дає змогу швидко створювати прості веб-додатки з використанням мови JavaScript), Django (один з найпопулярніших фреймворків з використанням мови програмування Python, що побудований на MVC архітектурі (англ. *Model-View-Controller* – система модель-відображення-контролер, скорочено MVC, що використовується в багатьох сучасних веб-системах), що дає змогу в дуже простій формі розробляти веб-додатки і надає безліч інструментів для зв’язку з базою даних, побудови шаблонів веб-сторінок тощо) та Laravel (фреймворк для розробки на мові програмування PHP, що підходить для розробки невеликих веб-сайтів, побудований на архітектурі MVC та надає низку вбудованих функцій, таких як підтримка ООП, авторизація, міграція бази даних тощо, і є одним з найпопулярніших та використовуваних фреймворків для мови PHP).

Наразі дуже поширеною є практика поділу додатку на клієнтську та серверні частини, що дозволяє багатьом клієнтами спілкуватись з одним серверним додатком. Описані вище технології використовуються здебільшого

для клієнтської частини (окрім фреймворків Django та Laravel). Слід також додати, що для серверної частини також можуть використовуватись мови сімейства Java (такі, як власне Java, а також Kotlin, Scala та Groovy) з використанням різноманітних фреймворків, таких як Spring (дозволяє легко створювати веб-додатки засобами мови Java з підтримкою Groovy та Kotlin як альтернативних мов, з вибором багатьох видів архітектур залежно від потреб замовника), Spring Boot (фреймворк, заснований на Spring, призначений для побудови автономних мікросервісів та серверних додатків), Spark (фреймворк для легкої побудови веб-додатків засобами Java та Kotlin) тощо, а також мови програмування Go, Python тощо.

### 1.2. Особливості мови програмування Kotlin

Як відомо, Java – строго типізована об’єктно-орієнтована мова програмування, випущена в 1995 році компанією Sun Microsystems, з 2009 року знаходиться в підпорядкування Oracle. Основною її особливістю є те, що написаний Java код компілюється в так званий байт-код (машинно-незалежний код низького рівня), який інтерпретується на віртуальній машині під потреби конкретної платформи. Таким чином, мова програмування Java є дійсно крос-платформеною, оскільки скомпільований байт-код може виконуватись однаково на будь-якій машині, на якій встановлено відповідну віртуальну машину з інтерпретатором. Відповідно засоби розробки та компіляції JDK (англ. *Java Development Kit* – набір засобів розробки на Java, скорочено JDK), віртуальна машина інтерпретації JVM (англ. *Java Virtual Machine* – віртуальна машина Java, скорочено JVM) та середовище виконання коду JRE (англ. *Java Runtime Environment* – середовище виконання Java, скорочено JRE) надаються компанією Oracle [1]. Слід зазначити, що ці перелічені компоненти повинні бути однієї версії та версія Java коду, на якій написана програма повинна не перевищувати версію цих компонентів.

Мова програмування Kotlin виникла як покращена версія мови Java. На її розробку вплинули такі мови програмування як Java (використання віртуальної машини та системи очистки невикористовуваної пам'яті), Python (гнучкість та легкість написання коду), C# (використання покращень введених відносно Java, наприклад, функцій розширення), Scala (перевизначення існуючих операторів), Groovy (вирішення проблеми безпеки від пустих вказівників), JavaScript (веб платформа як ціль мови програмування) та звісно такі мови програмування як C, C++ та Haskell. Як наслідок, отримали простий і зрозумілий синтаксис, підтримку не одної, а декількох парадигм (об'єктно-орієнтоване програмування, функціональне програмування та імперативне програмування), не тільки кросс-платформеність, а ще й мульти-платформеність (JVM – код Kotlin, як і Java код, компілюється в байт-код, який легко запускається на будь-яких платформах з встановленим JVM, Android – оскільки ця платформа має власне середовище виконання ART, код Kotlin також може компілюватися в відповідні файли, JavaScript – код Kotlin може компілюватися в JavaScript код для виконання у веб-браузері та власне компіляція в машинний код для виконання на специфічній платформі) [2]. Крім того, для програмування на мові Kotlin можна з легкістю використовувати існуючі Java бібліотеки, також він має вбудований плагін для перетворення Java коду в Kotlin код [3].

Далі перелічено основні відмінності мови програмування Kotlin від мови програмування Java [3-4]:

- поява типів безпечних від пустих вказівників (null safe), об'єкт такого типу обов'язково має бути ініціалізованим і не може містити посилання на пусту область пам'яті;
- можливість оголошення змінних, що можуть (ключове слово var) або не можуть (ключове слово val) бути модифікованими;
- можливість оголошення функцій за межами класів;
- вирази, такі як умовний оператор, можуть повертати значення, усунуто тернарний умовний оператор;
- уведено числові проміжки;

- оголошення полів класу прямо всередині конструктора;
- за замовчуванням клас або метод не може бути наслідуваним;
- поява функцій розширення (можна додати до існуючого класу метод, не наслідуючи його;
- поява найменованих аргументів при виклику функції або конструктора)
- поява класів даних, спеціально для зберігання унікальних одиниць об’єктів тощо.

В загальному, мова програмування Kotlin є більш сучасною відносно мови Java, містить в собі функціональності та особливості, запозичені з мов, де були перевірені, для максимальної зручності та ефективності використання.

## 1.3. Фреймворк Spring

### *1.3.1. Загальний опис. Модель MVC*

Фреймворк Spring призначений для розробки веб-додатків засобами мов програмування з виконанням коду в JVM, починаючи з версії 5.1 вимагає версію JDK не меншу на 8.60. Фреймворк має досить широкий спектр застосування: від монолітних додатків до незалежних мікросервісів, що працюють в хмарному середовищі, від додатків, що надають HTML код напряму браузеру користувача до сервісів, що надають дані у форматах JSON, XML тощо різним клієнтам у статичному або навіть динамічному форматі, має відкритий вихідний код та має велику спільноту розробників, які займаються його підтримкою та забезпечують неперервний зворотній зв’язок [5].

Фреймворк в загальному побудований на архітектурі MVC, проте дозволяє також з легкістю будувати різні API системи типів REST або GraphQL. Проект поділений на різні модулі, такі як Spring MVC (забезпечення відображення веб-сторінок, обміну інформацією з клієнтською стороною), Spring Security (модуль для забезпечення на конфігурації безпеки додатку), Spring Data (робота з даними, зв’язок із базою даних тощо), Spring Cloud (налаштування хмарних сервісів для

запуску веб-додатків), Spring Boot (модуль для легкої побудови та запуску мікросервісів) тощо. За необхідності модулі легко додаються до існуючого проекту.

Модель MVC у фреймворку Spring (як і в більшості інших систем, що підтримують цю архітектуру) складається із наступних складових (див. Рис. 1.1):

- модель (Model) – одиниця даних або певний об’єкт, що використовуються в програмі;

- інтерфейс користувача (View) слугує для візуалізації даних однієї або декількох моделей;

- контролер (Controller) – об’єкт, що слугує для отримання і опрацювання даних з клієнтської або іншого сервера і надсилає у відповідь дані в спеціальному форматі або переадресовує на інтерфейс користувача (View).


Рис. 1.1 – Діаграма роботи моделі MVC

Так як в загальному фреймворк Spring MVC побудований на основі технології Java Servlets, запити, які надійшли до додатку, спершу обробляються спеціальним сервлетом *DispatcherServlet*, який перевіряє чи є доступний контролер з методом, що відповідає URL адресі та вхідним параметрам і виконує

переадресацію на відповідний метод. Якщо відповідний контролер повертає назву певного відображення, то його буде надіслано до *ViewResolver* (конфігурація, яка по назві відображення надає сам його об'єкт, наприклад HTML або JSP файл), якщо ж контролер повертає тільки дані, вони будуть відправлені як відповідь у відповідному форматі.

### *1.3.2. Рівень доступу до даних*

Робота з базами даних у фреймворку Spring з легкістю здійснюється з використанням модуля Spring Data або Spring Data JPA. В поєднанні з об'єктно-реляційною системою (англ. *Object-Relational Mapping* – об'єктно-реляційне відображення, скорочено ORM) можна з легкістю налаштовувати зв'язок між таблицями бази даних та об'єктів у відповідній мові програмування [5].

Модуль Spring Data JPA має справу з розширеною підтримкою рівнів доступу до даних (модель, репозиторій тощо) на основі JPA (*Java Persistence API* - стандартизований інтерфейс для ORM фреймворків мови Java), що значно полегшує створення додатків на основі фреймворку Spring, яким необхідний зв'язок з базою даних. Протягом досить довгого проміжку часу реалізація рівня доступу до даних була досить громіздкою – необхідно було писати велику кількість подібного коду, імплементовувати логіку відображення результатів запиту в модель тощо. На зміну цьому прийшла легка реалізація рівнів доступу до даних, де зі сторони розробника немає необхідності в імплементації цього рівня, тільки його конфігурація та налаштування коду необхідних запитів.

Деякі переваги цієї системи:

- окрема конфігурація зв'язку за базою даних (при необхідності її можна з легкістю змінити, об'єкт зв'язку конфігурується окремо і постачається як залежність модуля Spring Data);
- з використанням зв'язку типу *ComboPooledDataSource* можна налаштувати ORM таким чином, щоб структура таблиць в базі даних

автоматично змінювалась відповідно до конфігурації об’єктів в коді програми;

- з використанням модуля Spring Data JPA та об’єктів, що імплементують інтерфейс *JpaRepository* можна створювати методи для отримання даних з бази без їх явної ініціалізації (яка відбувається засобами Spring Data вже на етапі виконання коду), наприклад за допомогою зазначення тільки запиту до бази, який цей метод повинен виконувати, або базуючись виключно на назві методу в специфічному форматі.

### *1.3.3. Налаштування безпеки. Spring Security*

Для сучасних веб-додатків дуже важливим елементом є налаштування його безпеки, які повинні зберігати дані користувачів належним чином, відхиляти неавторизовані або підозрілі запити, які не відповідають загальноприйнятим політикам, обмежувати доступ до певних частин додатку користувачам, що не повинні мати цього доступу тощо.

Для забезпечення безпеки веб-додатку застосовується модуль Spring Security, який дозволяє без великої кількості коду налаштувати політику безпеки додатку, різні види автентифікації користувача, знищення даних сесії, двофакторна автентифікація тощо. Власне процес автентифікації відбувається з використанням фільтрів сервлетів, які перехоплюють запити, виконуються певні операції над об’єктами HTTP запиту та відповіді та передають запит і відповідь наступним фільтрам в ланцюжку, оскільки їх може бути декілька. Саме під час цього процесу модуль Spring Security проводить процес автентифікації, блокує запити від неавторизованих користувачів та ті, що не відповідають політиці безпеки, додає необхідні заголовки до відповіді тощо.

Існує наступний перелік об’єктів, що приймають в цьому участь [5]:

- *AuthenticationFilter* – це HTTP фільтр, який власне й перехоплює запити та намагається їх автентифікувати;

- *AuthenticationManager* – основний програмний інтерфейс, що приймає запит, переданий описаним вище фільтром, та може передати його одному або декільком екземплярам інтерфейсу *AuthenticationProvider*;
- *AuthenticationProvider* – інтерфейс, реалізуючи який можна виконувати процес автентифікації запиту, переданого об'єктом *AuthenticationManager*, кожен тип автентифікації повинен мати власну реалізацію цього інтерфейсу;
- *UserDetailsService* – інтерфейс, призначений для реалізації розробником додатку для отримання даних про користувача з бази даних оперативної пам'яті тощо;
- *PasswordEncoder* – інтерфейс для шифрування паролів користувачів, є обов'язковим починаючи з версії Spring Security 5.

Для повного уявлення про зв'язок цих компонентів див. Рис. 1.2.

Кожен з цих об'єктів може бути перевизначеним та зареєстрованим в контексті Spring Security за необхідності. Крім того можливим є реєстрація власних фільтрів HTTP запитів, реєстрація власного фаєрволу тощо.

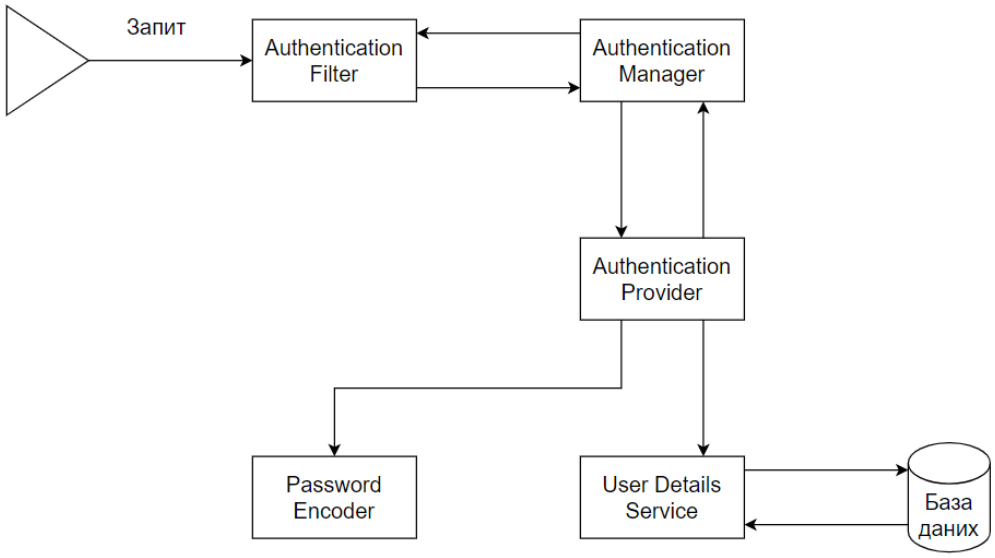


Рис. 1.2 – Діаграма зв'язку компонентів Spring Security

В загальному використання фреймворку Spring Security має наступні переваги:

- підтримка базової HTTP автентифікації;
- підтримка OAuth 2 автентифікації, можливість налаштувань авторизації через соціальні мережі;
- підтримка налаштувань політик CORS та CSRF;
- підтримка авторизації через форму;
- знищення даних сесії при виході з додатку;
- можливість налаштування переадресації після успішного або неуспішного входу в додаток;
- шифрування паролів користувачів;
- можливість реєстрації власних фільтрів запитів тощо.

Політика CORS (англ. *Cross-origin resource sharing* – перехресний доступ до ресурсів, скорочено CORS) – це, як слідує з назви, політика обмеження доступу до певних ресурсів третіми сторонами. В загальному, це механізм, що дозволяє запитувати обмежені ресурси іншого домену, що знаходиться за межами домену, з якого відбувається запит. Зазвичай за замовчуванням такі “міждоменні” запити є забороненими, позаяк налаштування політики CORS дозволяє їх проведення. Наприклад, можна обмежити доступ до серверу тільки з певної клієнтської частини або іншого сервера, або ж обмежити список можливих джерел, на які буде цей сервер посилатись, або обмежити типи вхідних HTTP запитів до сервера тощо. Для методів, які можуть змінювати дані (наприклад, POST або DELETE), політика вимагає, щоб браузери попередньо виконували запит методом OPTIONS для перевірки списку доступних методів перед виконанням основного запиту.

### 1.4. Особливості мови програмування TypeScript

Мова програмування TypeScript тісно пов’язана з мовою JavaScript, проте має свої відмінності: вона пропонує всі функції та можливості JavaScript, а крім них надає додаткову функціональність – систему типів. Тобто ця мова

програмування є типізованою. Проте в той же час існуючий JavaScript код в той же час є і TypeScript кодом, з тою відмінністю що інтерпретатор покаже всі помилки пов’язані з типізацією даних, що дозволить уникнути багатьох помилок в ході виконання програми [6].

В загальному ситуація схожа з мовами програмування Kotlin та Java, хоча TypeScript в порівнянні з своїм родичом має набагато меншу кількість відмінностей, і всі вони пов’язані з типізацією даних. Знаючи принципи роботи JavaScript можна з легкістю побудувати систему, яка буде приймати існуючий код, проте додаючи до неї типи, без необхідності додавання спеціальних символів чи змін синтаксису мови програмування. Для цього Typescript підтримує розширення мови JavaScript, яке дозволяє вказувати типи в потрібних місцях коду [6].

Набір примітивних типів мови JavaScript (*bigint*, *number*, *string*, *symbol*, *boolean*, *null* та *undefined*) було розширено з додаванням таких типів як: *any* (дозволити будь-який тип), *unknown* (невизначений тип), *never* (об’єкт цього типу не може бути створеним) та *void* (пустий тип для функції, що нічого не повертає).

Далі перелічено основні відмінності мови TypeScript від JavaScript:

- явна типізація даних (задана в коді);
- неявна типізація даних (при ініціалізації змінної);
- можливість створення класів та інтерфейсів;
- можливість створення власних композитних типів даних використовуючи наявні примітивні;
- можливість створення загальних (generic) класів та інтерфейсів до певних типів тощо.

В загальному використання мови програмування TypeScript для веб-розробки дозволяє уникнути багатьох проблем та помилок, що стаються під час використання мови JavaScript та пов’язані з типізацією, невчасною ініціалізацією змінних тощо.

### 1.5. Фреймворк Angular

Angular – це платформа розробки веб-додатків, що побудована на мові програмування TypeScript, яка включає [7]:

- компонентно орієнтований фреймворк для створення масштабованих та динамічних веб-додатків;
- колекцію інтегрованих бібліотек та модулів з широким функціоналом, включаючи клієнт-серверний зв’язок, переадресацію та маршрутизацію всередині додатку, керування формами тощо;
- набір інструментів для полегшення розробки, редагування та тестування коду.

Цей фреймворк широко застосовується у веб-розробці починаючи від невеликих проектів і закінчуючи масштабними корпоративними проектами. Своєрідна екосистема цього фреймворку включає більше 1.7 мільйона розробників, авторів бібліотек та модулів тощо. Основними особливостями та перевагами цього фреймворку є:

- поділ функціоналу на незалежні між собою компоненти;
- можливість перевикористання шаблонів сторінок та цілих компонентів для уникнення дуплікації коду та легкої внесення змін;
- простота додавання та встановлення нових бібліотек та залежностей через засоби командної стрічки;
- простота налаштування маршрутизації між різними компонентами всередині додатку;
- можливість динамічного формування контенту веб-сторінок, виведення даних тощо;
- легкість розробки та внесення оновлень (немає необхідності перезапускати сервер чи оновлювати ресурси, що значно пришвидшує час розробки).

Компонентна модель фреймворку забезпечує інкапсуляцію даних та інтуїтивно зрозумілу структуру проекту, полегшує модульне тестування і

покращує загальну читабельність коду [7]. Це є прикладом саме тих обмежень, які накладає фреймворк на розробника (на відміну від простої бібліотеки), проте значно полегшує процес розробки та розуміння структури іншими. Кожен компонент включає в себе клас мови програмування TypeScript, в якому знаходиться вся логіка компоненту, шаблон HTML та файл CSS зі стилями, підключеними в шаблоні. Кожен компонент можна підключити в шаблоні будь-якого іншого компонента за допомогою унікального HTML тегу, згенерованого для кожного компоненту. Слід зазначити, що забороненим є рекурсивне підключення компонентів.

Як було описано вище, кожен компонент має шаблон веб-сторінки у HTML файлі. Вказується він або відносним шляхом до файлу (можна підключати шаблони і які не є в одній директорії з класом компоненту) або прямо в самому класі компоненту. Фреймворк Angular розширює стандартні можливості HTML за допомогою додаткового синтаксису, який дозволяє підставляти динамічні дані з класу компоненту (які будуть оновлені на сторінці динамічно під час їх оновлення в коді програми), додавати умови для виведення даних, цикли тощо, додавати додаткові атрибути до HTML компонентів для синхронізацією їх з станом компонента, проведенням перевірок тощо. Так, можна вказати, що значення певних елементів форми будуть відповідати стану полів об’єкта збереженого в компоненті чи зробити перевірку чи доступне збереження даних форми в залежності від того чи коректно заповнені її поля тощо.

За допомогою засобів командної стрічки Angular CLI можна за допомогою команд ініціалізовувати нові компоненти з усіма необхідними файлами, додавати нові залежності та бібліотеки, запускати чи зупиняти сервер, проводити компіляцію коду, запускати модульні тести тощо, що значно пришвидшує процес розробки додатку.

# Розділ 2. Створення веб-додатку

## 2.1. Архітектурне рішення

Роботу над додатком було розпочато з вибору архітектури проекту. Перш за все необхідно було місце для зберігання даних (файлове сховище або база даних). Також був необхідний механізм комунікації з цим сховищем для передачі даних до користувача додатку та механізм обробки цих даних на клієнтській стороні. Отож в архітектурному плані працювати було необхідно з трьома ключовими елементами: базою даних (є кращим варіантом ніж файлове сховище через більшу швидкодію запитів), серверною та клієнтськими частинами. На Рис. 2.1 наведено розроблену діаграму комунікації цих частин.

Роботу над додатком було розпочато з вибору оптимальних технологій для бази даних, серверної та клієнтської частини. Для бази даних вибір стояв між реляційною та нереляційною технологіями, позаяк було обрано реляційну, яка має ряд переваг, таких як: більший вибір можливостей під'єднання до бази даних з використанням різних мов програмування, можливість побудови прямих зв'язків між таблицями та індексації значень для пришвидшення виконання запитів, можливість подальшого розгортання структури бази даних тощо. Отож для бази даних було обрано технологію PostgreSQL через широкий набір можливостей побудови запитів та легкість розгортання на будь-якому середовищі.

Для серверної частини вибір технології базувався на виборі перш за все фреймворку, а не мови програмування. Найбільшими кандидатами були фреймворки Spring (сумісний з мовами програмування що використовують байт-код, такими як Java, Kotlin, Groovy та Scala) та Django (сумісний тільки з мовою програмування Python). Отож було обрано перший через сумісність з більшою кількістю мов програмування, легкістю розгортання на Tomcat сервері. В якості

мови програмування для серверної частини використовувалася мова Kotlin, також було показано можливість інтегрування частин коду написаних на Java.

Для клієнтської (браузерної) частини виріб стояв між мовами програмування JavaScript та TypeScript, відповідно і між фремворками React та Angular. Було обрано другий варіант через можливість типізації даних в мові програмування TypeScript для полегшення інтеграції з серверною частиною.

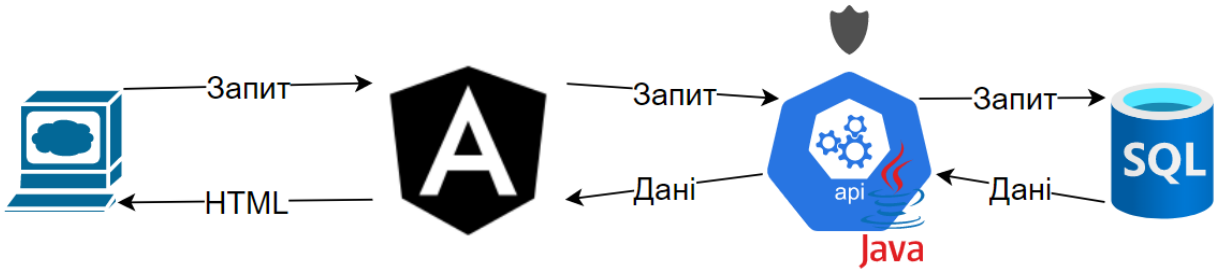


Рис 2.1 – Діаграма зв’язку частин додатку.

### 2.2. Структура бази даних

Після вибору технологій розпочалася робота над розробкою структури бази даних. Було вирішено розробити схему таким чином, щоб дозволити розробку якомога більшої кількості функціональностей додатку без внесення змін до структури бази даних (додавання нових таблиць, полів, функцій тощо). Тобто, необхідно було забезпечити можливість створення декількох опитувань в межах одного посту за необхідності, чи наприклад створення посту взагалі без опитувань, додавання необмеженої кількості можливих опцій для голосування та забезпечення їх варіативності, можливості генерації потрібних варіантів. Також необхідно було забезпечити можливість надання доступу до контенту певним користувачам або групам користувачів в додатку.

Розроблені таблиці корегувались допоки не досягли необхідного для функціональності додатку вигляду. Практично всі таблиці мають первинний ключ, який слугує ідентифікатором кожного запису в таблиці, окрім тих які

наслідують (доповнюють) інші таблиці (в цьому випадку ключем слугує вторинний ключ на первинний ключ батьківської таблиці).

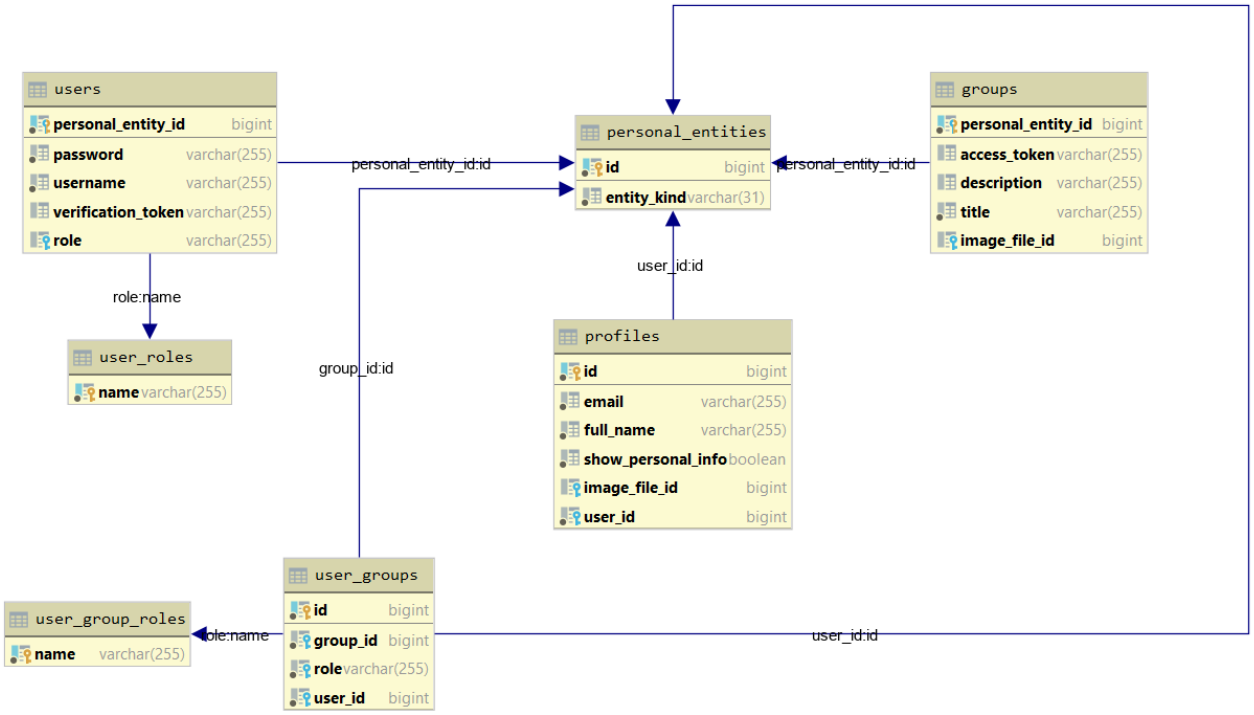


Рис. 2.2 – Структура таблиць пов’язаних з збереженням даних користувачів.

На Рис. 2.2 позначено схему таблиць пов’язаних з користувацькими даними. Можна побачити, що таблиці *users* (зберігає дані про користувачів) та *groups* (зберігає дані про групи користувачів) є нащадками таблиці *personal_entity*, яка щоправда має тільки одне поле – унікальний ідентифікатор, проте така структура дозволяє в будь-якому потрібному місці прирівняти зв’язок з таблицею користувачів і таблицею груп, що дає змогу на рівні коду визначити з яким саме об’єктом маємо справу і проектувати логіку необхідним чином.

Своєрідним розширенням до таблиці *users* є таблиця *profiles*, яка зберігає дані профілю користувача, такі як повне ім’я, адреса електронної пошти тощо, і пов’язана з першою зовнішнім ключем (ти зв’язку один-до-одного). Чому було обрано створити окрему таблицю замість збереження даних профілю в таблиці користувачів? Тому що остання використовується як збереження логіну / паролю користувача в додатку для автентифікації в додатку і на цьому етапі нам

абсолютно немає необхідності знати дані профілю користувача, а витягнення зайвих даних завжди супроводжується певними проблемами, такими як їх подальша обробка, збільшення часу виконання запиту тощо. За необхідності, знаючи ідентифікатор авторизованого користувача в системі, можна з легкістю знайти інформацію про його профіль в базі даних.

На Рис. 2.3 позначено схему таблиць, пов'язаних з даними про опитування і їх проходження користувачами. Як можна побачити, інформація про опитування зберігається в таблиці *polls*, а доступні опції зберігаються в таблиці *poll_options*, яка зв'язана з першою за допомогою зовнішнього ключа методом багато-до-одного. За допомогою проміжної таблиці (тип зв'язку багато-до-багатьох) *poll_reactions* можна зберігати дані про голосування користувачем, прив'язавши за допомогою ключа його ідентифікатор до ідентифікатору опції опитування.

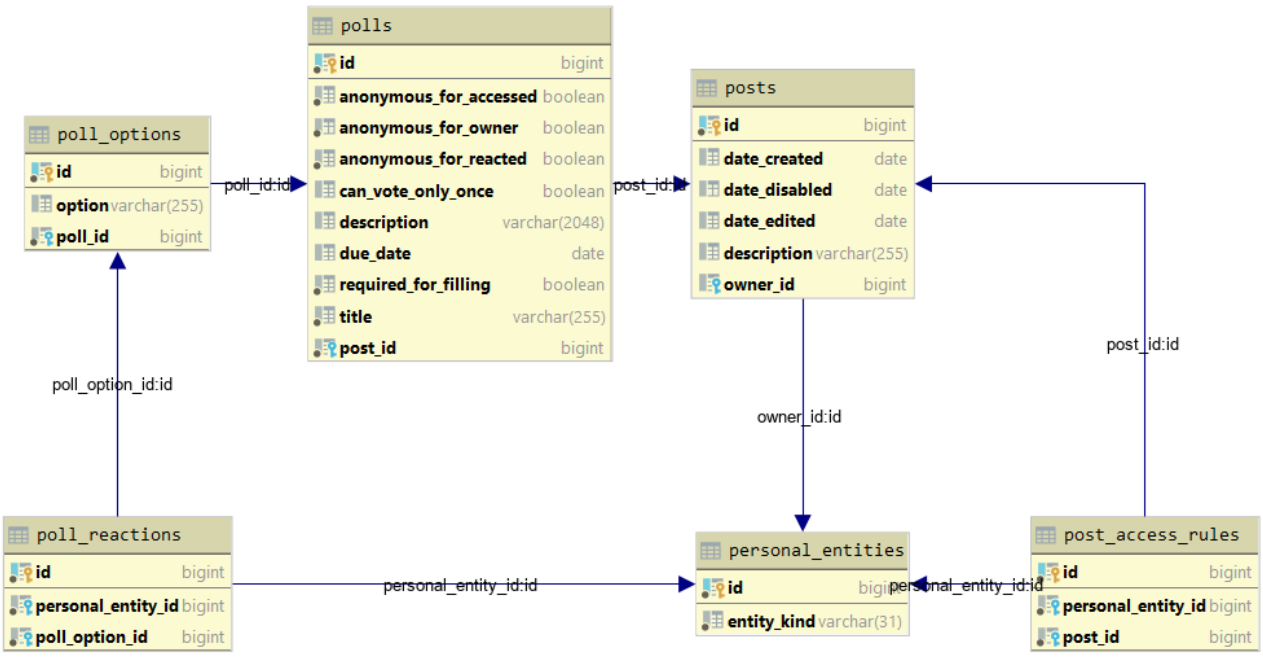


Рис 2.3 – Структура таблиць, пов'язаних з даними про опитування.

В ході розробки схеми бази даних було прийнято рішення прив'язати об'єкт опитування до певного об'єкту таблиці *posts* який репрезентує запис /

допис користувача в додатку. Таким чином з точки зору бази даних можна зв’язати декілька опитувань з одним дописом користувача, або взагалі створити допис без опитування при необхідності. Також за допомогою проміжної таблиці *post_access_rules* можна обмежувати доступ до певного допису з набором опитувань, обмежуючись певними користувачами або групами користувачів. Це залишає дві можливості реалізації доступу для опитування, для якого не вказано записи в цій таблиці: або ж опитування доступне всім користувачам а додатку, або нікому.

В загальному, підсумовуючи вище описане, структура бази даних була розроблена щоб бути максимально гнучкою та загальною для будь-яких потреб додатку.

### 2.3. Зв’язок з базою даних

Для зв’язку з базою даних було використано інтерфейс Java DataBase Connectivity (англ. *З’єднання з базою даних на Java*, скорочено JDBC), призначений для взаємодії Java застосунків з різними системами управління базами даних (скорочено СУБД). В загальному, можна сказати, що використання JDBC для зв’язку з базами даних в Java-орієнтованих додатках є очевидним через те, що цей пакет входить до складу Java SE (стандартний пакет для розробки програм).

Власне для безпосередньої взаємодії з базою даних було використано ComboPooledDataSource, який дозволяє створювати або змінювати структуру таблиць відповідно до конфігурації сутностей в коді програми. Для забезпечення з’єднання достатньо вказати назву драйвера для відповідної СУБД, посилання на хост бази даних з її назвою у форматі JDBC URL, облікові дані для авторизації, а також мінімальну / максимальну кількість потоків зв’язків об’єкту ComboPooledDataSource, і з’єднання буде встановлено за вказаною конфігурацією. А конфігурація сутностей, що фактично є відображеннями таблиць, відбувалась за допомогою ORM (англ. *Object-Relational Mapping* –

*Об'єктно-реляційне відображення* – технологія в програмуванні, що забезпечує зв'язок бази даних з концепціями об'єктно-орієнтованих мов програмування) інтерфейсу Java Persistence API (скорочено JPA) з реалізацією Hibernate. На Рис. 2.4 показано діаграму зв'язку з СУБД засобами JPA.

Для створення логіки на рівні доступу до об'єктів бази даних (англ. *Data Access Object Layer*, скорочено DAO рівень) було використано модуль Spring Data JPA, який дозволяє створювати реалізації інтерфейсів зв'язку з базою даних без їх фізичної реалізації. Відповідно до конфігурації інтерфейсів в коді програми модуль автоматично створює відповідні реалізації та підставляє їх в місця використання. Тобто на рівні коду ви працюєте з інтерфейсом, а на рівні виконання коду – з класами реалізаціями цих інтерфейсів, створених автоматично модулем Spring Data JPA.

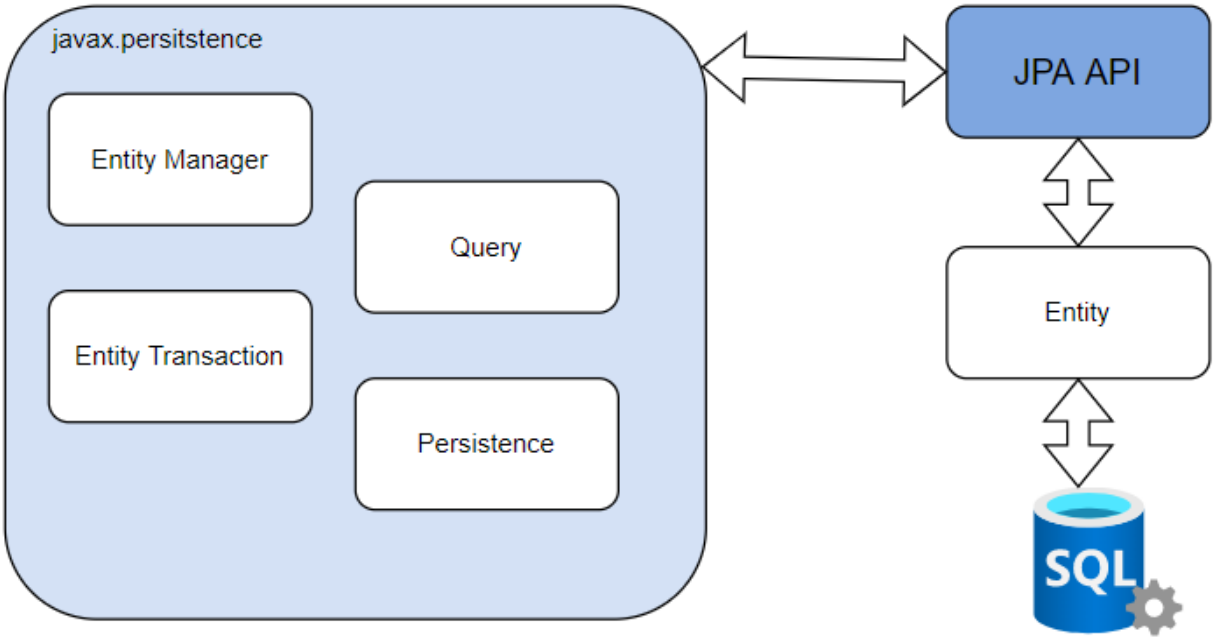


Рис. 2.4 – Діаграма зв'язку з СУБД засобами JPA

Цей підхід було обрано через те, що він значно спрощує встановлення з'єднання з базою даних, процеси міграції на нову версію (структура таблиць буде автоматично відповідати сутностям в коді програми, звісно якщо модифікація буде неможливою через існуючі зовнішні ключі та зв'язки,

необхідно буде провести додавання, видалення або модифікацію цих даних безпосередньо в СУБД). Звісно, при певних умовах, використання ORM може уповільнити виконання запитів до бази даних через виконання відображень та перетворень, або наприклад через витягнення непотрібних для поточного запиту даних тощо, проте при правильній конфігурації та використанні власноруч написаних запитів різницю у часі виконання можна звести до мінімуму. Хоча при великих об’ємах передачі даних навіть найменша різниця набуде ширшого масштабу, у випадку створення звичайного веб-додатку без необхідності такої передачі такий підхід є абсолютно виправданим з точки зору легкої конфігурації та модифікації, зручності використання. Проте потрібно зазначити, що цей аналіз потрібно проводити на етапі проектування архітектурного рішення для системи.

```
@Repository
interface GroupRepository: JpaRepository<Group, Long> {

    fun findByTitle(title: String): Group?

    @Query(value = "SELECT DISTINCT title FROM groups", nativeQuery = true)
    fun getAvailableGroupNames(): List<String>
```

В коді вище відображено інтерфейс *GroupRepository*, який наслідує інтерфейс *JpaRepository*, вказуючи, що сутністю є клас *Group*, а тип первинного ключа – Long. Анотація *@Repository* вказує, що цей інтерфейс є на рівні доступу до бази і, знаючи що він наслідує *JpaRepository*, фреймворк автоматично створить клас, що його імплементує та його об’єкт. Метод *findByTitle* призначений для пошуку об’єкту групи користувачів за її назвою. Він не має реалізації в коді, проте базуючись на його назві, Spring Data JPA автоматично створить реалізацію цього методу в самостійно створеному класі нащадку.

В методі *getAvailableGroupNames* продемонстровано трохи інший варіант, а саме – задання запиту до бази шляхом додавання анотації *@Query*. Вказаний запит (на прикладі відповідає за одержання доступних назв груп для відображення на користувацькій стороні) буде виконуватись при виклику цього методу. За допомогою параметру *nativeQuery = true*, ми вказуємо, що цей запит

написаний на тій же мові, яка використовуються в СУБД (в даному випадку – PostgreSQL), оскільки за замовчуванням використовуються HQL (англ. *Hibernate Query Language* – мова запитів Hibernate). Повний код цього інтерфейсу можна знайти в додатку А.

### 2.4. Бізнес логіка, контролери

Різноманітна логіка, пов’язана з створенням об’єктів, реєстрацією та авторизацією в додатку, відправкою електронних листів тощо була винесена в окремий сервісний рівень для забезпечення її незалежності від рівня доступу до бази даних, конфігураційного та контролер рівнів. Так, було створено декілька сервісів, наприклад для операцій пов’язаних з одним користувачем, групою користувачів, опитуваннями, відправкою електронних листів тощо.

```
@Service
class GroupService (
    @Autowired val groupRepository: GroupRepository,
    @Autowired val userGroupRepository: UserGroupRepository,
    @Autowired val userGroupRoleRepository: UserGroupRoleRepository,
    @Autowired val userRepository: UserRepository,
    @Autowired val pollRepository: PollRepository
) {
```

В коді вище вказано опис (сигнатуру) класу *GroupService*, відповідального за логіку, пов’язану з створенням, модифікацією груп користувачів, отриманням інформації про них, додавання користувача до групи тощо. Тут сам клас позначений анотацією *@Service*, за допомогою якої фреймворк Spring визначає що це клас сервісного рівня і автоматично створить тільки один об’єкт цього класу і підставить на етапі виконання у всі необхідні місця в програмному коді. Як можна побачити, цей клас використовує п’ять об’єктів рівня доступу до бази даних, ін’єкцію цих об’єктів забезпечує анотація *@Autowired*. Об’єкти, позначені цією анотацією будуть автоматично підставлені фреймворком на етапі виконання коду. Повний код вказаного класу можна знайти в додатку Б.

На рівні контролерів, як і на сервісному, було створено декілька класів для кожної частини функціональності додатку.

```
@RestController
@RequestMapping("/group")
class GroupController (
      @Autowired val groupService: GroupService
) {

   private val logger: Logger = LogManager.getLogger(this.javaClass)

   @PostMapping("/create")
   fun createGroup(@RequestBody groupDto: GroupDto): GenericResponse<GroupDto> {
      return try {
         GenericResponse.of(groupService.createGroup(groupDto))
      } catch (ex: Exception) {
         processException(ex)
      }
   }
```

В коді вище вказано сигнатуру класу *GroupController* та одного з його методів *createGroup*, призначеного для створення групи користувачів. Як можна побачити, клас позначений анотацією *@RestController*, яка визначає його як клас-контролер, тобто обробник HTTP запитів, що надходять до додатку. Анотація *@RequestMapping(“/group”)* позначає, що цей контролер буде обробляти запити що починаються з вказаної стрічки “/group”. Метод *createGroup* позначений анотацією *@PostMapping(“/create”)*, яка вказує що цей метод буде відповідальним за запити, надіслані методом HTTP POST, по відносному шляху “/group/create”. Повний код вказаного класу можна знайти в додатку В.

## 2.5. Безпека додатку

### *2.5.1. Налаштування авторизації*

Для забезпечення безпеки додатку на серверній частині використовувався модуль Spring Security, конфігурація відбувалась за допомогою власноруч написаного класу, зареєстрованого засобами фреймворку Spring. Для авторизації застосовувалась базова HTTP конфігурація з зберіганням згенерованого токену в файлах кукі, з їх видаленням при виході з додатку та з налаштованою CORS (англ. *Cross-Origin Resource Sharing* – спільне використання ресурсів різних джерел, скорочено CORS).

До частини функціоналу додатку було відкрито доступ без авторизації, як наприклад, для функціоналу реєстрації, до решти доступ було надано тільки авторизованим користувачам. В разі якщо неавторизований користувач спробує дістатись функціоналу для нього непризначеного, його запит буде перенаправлено на метод що поверне код 401 (базовий код неавторизованого користувача в протоколі HTTP). В базовому варіанті HTTP авторизації в такій ситуації користувача було б перенаправлено на HTML сторінку або іншу URL адресу, але так як серверна частина не має в собі нічого, що відповідає за HTML сторінки і не повинна їх повертати користувачу, а перенаправлення на іншу URL адресу може спричинити конфлікти через CORS конфігурацію, то було вирішено повертати власне звичайну відповідь від сервера, проте з спеціальним кодом 401.

В CORS налаштуваннях було дозволено лише такі HTTP методи як GET (отримати дані), POST (додати дані), PUT (оновити дані), DELETE (видалити дані) та OPTIONS (був доданий здебільшого для підтримки функціоналу браузера Google Chrome, який використовує його для отримання і перевірки CORS налаштувань). Відповідно в конфігурації контролерів функціонал був розділений між цими різними методами. Було вказано також список доступних HTTP заголовків, якими було визнано: *Access-Control-Allow-Origin, Access-Control-Allow-Credentials, Access-Control-Allow-Headers, Authorization, Content-Type, Accept. WithCredentials* та *Set-Cookie*. Якщо на серверну частину буде надіслано запит, який буде мати заголовок або декілька заголовків не з вказаного списку, то такий запит буде автоматично відхилено в цілях безпеки. В якості джерела запитів (яке саме задається заголовком *Access-Control-Allow-Origin* було вказано тільки URL адресу розробленої клієнтської частини, в разі появи іншого клієнта для серверної частини необхідно буде тільки додати його адресу до цієї конфігурації. Для уявлення про повну конфігурацію безпеки див. додаток Д.

### *2.5.2. Шифрування паролів*

Іншим засобом захисту додатку є шифрування паролів. Навіть, якщо зловмисникам вдасться певним чином дістати їх з бази даних, використати їх буде неможливо через те, що зберігаються вони в зашифрованому вигляді. Власне для цього застосовувалась функція *bcrypt* (адаптивна криптографічна функція, що застосовується для шифрування різних ключів. Функція заснована на шифрі Blowfish, найперше представленим у 1999 році, який на відміну від багатьох інших алгоритмів має досить складну фазу підготовки ключа), представлена в класі *BCryptPasswordEncoder*, який постачається фреймворком Spring. Ще однією особливістю цього підходу є те, що шифратор не містить зворотної логіки, тобто при перевірці паролю при вході користувачем відбувається шифрування введеного паролю і отримана стрічка порівнюється з збереженою в системі. Таким чином навіть отримавши доступ до бібліотеки, що забезпечує шифрування, ви не зможете використати для відновлення збережених паролів користувачів.

### *2.5.3. Налаштування зберігання кукі файлів*

Однією з проблем, з якими довелось стикнутись при налаштуванні безпеки додатку стала політика перевірки файлів кукі першої та третьої сторін, уведена більшістю браузерів в 2019-20му роках. Для кожного файлу кукі повинен бути встановлений атрибут *SameSite*, який може набувати значень *Lax* (кукі файли не визначаються для доступу третім сторонам і надсилаються тільки коли користувач переходить на самий сайт за посиланням; цей тип встановлюється за замовчування для багатьох браузерів), *Strict* (кукі файли визначаються для доступу тільки першим сторонам і не доступні при крос-доменному обміні) та *None* (кукі файли будуть відправлені у всіх можливих контекстах, тобто будуть доступні третім сторонам).

Враховуючи вище описане, якщо клієнтська та серверні частини будуть мати різні доменні імена, щоб забезпечити вільну передачу файлів кукі

(нагадаємо що використана базова HTTP авторизація зберігає дані про авторизованого користувача саме в них) між ними, доведеться використовувати метод *SameSite=None*, щоб кукі файли з серверної частини не були відхилені браузером користувача. Це не стало би проблемою, якби більшість сучасних браузерів, таких як Google Chrome, Edge, Opera тощо не використовували за замовчуванням тип *Lax*, мало не єдиним винятком є браузер Internet Explorer, який по замовчуванню досі дозволяє крос-доменний обмін кукі-файлами.

Отож необхідно було дозволити доступ до кукі файлів третім сторонам при їх створенні, яке відбувається засобами Spring Security. Неприємним моментом стало те, що навіть останні версії цього модуля не підтримують встановлення атрибуту *SameSite* із значенням *None* для забезпечення крос-доменного обміну кукі файлами. Тому необхідно було встановити його власноруч. Найпершим рішенням було створити HTTP фільтр для запитів на авторизацію, який би встановлював потрібний атрибут необхідному файлу. Проте цьому завадило те, що власне засоби Spring Security одразу після його встановлення (через заголовок *Set-Cookie*) блокують стан відповіді сервера і внести зміни до заголовків, тіла відповіді тощо є неможливим (див. Рис. 2.5). Очевидно, що це зроблено для забезпечення безпеки заголовків, в яких зберігається інформація авторизованого користувача.

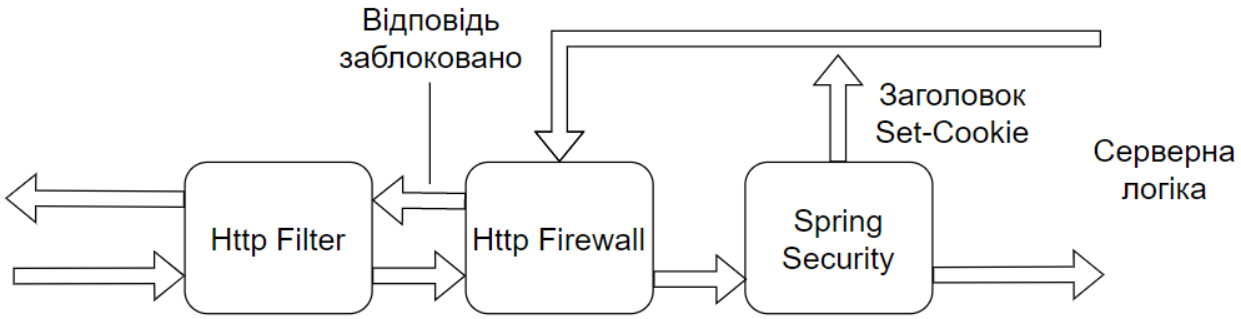


Рис. 2.5 – Діаграма проходження успішно авторизованого запиту

Враховуючи це, єдиним виходом додати необхідні атрибути було модифікувати заголовки, відповідальні за кукі файли перед тим як об’єкт відповіді сервера перейде в заблокований стан. Для цього необхідно було

зареєструвати власний HTTP фаєрвол на серверній частині. На Рис. 2.5 показано діаграму проходження запиту через фільтр та фаєрвол. Засобами фреймворку Spring це можна зробити шляхом імплементації інтерфейсу *HttpFirewall* та реєстрації створеного класу фаєрволу як компонента Spring за допомогою анотації *@Component*. Далі можна перевизначити методи, котрі повертають обгортки об'єктів запиту та відповіді сервера і виконувати необхідні модифікації з заголовками запиту під час отримання або зміни HTTP сесії.

Отож, встановивши атрибут для забезпечення крос-доменного обміну створеному кукі файлу (також необхідно було встановити атрибут *Secure*, який вказує, що поточний кукі файл є безпечним, що є потрібним при використанні атрибуту *SameSite=None*) ми змогли забезпечити його зберігання в браузері в межах домену клієнтської частини для використання в межах домену серверної частини.

## 2.6. Клієнтська частина

### *2.6.1. Загальна конфігурація. Компоненти*

Як вже було вказано вище, для написання клієнтської частини використовувалась мова програмування TypeScript (типізована версія мови програмування JavaScript) та фреймворк Angular 8 (для перегляду діаграми зв'язку компонентів див. Рис. 2.1. Структура проекту складається з власне коду додатку, поділеного на різні компоненти та конфігураційних файлів.

Кожен створений компонент складається з чотирьох файлів: файлу з TypeScript кодом, в якому зберігається вся логіка поточного компоненту, файлу з TypeScript кодом, в якому зберігається конфігурація для запуску тестування компоненту (зазвичай залишається згенерованою за замовчуванням), HTML файлу, в якому зберігається власне код веб-сторінки, та CSS файлу, в якому зберігаються таблиці стилів, підключені в попередньому файлі. Всі створені компоненти (зазвичай один компонент відповідав одній сторінці, за виключенням сторінок, що відповідають за створення та редагування опитувань

та груп, такі сторінки мають спільний компонент з деякими розгалуженнями логіки; також в окремий компонент винесена, наприклад, конфігурація хедеру додатку) підключені в основному компоненті, який власне підключений за допомогою тегу на основній сторінці *index.html*. Таким чином при переадресації на адресу додатку за замовчуванням, буде підключений основний компонент, який визначить на який під-компонент здійснити переадресацію.

Розглянемо детальніше конфігурацію одного компонента. В наступному коді вказано сигнатуру компонента *CreateGroupComponent*, відповідального за сторінки створення та редагування груп користувачів.

```
@Component({
  selector: 'app-create-group',
  templateUrl: './create-group.component.html',
  styleUrls: ['./create-group.component.css']
})
export class CreateGroupComponent implements OnInit {

  group: Group;
  username: string;
  availableUserNames: string[] = [];
  memberNameInputValue: string;

  constructor(private http: HttpClient, private router: Router, private route: ActivatedRoute) {
  }
```

Як можна побачити, оголошений клас імплементує інтерфейс *OnInit*, з якого необхідно перевизначити метод *ngOnInit*, який викликається при ініціалізації компонента, оскільки це вимагається для будь-якого компонента. Ініціалізацію об’єктів, пов’язаних з запитами на сервер, або іншими підключеними сервісами слід проводити саме в цьому методі. Створений клас позначений анотацією *@Component*, яка визначає його як компонент фреймворку Angular (дуже схожа ситуація з конфігурацією компонентів у фреймворку Spring). В об’єкті, що передається як параметр цієї анотації вказано посилання на HTML та CSS файли, що мають бути підключеними до цього компоненту. Вони можуть зберігатись і в іншій директорії, не обов’язково в тій же що й сам компонент, тільки потрібно вказати коректний шлях.

Конструктор створеного класу приймає об’єкти класів *HttpClient* (використовується для виконання HTTP запитів на серверну частину), *Router* (використовується для переадресації на інші компоненти) та *ActivatedRoute* (використовується для отримання поточного URL шляху та його параметрів). Для перегляду коду вказаного компоненту див. додаток Е.

### *2.6.2. Виконання запитів*

Розглянемо детальніше виконання HTTP запитів з клієнтської частини на серверну.

```
this.http.post<GenericResponse>(SERVER_URL + '/group/create', this.group,
HTTP_OPTIONS).toPromise()
  .then(data => {
    if (data.success) {
      this.router.navigate(['/viewGroup/' + data.result.id]);
    }
  })
  .catch(error => {
    if (error.status === 401) {
      this.router.navigate(['/login', {originUrl: '/createGroup'}]);
    }
  });
```

В коді вище вказано приклад виконання запиту на серверну частину з описаного вище компоненту. Як можна бачити, запит виконується за допомогою об’єкту класу *HttpClient* (також описано вище). В даному випадку показано запит на створення групи, вказано URL адресу серверу та відносну адресу методу контролера, передається об’єкт групи та об’єкт з HTTP конфігурацією. Як можна побачити, в разі успішного виконання запиту, користувача буде переадресовано на сторінку перегляду новоствореної групи з використанням ідентифікатора, що був надісланий сервером, а якщо сталась помилка, пов’язана з авторизацією користувача, його буде переадресовано на сторінку входу в додаток.

Для успішного виконання запитів між різними доменами (між клієнтською частиною та серверною) необхідно було правильно налаштувати CORS політику для обох сторін. Детальніше про ці налаштування на серверній частині див. пункт 2.5.1. З клієнтської сторони для правильного роботи цієї системи

необхідно було вказати наступні заголовки *Access-Control-Allow-Origin*, *Access-Control-Allow-Credentials* та *Access-Control-Allow-Headers*. Значенням першого було визначено адресу самої клієнтської частини, другого позитивне булеве значення, а третього – список дозволених заголовків для запитів на клієнтську частину – заголовки *Content-Type*, *WithCredentials* та *Set-Cookie*. Ці заголовки необхідно було передавати з усіма запитами на серверну частину, для цього їх було винесено в спеціальний об’єкт *HTTP_OPTIONS* для доступності з будь-якої точки додатку.

### *2.6.3. Переадресація*

Переадресація URL адрес на компоненти в межах клієнтської частини відбувається за допомогою спеціального модуля, в якому вказано до яких адрес прив’язувати які компоненти. Наприклад:

```
const routes: Routes = [
  ...
  {path: 'viewGroup/:groupId', component: ViewGroupComponent},
  {path: 'createGroup', component: CreateGroupComponent},
  {path: 'editGroup/:groupId', component: CreateGroupComponent},
  ...
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

В даному коді визначено модуль переадресації *AppRoutingModule*, якому задано список об’єктів, що мають два поля: відносний до адреси хосту клієнтської частини шлях та компонент, який буде йому відповідати. За необхідності вказується параметри, значення яких можна буде використати в компоненті. Так, на даному прикладі, шляху *editGroup/4* буде відповідати компонент *CreateGroupComponent*, розглянутий вище, а по ключу *groupId* буде записано значення *4*. Для успішного виконання переадресацій в межах основного

компоненту необхідно імпортувати його в конфігурації останнього, а також додати тег <*router-outlet*> на HTML сторінці основного компонента.

Слід зазначити, що переадресація URL адреси на певний компонент фреймворку Angular працює і в разі прямого переходу на цю адресу з браузера так і під час переадресації з іншого компонента. Остання виконується за допомогою об’єкту класу *Router*, який можна підключити в необхідному компоненті.

# Розділ 3. Інструкція користувача та опис додатку

## 3.1. Реєстрація та авторизація

Для зручності користування додатком процес авторизації та реєстрації було зроблено максимально простим. Проте з цілей безпеки додатку процес реєстрації вимагає підтвердження адреси електронної пошти. На Рис. 3.1 відображено вікно входу в додаток. Форма входу містить перевірку чи поля для введення логіну / паролю не є пустими. В якості логіну можна використовувати електронну адресу, вказану при реєстрації та ім’я користувача (власне логін) на сайті.

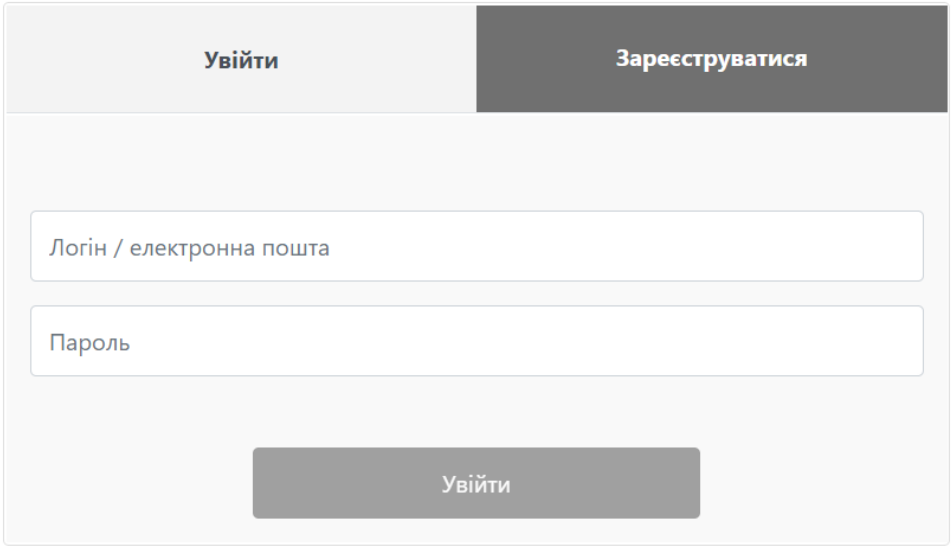


Рис. 3.1 – Вікно входу у веб-додаток

На Рис. 3.2 показано вікно реєстрації у створеному додатку. Достатньо заповнити форму з п’яти полів: повне ім’я для відображення на сайті, актуальну адресу електронної пошти, логін на сайті, та пароль із підтвердженням. Дана форма також містить перевірку чи всі поля є заповненими, а також чи пароль було успішно підтверджено. Слід зазначити, що логін користувача на сайті повинен бути унікальним, якщо користувач спробує зареєструватися з вже існуючим логіном, то отримає помилку.

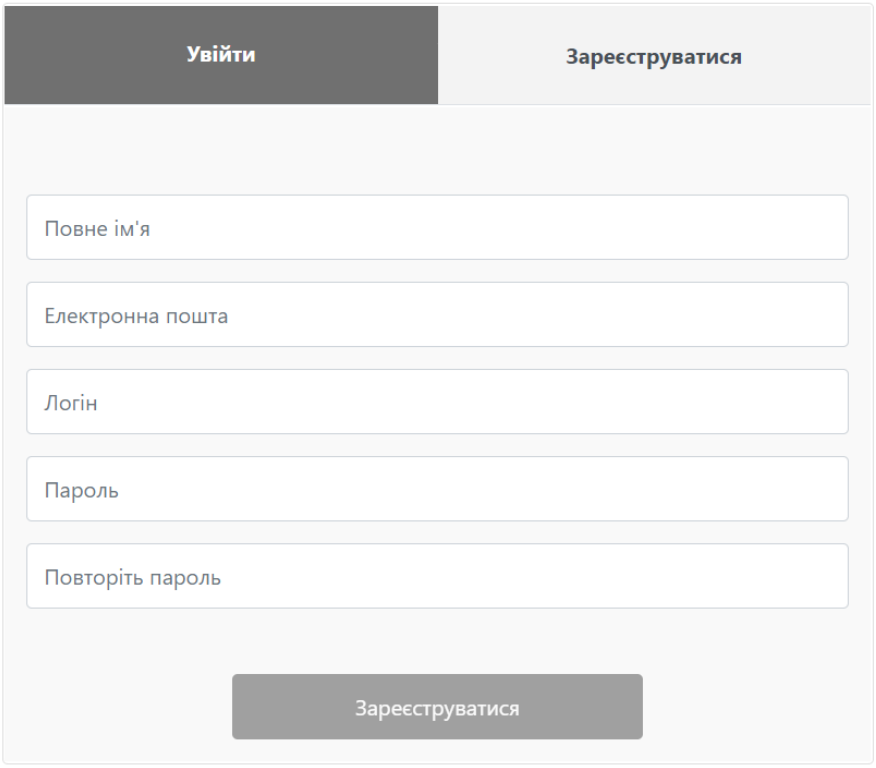


Рис. 3.2 – Вікно реєстрації в додатку

Як вже було зазначено вище, після реєстрації на вказану адресу електронної пошти прийде повідомлення для її підтвердження. В листі буде вказано посилання на веб-додаток з унікальною згенерованою стрічкою, перейшовши за яким, користувач потрапить на сторінку додатку де буде вказано, чи вдалось підтвердити адресу електронної пошти. Після успішної перевірки користувач зможе успішно авторизуватися в додатку.

Перейти на сторінку входу / реєстрації можна за допомогою посилання в правому верхньому куті додатку, в разі якщо користувач перейде за посиланням, яке вимагає авторизації, його буде автоматично перенаправлено на цю сторінку, а після успішного входу - назад до сторінки з якої відбулась переадресація.

### 3.2. Можливості користувача

Якщо в додатку передбачена авторизація (як в даному), виникає дві ситуації: якщо користувач авторизований і, відповідно, якщо ні. Очевидно, що неавторизований користувач буде мати менше можливостей в додатку, оскільки

значна кількість функціоналу додатку для проведення опитувань прив’язана до конкретних користувачів і передбачає авторизацію. Для неавторизованого користувача передбачено наступний короткий набір можливостей:

- реєстрація в додатку;
- перегляд списку опитувань, доступних всім користувачам;
- перегляд конкретного опитування, якщо воно доступне всім користувачам.

В порівнянні, авторизований користувач має наступний набір можливостей:

- вхід / вихід з додатку (авторизація / закінчення сесії);
- редагування даних профілю;
- створення та редагування опитування;
- створення та редагування групи користувачів;
- голосування в доступних опитуваннях;
- перегляд інформації про списки тих хто проголосував, якщо цей дозволено рівнем анонімності опитування;
- перегляд списку створених опитувань, списку груп до яких належить користувач тощо.

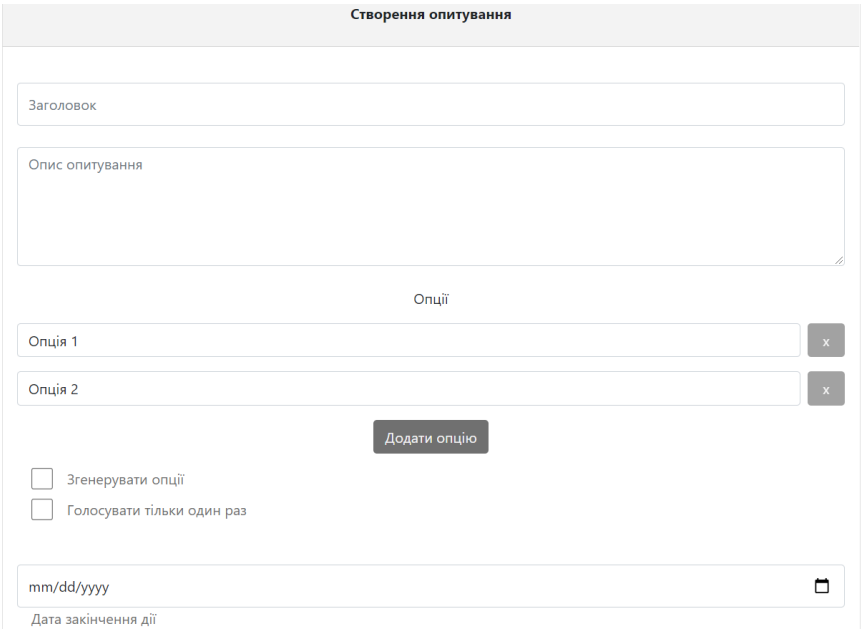


Рис. 3.3 – Частина форми створення опитування

На Рис. 3.3 зображено верхню частину форми для створення опитування. Для створення опитування обов’язково необхідно вказати його заголовок, дату закінчення дії та мінімум дві доступні опції. Слід зазначити що користувач може додавати будь-яку кількість опцій, проте видалити може будь-які крім останніх двох. Користувачу доступна генерація доступних числових варіантів із зазначенням проміжку та кроку. Також доступна можливість вибору опції тільки один раз, в такому випадку при її виборі користувачем вона стане недоступною для інших.

В графі “Доступ до опитування” можна вибрати одну або декілька груп в яких буде розміщено поточне опитування. В такому разі опитування буде доступне тільки користувачам, що є членами цих груп. Також можна встановити рівень анонімності опитування, чи буде воно анонімним для власника та тих хто проголовував, чи тих кому доступне.

Для створення групи достатньо вказати її назву та опис. Користувач що її створює буде автоматично включений як член цієї групи та її адміністратор, також він має змогу додати інших користувачів за їхніми логінами. Після створення групи стане доступним посилання для приєднання на сторінці перегляду інформації про групу. Це дозволяє не додавати кожного користувача до групи, а поширити посилання між ними, отож будь-який зареєстрований користувач із цим посиланням зможе доєднатися до поточної групи. Посилання на створення опитування та групи доступні для користувача на головній сторінці додатку.

На Рис. 3.4 зображено форму перегляду опитування з відповідної сторінки. На сторінці вказано назву та опис опитування, а також список доступних для голосування опцій. Якщо користувач не проголосував в цьому опитуванні, опції будуть доступні для вибору, якщо проголосував – то ні, а натомість будуть відображені результати головування за кожну опцію у відсотках. Якщо ж опція стає недоступною після вибору одним користувачем, то замість відсотків буде відображатись назва опції.

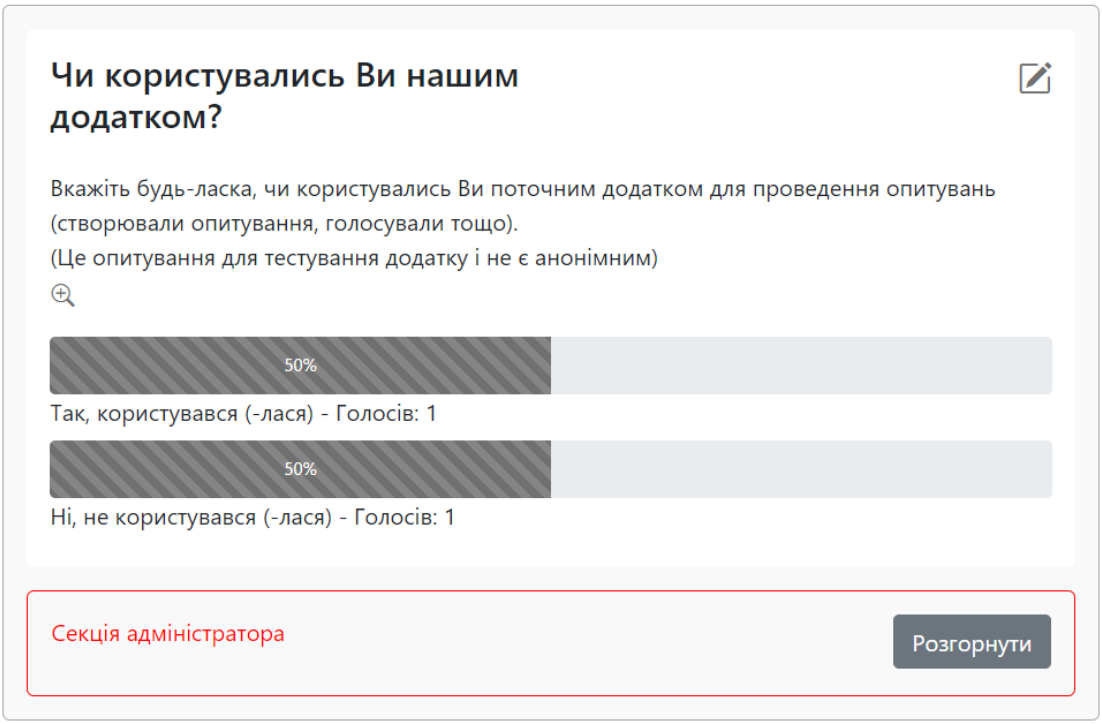


Рис. 3.4 – Форма перегляду опитування

Зліва від форми перегляду опитування буде доступним список користувачів що проголосували за кожну опцію, якщо результати опитування не є анонімними для поточного користувача. По замовчуванню для кожної опції відображається лише три перших користувача, що за неї проголосували, проте список для кожної опції можна розгорнути в разі необхідності. В правому верхньому куті форми перегляду опитування буде відображено значок посилання переходу на сторінку редагування опитування, якщо поточний користувач є його власником. Під доступними опціями / результатами опитування розміщено секцію адміністратора, яка доступна тільки власнику опитування. Наразі ця секція дозволяє проголосувати за певну опцію від імені іншого користувача, якому доступне поточне опитування ( це може бути потрібно наприклад, якщо викладач хоче вибрати варіант лаборантові роботи для певних студентів наперед, а решті дати можливість вибрати варіанти самостійно тощо). Слід зазначити, що секція буде недоступна, якщо опитування доступне всім користувачам в додатку, а не для конкретної групи або груп.

Отож бачимо, що можливості користувача широко різняться в залежності від того, чи є він авторизованим в додатку, чи є власником певного опитування або групи тощо.

### 3.3. Мобільна версія додатку

Слід зазначити, що створеним веб-додатком можна користуватись як з комп'ютерів, так і з мобільних пристроїв. Для цього було створено спеціальні мобільні версії веб-сторінок додатку або їх частин, щоб при користуванні додатком на мобільному пристрої в користувача не виникло труднощів, пов'язаних з малим розміром тексту, некоректним розміщенням елементів, масштабуванням тощо (див. Рис. 3.5). В мобільній версії додатку доступні всі ті ж можливості як і в звичайній (вхід, реєстрація, функціонал створення, редагування та проведення опитування, створення та редагування груп тощо).

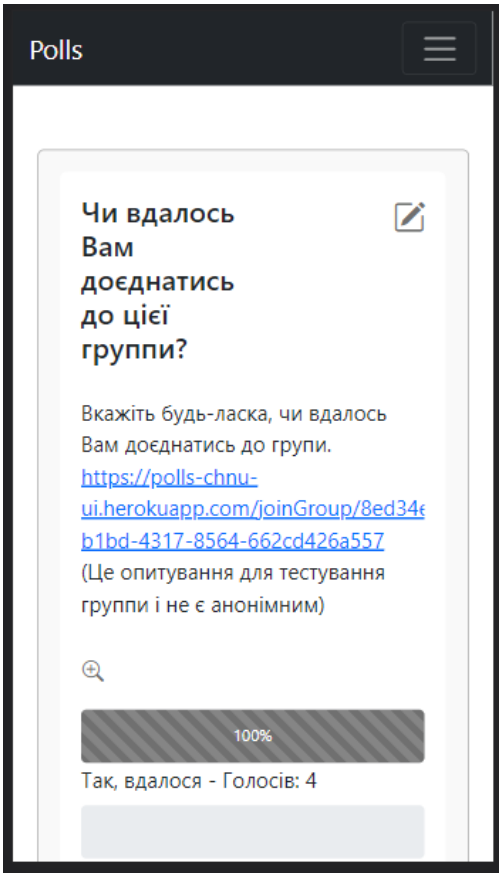


Рис. 3.5 – Сторінка перегляду опитування в мобільній версії додатку на прикладі розширення пристрою Samsung Galaxy S5

### 3.4. Практичне застосування додатку

Створений веб-додаток може застосовуватись в навчальному процесі на факультеті, а також використовуватись студентським парламентом.

Надалі описано практичні особливості, які відрізняють додаток від існуючих аналогів:

- динамічність доступності опитування (опитування можна легко редагувати, вказавши для доступу інші групи, або зробити його загальнодоступним);
- можливість додавати необмежену кількість варіантів, доступних для голосування;
- можливість генерації варіантів, доступних для голосування (підходить наприклад для вибору варіантів лабораторних робіт тощо, див Рис. 3.6);
- додаток є одним спільним місцем для будь-яких опитувань (не важливо чи потрібно дізнатись думку студентів чи вибрати варіанти лабораторних робіт, корегуючи параметри при створенні опитування можна забезпечити практично всі вимоги);
- можливість власнику опитування вибирати опції для учасників опитування (наприклад як в ситуації описаній в пункті 3.2).

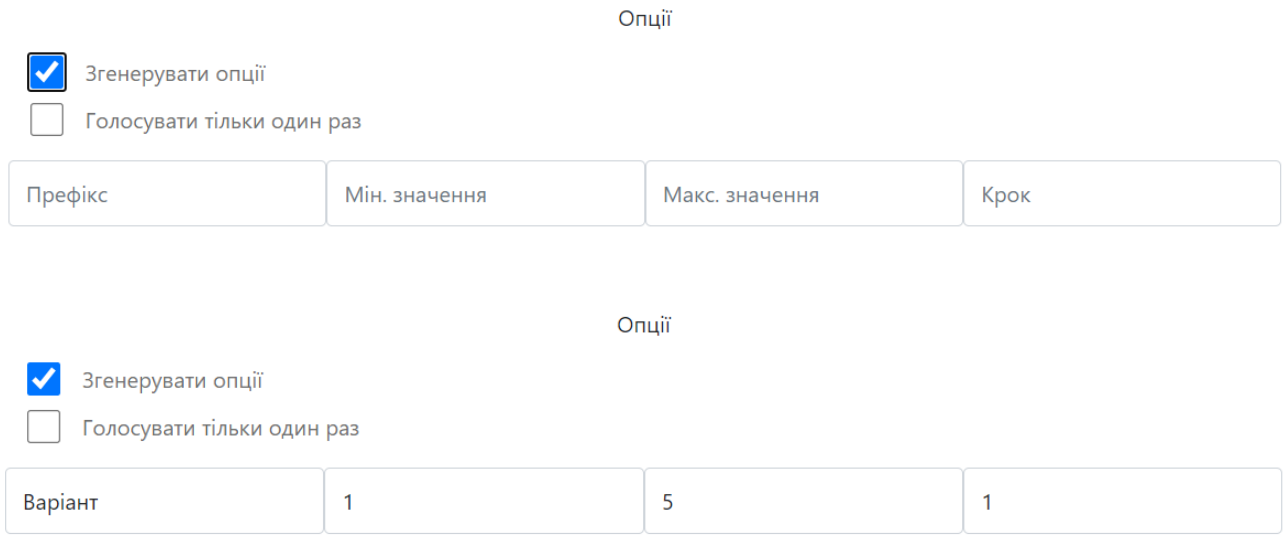


Рис. 3.6 – Форма з генерацією доступних опцій для опитування в незаповненому та заповненому вигляді

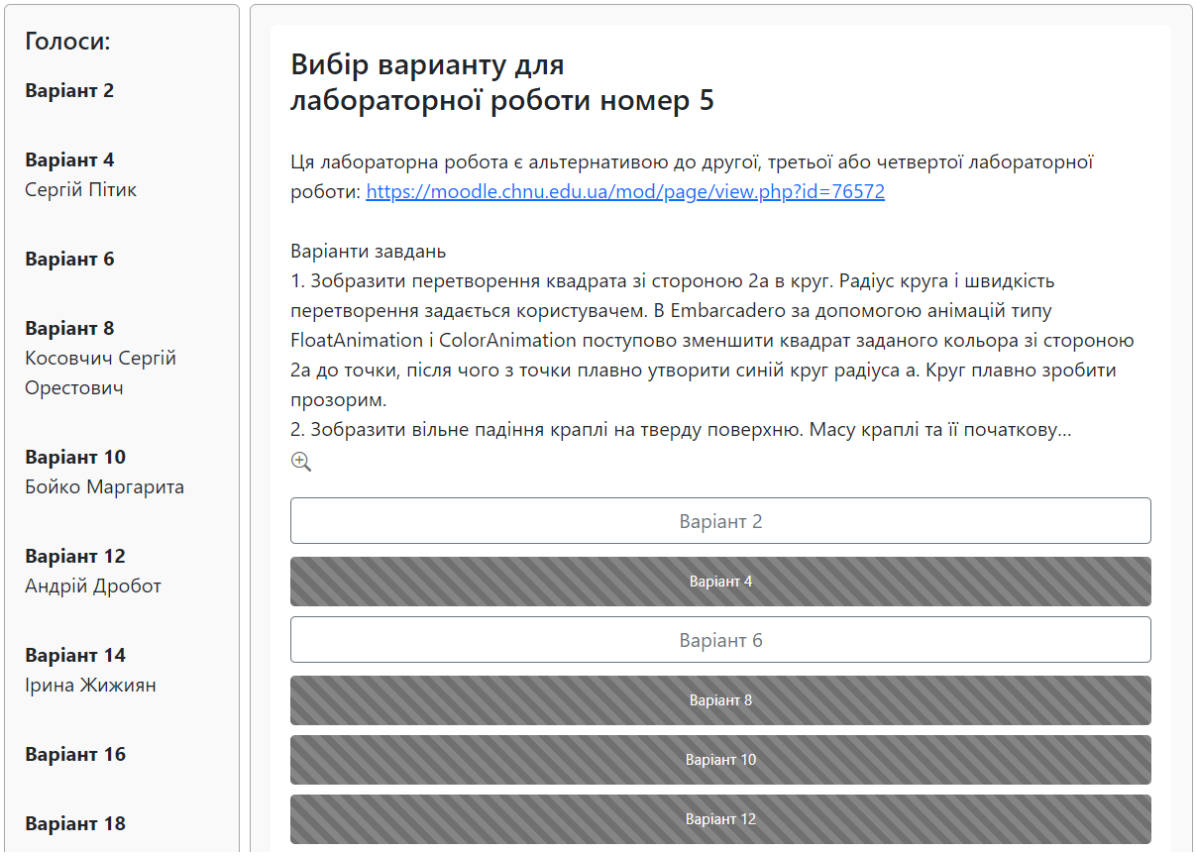


Рис. 3.7 – Зображення опитування, використаного для вибору варіантів лабораторної роботи студентами п’ятого курсу

Можливості використання додатку на факультеті є наступними:

- вибір студентами варіантів лабораторних робіт (набуло практичного застосування, див. Рис. 3.7);
- вибір студентами тем курсових робіт (див. Рис. 3.8);
- вибір студентами предметів за вибором (планується реалізація в наступному семестрі);
- проведення опитувань серед студентів різних груп (наприклад під час пандемії проводяться опитування про рівень вакцинації / захворюваності серед студентів тощо).

З описаного вище, додаток може набути досить широкого спектру застосування в межах факультету, крім того існує можливість його подальшого розвитку, наприклад, для створення глобальної мережі для проведення опитувань. Розроблений веб-додаток розміщено за посиланням [10].

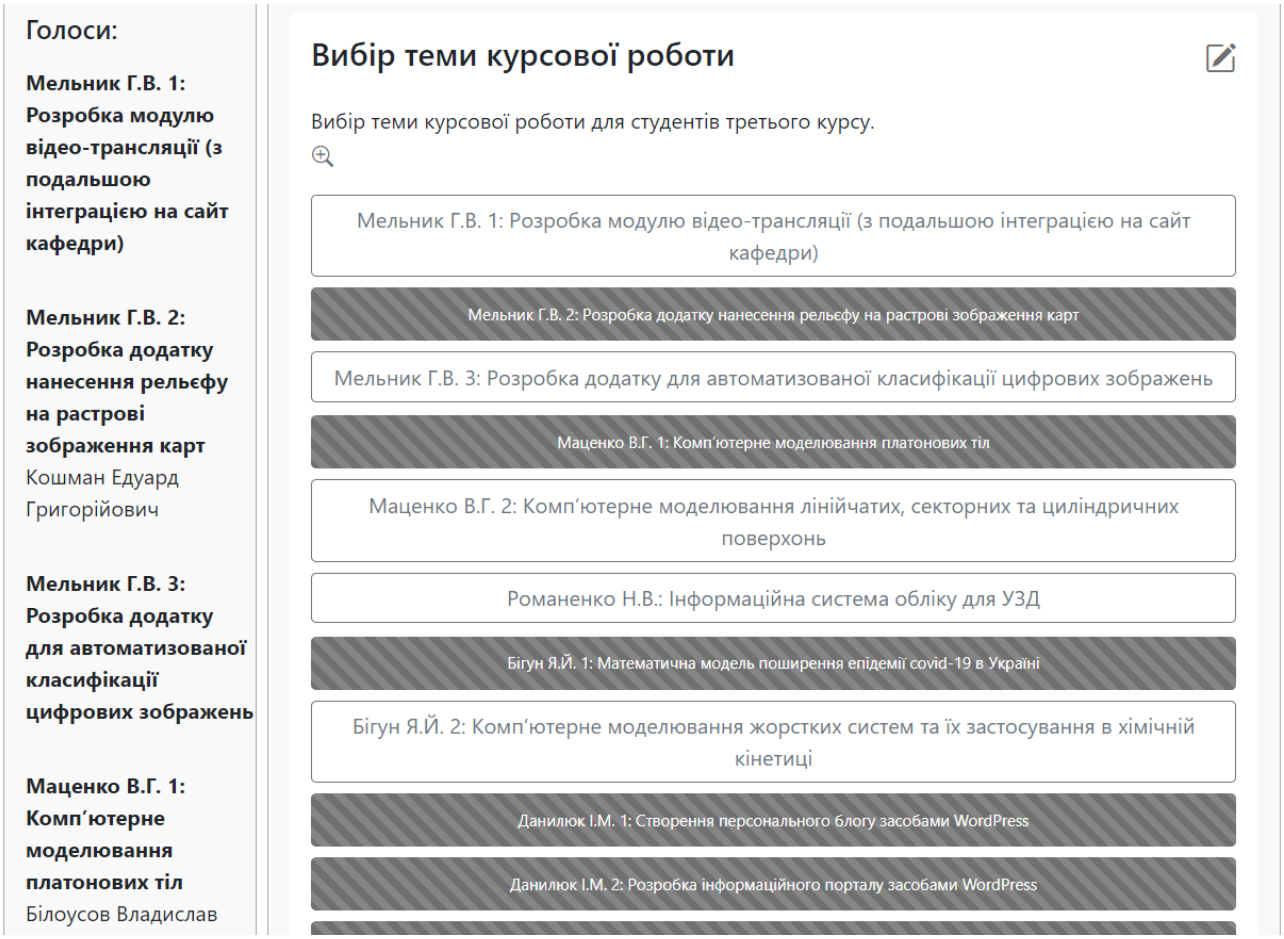


Рис. 3.8 – Зображення опитування для вибору тем курсових робіт

# Висновки

Внаслідок виконання дипломної роботи “Створення веб-додатку для проведення голосувань та опитувань на факультеті” отримано наступні результати:

- розглянуто сучасні веб-системи та методи їх застосування;
- здійснено проектування архітектури та вибір технології для розробки веб-додатку для проведення голосувань та опитувань;
- здійснено теоретичний та практичний огляд вибраних для розробки технологій;
- закріплено засвоєні теоретичні та практичні знання пов’язані з розробками веб-систем;
- здійснено програмну реалізацію веб-додатку для проведення голосувань та опитувань для застосування в навчальному процесі в межах факультету;
- описано методи розв’язання проблем, які виникли в ході програмної реалізації додатку.

# Перелік посилань


1. Хорстманн К., Корнел Г. – Java. Библиотека профессионала. Том 1. Основы. – Москва * Санкт-Петербург * Киев 2016
2. Bruce Eckel, Svetlana Isakova – Atomic Kotlin – USA * Germany – MindView LLC 2021
3. Онлайн курс Jetbrains – Kotlin for Java Developers [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/kotlin-for-java-developers/home/welcome
4. Офіційна документація Kotlin [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/docs/home.html
5. Офіційна документація фреймворку Spring [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-framework/docs/current/reference/html/
6. Офіційна документація Typescript [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/
7. Офіційна документація фреймворку Angular [Електронний ресурс]. – Режим доступу: https://angular.io/tutorial
8. Проект серверної частини на GitHub [Електронний ресурс]. – Режим доступу: https://github.com/MikeMalyar/polls
9. Проект клієнтської частини на GitHub [Електронний ресурс]. – Режим доступу: https://github.com/MikeMalyar/polls-ui
10. Адреса веб-додатку [Електронний ресурс]. – Режим доступу: https://polls-chnu-ui.herokuapp.com/

# Додатки

## Додаток А

(Файл **GroupRepository.kt**, код інтерфейсу **GroupRepository)**

```
package com.chnu.polls.repository.personal

import com.chnu.polls.domain.personal.Group
import org.springframework.data.jpa.repository.JpaRepository
import org.springframework.data.jpa.repository.Query
import org.springframework.stereotype.Repository

/**
 * Інтерфейс, призначений для роботи з сутністю Group на рівні бази даних
 */
@Repository
interface GroupRepository: JpaRepository<Group, Long> {

    /**
     * Метод, призначений для одержання групи за її назвою
     */
    fun findByTitle(title: String): Group?

    /**
     * Метод, призначений для одержання назв доступних в додатку груп
     */
    @Query(value = "SELECT DISTINCT title FROM groups", nativeQuery = true)
    fun getAvailableGroupNames(): List<String>

    /**
     * Метод, призначений для одержання групи за її унікальним токеном
     */
    fun getByAccessToken(accessToken: String): Group?
}
```

# Додаток Б

(Файл **GroupService.kt**, код класу **GroupService**)

```
package com.chnu.polls.service

import com.chnu.polls.domain.personal.Group
import com.chnu.polls.domain.personal.UserGroup
import com.chnu.polls.domain.personal.UserGroupRole
import com.chnu.polls.dto.GroupDto
import com.chnu.polls.dto.MyGroupDto
import com.chnu.polls.repository.activity.PollRepository
import com.chnu.polls.repository.personal.GroupRepository
import com.chnu.polls.repository.personal.UserGroupRepository
import com.chnu.polls.repository.personal.UserGroupRoleRepository
import com.chnu.polls.repository.personal.UserRepository
import com.chnu.polls.util.getLoggedUser
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.stereotype.Service

/**
 * Сервісний клас, призначений для зберігання логіки,
 * пов'язної з групами користувачів в додатку
 */
@Service
class GroupService (
        @Autowired val groupRepository: GroupRepository,
        @Autowired val userGroupRepository: UserGroupRepository,
        @Autowired val userGroupRoleRepository: UserGroupRoleRepository,
        @Autowired val userRepository: UserRepository,
        @Autowired val pollRepository: PollRepository
) {

    /**
     * Метод, призначений для створення групи користувачів
     */
    fun createGroup(groupDto: GroupDto): GroupDto {

        // перевірка чи існує група з такою назвою
        if (groupRepository.findByTitle(groupDto.title) != null) {
            throw IllegalArgumentException("Group title ${groupDto.title} is already occupied")
        }

        var group = GroupDto.toGroup(groupDto)

        // збереження створеного об'єкту
        group = groupRepository.save(group)
        group = groupRepository.getOne(group.getId())

        val adminRole = userGroupRoleRepository.getOne(UserGroupRole.Roles.ADMIN.toString())

        val user = getLoggedUser().orElseThrow { throw IllegalStateException("User is not logged in") }

        // прив'язуємо поточного користувача як адміністратора групи
        val userGroup = UserGroup().setUser(user)
                .setGroup(group)
                .setRole(adminRole)
        userGroupRepository.save(userGroup)
```

```
    return GroupDto.fromGroup(group)
}

/**
 * Метод, призначений для отримання групи по унікальному ідентифікатору
 */
fun getGroup(id: Long): GroupDto? {

    // отримуємо групу користувачів
    val group = groupRepository.findOne(id)

    return mapGroupToDto(group)
}

/**
 * Метод, призначений для отримання групи за назвою
 */
fun getGroup(title: String): GroupDto? {

    // отримуємо групу користувачів
    val group = groupRepository.findByTitle(title)

    return mapGroupToDto(group)
}

/**
 * Метод, призначений для оновлення інформації групи
 */
fun updateGroup(groupDto: GroupDto): GroupDto? {

    // отримуємо групу користувачів
    var group = groupRepository.findOne(groupDto.id)

    if (group != null) {

        // оновлюємо дані групи
        group.setTitle(groupDto.title)
                .setDescription(groupDto.description)

        // отримуємо користувачів, що входять до групи
        val members = userGroupRepository.findAllByGroup(group).toMutableList()
        groupDto.memberNames.forEach { memberName ->

            val user = userRepository.getByUsername(memberName)
            if (user != null) {

                // якщо такий користувач не належить групі, додаємо його
                if (members.none { userGroup -> userGroup.getUser().username == memberName }) {

                    val userRole = userGroupRoleRepository.getOne(UserGroupRole.Roles.USER.toString())

                    val userGroup = UserGroup().setUser(user)
                            .setGroup(group)
                            .setRole(userRole)
                    userGroupRepository.save(userGroup)
                } else {    // інакше - не потрібно робити ніяких змін
                    members.removeIf { userGroup -> userGroup.getUser().username == memberName }
                }
            }
        }

        // видаляємо користувачів що залишились в групі
        members.forEach { userGroup ->
```

```
            userGroupRepository.delete(userGroup)
        }

        // зберігаємо групу
        group = groupRepository.save(group)

        return GroupDto.fromGroup(group)
    }
    return null
}

/**
 * Метод, що повертає список доступних груп
 */
fun getAvailableGroupNames(): List<String> {

    return groupRepository.getAvailableGroupNames()
}

/**
 * Метод, що повертає список груп авторизованого користувача з кількістю нового контенту
 */
fun getLoggedUserGroupsList(page: Int, count: Int): List<MyGroupDto> {
    val user = getLoggedUser()

    // якщо користувач авторизований
    return if (user.isPresent) {
        // знаходимо список груп для користувача
        userGroupRepository.findAllByUser(user.get())
                .map { userGroup ->
                    val group = userGroup.getGroup()
                    val result = MyGroupDto.fromGroup(group)
                    // знаходимо кількість нового контенту в групі
                    result.newContent = pollRepository.getPollsAvailableForPersonalEntityIdAndNotReacted(
                            group.getId()!!, user.get().getPersonalEntityId()!!, page, count
                    ).size.toLong()
                    result
                }
    } else {    // інакше повертаємо пустий список
        listOf()
    }
}

/**
 * Метод, призначаний для додавання користувача до групи за унакільним токеном
 */
fun joinGroupByToken(token: String): Boolean {
    val user = getLoggedUser()
    // знаходимо групу за токеном
    val group = groupRepository.getByAccessToken(token)

    // якщо користувач авторизований та група існує (токен коректний)
    return if (user.isPresent && group != null) {
        val userRole = userGroupRoleRepository.getOne(UserGroupRole.Roles.USER.toString())

        val existingGroup = userGroupRepository.findAllByUserAndGroup(user.get(), group)
        if (existingGroup.isNotEmpty()) {
            return true
        }

        // додаємо користувача до групи
        val userGroup = UserGroup().setUser(user.get())
                .setGroup(group)
```

```
                .setRole(userRole)
            userGroupRepository.save(userGroup)

            true
        } else {
            false
        }
    }

    /**
     * Метод для перетворення групи в об'єкт для передачі
     */
    private fun mapGroupToDto(group: Group?): GroupDto? {
        if (group != null) {
            val groupDto = GroupDto.fromGroup(group)

            // знаходимо імена користувачів, що належать групі
            val memberNames = userGroupRepository.findAllByGroup(group)
                .map { userGroup -> userGroup.getUser().username }
            groupDto.memberNames = memberNames
            return groupDto
        }
        return null
    }
}
```

## Додаток В

(Файл **GroupController.kt**, код класу **GroupController**)

```
package com.chnu.polls.controller.personal

import com.chnu.polls.dto.GroupDto
import com.chnu.polls.dto.MyGroupDto
import com.chnu.polls.rest.GenericResponse
import com.chnu.polls.service.GroupService
import org.apache.logging.log4j.LogManager
import org.apache.logging.log4j.Logger
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.web.bind.annotation.*

/**
 * Контроллер, призначений для операцій з групами користувачів
 */
@RestController
@RequestMapping("/group")
class GroupController (
        @Autowired val groupService: GroupService
) {

    // унікальний логер клас для контролера
    private val logger: Logger = LogManager.getLogger(this.javaClass)

    /**
     * Метод POST, призначений для створення групи користувачів
     */
    @PostMapping("/create")
    fun createGroup(@RequestBody groupDto: GroupDto): GenericResponse<GroupDto> {
        return try {
            GenericResponse.of(groupService.createGroup(groupDto))
        } catch (ex: Exception) {
            processException(ex)    // опрацьовуємо виключення
        }
    }

    /**
     * Метод PUT, призначений для оновлення даних групи
     */
    @PutMapping("/update")
    fun updateGroup(@RequestBody groupDto: GroupDto): GenericResponse<GroupDto?> {
        return try {
            GenericResponse.of(groupService.updateGroup(groupDto))
        } catch (ex: Exception) {
            processException(ex)    // опрацьовуємо виключення
        }
    }

    /**
     * Метод GET, призначений для отримання групи по ідентифікатору
     */
    @GetMapping("/get/{id}")
    fun getGroupById(@PathVariable id: Long): GenericResponse<GroupDto?> {
        return try {
            GenericResponse.of(groupService.getGroup(id))
        } catch (ex: Exception) {
```

```
            processException(ex)    // опрацьовуємо виключення
        }
    }

    /**
     * Метод GET, призначений для отримання групи за назвою
     */
    @GetMapping("/getByTitle/{title}")
    fun getGroupByTitle(@PathVariable title: String): GenericResponse<GroupDto?> {
        return try {
            GenericResponse.of(groupService.getGroup(title))
        } catch (ex: Exception) {
            processException(ex)    // опрацьовуємо виключення
        }
    }

    /**
     * Метод GET, прзначений для отримання списку доступних груп
     */
    @GetMapping("/getAvailableGroupNames")
    fun getAvailableGroupNames(): GenericResponse<List<String>> {
        return try {
            GenericResponse.of(groupService.getAvailableGroupNames())
        } catch (ex: Exception) {
            processException(ex)    // опрацьовуємо виключення
        }
    }

    /**
     * Метод GET, призначений для отримання списку груп авторизованого користувача
     */
    @GetMapping("/getLoggedUserGroupsList/{page}/{count}")
    fun getLoggedUserGroupsList(@PathVariable page: Int,
                                @PathVariable count: Int): GenericResponse<List<MyGroupDto>> {
        return try {
            GenericResponse.of(groupService.getLoggedUserGroupsList(page, count))
        } catch (ex: Exception) {
            processException(ex)    // опрацьовуємо виключення
        }
    }

    /**
     * Метод, призначений для обробки помилок
     */
    private fun <T> processException(ex: Exception): GenericResponse<T> {
        logger.error(ex)    // логуємо помилку
        // повертаємо відповідь про помилку
        return GenericResponse.error(if (ex.localizedMessage != null) ex.localizedMessage else "")
    }
}
```

## Додаток Д

(Файл **SecurityConfiguration.kt**, код класу **SecurityConfiguration**)

```
package com.chnu.polls.config

import com.chnu.polls.service.UserService
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.ComponentScan
import org.springframework.context.annotation.Configuration
import org.springframework.security.authentication.AuthenticationManager
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder
import org.springframework.security.config.annotation.web.builders.HttpSecurity
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
import org.springframework.web.cors.CorsConfiguration
import org.springframework.web.cors.CorsConfigurationSource
import org.springframework.web.cors.UrlBasedCorsConfigurationSource

/**
 * Конфігураційний клас, що відповідає за налаштування безпеки додатку
 */
@Configuration
@EnableWebSecurity
@ComponentScan(basePackages = ["com.chnu.polls"])
open class SecurityConfiguration : WebSecurityConfigurerAdapter() {

    /**
     * Доступні HTTP методи
     */
    private val ALLOWED_METHODS = listOf("GET", "POST", "PUT", "DELETE", "OPTIONS")

    /**
     * Доступні HTTP заголовки
     */
    private val ALLOWED_HEADERS = listOf(
            "Access-Control-Allow-Origin",
            "Access-Control-Allow-Credentials",
            "Access-Control-Allow-Headers",
            "Authorization",
            "Content-Type",
            "Accept",
            "withCredentials",
            "Set-Cookie")

    @Autowired
    private lateinit var userService: UserService

    @Bean("authenticationManager")
    @Throws(Exception::class)
    override fun authenticationManagerBean(): AuthenticationManager {
        return super.authenticationManagerBean()
    }

    /**
     * Конфігурація шифрування паролів
     */
```

```
@Bean
open fun passwordEncoder(): BCryptPasswordEncoder {
   return BCryptPasswordEncoder()
}

/**
 * Конфігурація CORS
 */
@Bean
open fun corsConfigurationSource(): CorsConfigurationSource {
   val configuration = CorsConfiguration()
   configuration.allowedOrigins = listOf("https://polls-chnu-ui.herokuapp.com")
   configuration.allowedMethods = ALLOWED_METHODS
   configuration.allowCredentials = true
   configuration.allowedHeaders = ALLOWED_HEADERS
   val source = UrlBasedCorsConfigurationSource()
   source.registerCorsConfiguration("/**", configuration)
   return source
}

@Throws(Exception::class)
override fun configure(http: HttpSecurity) {
   http
         .httpBasic()
         .and()

         // увімкнення CORS політики
         .cors()
         .and()

         // відключення CSRF політики
         .csrf().disable()

         // конфігурація авторизації
         .formLogin()
         .loginPage("/user/login/failed")
         .and()     // перевизначити відповідь в разі не-авторизації

         // конфігурація виходу з додатку
         .logout()
         .logoutSuccessUrl("/user/logout")
         .clearAuthentication(true)
         .invalidateHttpSession(true)
         .deleteCookies("JSESSIONID", "XSRF-TOKEN")
         .and()
         .authorizeRequests()
         // адреси, дозволені неавторизованим користувачам
         .antMatchers("/user/login", "/user/logout", "/user/register", "/user/test",
               "/loggedUserFullName", "/poll/get/*", "/loggedUserName", "/group/get/*",
               "/getPollsAvailableForGroup/*/*/*", "/getPollsAvailableForLoggedUser/*/*",
               "/user/verifyToken/*")
         .permitAll()

         // адреси, дозволені тільки авторизованим користувачам
         .antMatchers("/user/profile", "/poll/create", "/poll/update", "/poll/vote/*",
               "user/getAllUsernames", "/group/create", "/group/update", "/getAvailableGroupNames",
               "/poll/getLoggedUserPolls", "/getLoggedUserPollsList/*/*", "/getLoggedUserGroupsList/*/*",
               "/user/joinGroupByToken/**")
         .authenticated()
}

// реєстрація власного сервісу для операцій з користувачами
@Throws(Exception::class)
```

```
    override fun configure(auth: AuthenticationManagerBuilder) {
        auth.userDetailsService(userService);
    }
}
```

## Додаток Е

(Файл **create-group.component.ts**, код класу **CreateGroupComponent)**

```
import {Component, OnInit} from '@angular/core';
import {Group} from '../models/group';
import {GenericResponse} from '../models/rest';
import {HTTP_OPTIONS, SERVER_URL} from '../config/http-config';
import {HttpClient} from '@angular/common/http';
import {ActivatedRoute, Router} from '@angular/router';
import { v4 as uuid } from 'uuid';

/*
  Компонент, відповіданьний за сторінки створення та редагування групи
 */
@Component({
  selector: 'app-create-group',
  templateUrl: './create-group.component.html',
  styleUrls: ['./create-group.component.css']
})
export class CreateGroupComponent implements OnInit {

  // об'єкт групи
  group: Group;

  // авторизований користувач
  username: string;
  // список доступних користувачів для групи
  availableUserNames: string[] = [];

  memberNameInputValue: string;

  constructor(private http: HttpClient, private router: Router, private route: ActivatedRoute) {
  }

  /*
    Ініціалізація компоненту
   */
  ngOnInit() {

    this.group = new Group();
    this.group.memberNames = [];

    // якщо цей параметр доступний то редагуємо існуючу групу
    const groupId = this.route.snapshot.paramMap.get('groupId');

    this.http.get<GenericResponse>(SERVER_URL + '/user/loggedUserName', HTTP_OPTIONS).toPromise()
      .then(data => {
        if (!data.result) {
          this.router.navigate(['/login', {originUrl: 'createGroup'}]);
        } else {
          this.username = data.result;

          this.http.get<GenericResponse>(SERVER_URL + '/user/getAllUsernames', HTTP_OPTIONS).toPromise()
            .then(users => {
              this.availableUserNames = users.result;
              if (this.availableUserNames.includes(this.username)) {
                this.availableUserNames.splice(this.availableUserNames.indexOf(this.username), 1);
              }
```

```
        });

      // якщо група вже існує
      if (groupId) {
        this.http.get<GenericResponse>(SERVER_URL + '/group/get/' + groupId, HTTP_OPTIONS).toPromise()
          .then(gropdData => {
            if (gropdData.result) {
              // витягаємо групу з відповіді сервера
              this.group = gropdData.result;
              this.processGroupMembers();

              // якщо користувач не має доступу до групи, адресуємо на 404
              if (!this.group.memberNames.includes(this.username)) {
                this.router.navigate(['**']);
              }
            } else {
              // такої групи не існує, переадресація на 404
              this.router.navigate(['**']);
            }
          });
      } else {
        this.processGroupMembers();
      }
    }
  });
}

/*
  Метод для обробки доступних користувачів для групи
 */
processGroupMembers() {
  if (!this.group.memberNames.includes(this.username)) {
    this.group.memberNames.push(this.username);
  }

  // tslint:disable-next-line:prefer-const
  for (let memberName in this.group.memberNames) {
    if (this.availableUserNames.includes(memberName)) {
      this.availableUserNames.splice(this.availableUserNames.indexOf(memberName), 1);
    }
  }
}

/*
  Метод для вибору користувача до групи
 */
addMemberNameOption() {
  const memberNameInputValue = this.memberNameInputValue;
  document.getElementById('availableUserNamesList').childNodes.forEach(child => {
    // @ts-ignore
    if (child.innerText === memberNameInputValue) {
      this.group.memberNames.push(memberNameInputValue);
      this.availableUserNames.splice(this.availableUserNames.indexOf(memberNameInputValue), 1);

      this.memberNameInputValue = '';
    }
  });
}

/*
  Метод для видалення користувача з групи
 */
removeMemberNameOption(i) {
```

```
    const memberName = this.group.memberNames[i];
    if (memberName !== this.username) {
      this.group.memberNames.splice(i, 1);
      this.availableUserNames.push(memberName);
      this.availableUserNames.sort();
    }
  }

  /*
    Метод для створення групи
   */
  createGroup() {
    if (!this.group.accessToken) {
      // створюємо унікальний токен для доступу до групи
      this.group.accessToken = uuid();
    }
    this.http.post<GenericResponse>(SERVER_URL + '/group/create', this.group, HTTP_OPTIONS).toPromise()
      .then(data => {
        if (data.success) {
          this.router.navigate(['/viewGroup/' + data.result.id]);
        }
      })
      .catch(error => {
        if (error.status === 401) {
          // користувач неавторизований, переадресація на сторінку входу
          this.router.navigate(['/login', {originUrl: '/createGroup'}]);
        }
      });
  }

  /*
    Метод для редагування групи
   */
  updateGroup() {
    this.http.put<GenericResponse>(SERVER_URL + '/group/update', this.group, HTTP_OPTIONS).toPromise()
      .then(data => {
        if (data.success) {
          this.router.navigate(['/viewGroup/' + data.result.id]);
        }
      })
      .catch(error => {
        if (error.status === 401) {
          // користувач неавторизований, переадресація на сторінку входу
          this.router.navigate(['/login', {originUrl: '/createGroup'}]);
        }
      });
  }
}
```