math_2023_043

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

Створення рекомендаційної системи закладів харчування за допомогою алгоритмів машинного навчання

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:
студент 6 курсу, 607 групи
Яхненко Станіслав Геннадійович
***Керівник**:*
доцент, канд.фіз.-мат. наук
Горбатенко М.Ю.

До захисту допущено
на засіданні кафедри
протокол № 8 від 5 грудня 2023 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2023

Анотація

Розроблено рекомендаційну систему закладів харчування та веб-додаток для відображення результатів з використанням алгоритмів машинного навчання, .NET, Python, TypeScript, Angular та SQL Server.

Застосунок призначений для використання для рекомендацій ресторанів користувачам.

Ключові слова:

Вебдодаток, машинне навчання, рекомендації, заклади харчування, користувач, авторизація, .NET, ASP.NET, Python, TypeScript, Angular та SQL Server.

Annotation

Developed a recommendation system for restaurants and a web application for displaying the results using machine learning algorithms, .NET, Python, TypeScript, Angular, and SQL Server..

The application is intended to be used to recommend restaurants to users.

Keywords:

Web application, machine learning, recommendations, restaurants, user, authorization, .NET, ASP.NET, Python, TypeScript, Angular, and SQL Server.

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

_____ С.Г. Яхненко

Зміст

Вступ 4
1 Огляд літератури за тематикою кваліфікаційної роботи 6
1.1 Проблематика 6
1.2 Огляд існуючих рекомендаційних додатків 7
2 Огляд технологій розробки веб-додатків та моделей машинного навчання 10
2.1 Сучасні підходи та інструменти для створення застосунків 10
2.2 Сучасні підходи створення моделей машинного навчання 11
2.3 Мова програмування Python 13
2.4 Мова програмування C# 14
2.5 Фреймворк ASP.NET 15
2.6 Мова програмування TypeScript 16
2.7 Фреймворк Angular 17
2.8 СКБД SQL Server 18
3 Теоретичний опис побудови платформи 19
3.1 Створення серверного додатку у вигляді RESTful API 19
3.2 Реалізація серверної логіки з Onion архітектурою 20
3.3 Аспекти інверсії керування та використання Dependency Injection 21
4 Постановка завдання та програмна реалізація 22
4.1 Постановка задачі 22
4.2 Створення датасету та методи збирання даних 23
4.3 Оцінка та вибір відповідного алгоритму машинного навчання 24
4.4 Архітектура системи 29
5 Функціонал додатку 41
5.1 Перший вхід. Реєстрація. Авторизація 41
5.2 Сторінка початкового вибору закладів 42
5.3 Сторінка перегляду рекомендацій 43
Висновки 46
Список використаної літератури 47
Додатки 49

Вступ

Актуальність роботи. В епоху, коли кулінарна індустрія процвітає, а вибір споживачів постійно розширюється, актуальність розробки складних рекомендаційних систем для ресторанів неможливо переоцінити. Різке зростання кількості різноманітних закладів харчування в поєднанні з різноманітними уподобаннями споживачів вимагає технологічно просунутого підходу до супроводу людей у допомозі вибору ресторанів. Ця кваліфікаційна робота заглиблюється в цю тему, маючи на меті використати можливості алгоритмів машинного навчання, щоб революціонізувати процес надання рекомендацій щодо закладів харчування.

Мета. Основною метою цієї дипломної роботи є дослідження та застосування алгоритмів машинного навчання у сфері рекомендаційних систем для ресторанів, та побудовою моделі штучного інтелекту в розрізі рекомендацій. Ця модель призначена для подальшої інтеграції у веб-додаток, що полегшить безпосередню взаємодію та залучення користувачів. Метою проекту є створення додатку, що слугує інтерфейсом між обчислювальними моделями машинного навчання та кінцевими користувачами, призначена для надання індивідуальних пропозицій щодо закладів харчування.

Завдання роботи. З мети випливає, що завданням кваліфікаційної роботи є розробка веб застосунку для перегляду рекомендацій. Застосунок повинен бути побудований на реальних даних (існуючих закладах) та надавати кінцевому користувачеві зручний інтерфейс.

Під час роботи над кваліфікаційною роботою було розв’язано такі задачі:

- зібрання масиву даних про заклади харчування в місті Чернівці та оцінки користувачів про них;
- підбір алгоритму машинного навчання для рекомендаційної системи;

- створення та тренування моделі штучного інтелекту на основі зібраних даних та підібраного алгоритму;
- розробка веб-застосунку для перегляду персоналізованих рекомендацій користувачу.

Створений застосунок складається з серверної частини, сервісу рекомендацій та клієнтської частин. Для збереження даних використовується реляційна база даних. Під час роботи використовувались мови програмування .NET та Python, фреймворки Angular і ASP.NET, СКБД – SQL Server.

Об’єкт дослідження – алгоритми машинного навчання для рекомендаційних систем, веб додатки та новітні підходи у їх розробці.

Практичне застосування полягає у можливості використання додатку бізнесами в індустрії харчування, як й самі ресторани, так й сервіси по агрегуванню закладів або сервіси доставки. Впроваджуючи більш досконалі системи рекомендацій, ці компанії можуть запропонувати покращений користувацький досвід, тим самим підвищуючи залученість та лояльність клієнтів.

Дипломна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань та додатків.

У першому розділі вивчено тематику кваліфікаційної роботи, а також уже створені подібні веб додатки. Другий розділ містить у собі огляд обраних для розробки технологій, їхніх переваг та недоліків, а також можливостей. Третій розділ складається з опису етапів розробки веб застосунку, описано фрагменти коду програми. Четвертий розділ слугує інструкцією для користувача додатку.

Під час написання дипломної роботи та створення веб застосунку використано інтернет ресурси за посиланнями [1-9].

1 Огляд літератури за тематикою кваліфікаційної роботи

1.1 Проблематика

У цифрову епоху на ландшафт систем рекомендацій ресторанів дедалі більше впливає поява нових закладів та виклики, які створюють поширені онлайн-платформи, такі як Google Maps та TripAdvisor. Ці платформи, хоча й широко використовуються, часто критикують за їхнє перенасичення рекламою та поширеність фальшивих відгуків. За статистикою, ця проблема є значною: близько 30% відгуків клієнтів в Інтернеті вважаються фальшивими. Крім того, 93% споживачів визнають, що відгуки в Інтернеті впливають на їхні рішення щодо купівлі, але 54% не купують товар, якщо підозрюють, що відгуки є фейковими. Цю думку підтверджує той факт, що 82% споживачів стикалися з фальшивими відгуками протягом останнього року, що призвело до того, що 28% респондентів не довіряють іншим відгукам, а 26% взагалі не довіряють бренду[1].

Масштаби цієї проблеми полягають не лише у сприйнятті, але й в економічному впливі: фальшиві відгуки коштують приблизно 152 мільярди доларів США щороку. Виявлено, що компанії платять за відгуки від $0,25 до $100, сприяючи цій оманливій практиці. Тривожна швидкість, з якою споживачі стикаються з фальшивими відгуками (85% вважають, що відгуки, які вони читають, "іноді або часто фальшиві"), загострює проблему для справжніх рекомендаційних систем. Ця шахрайська діяльність не лише вводить в оману споживачів, але й надає несправедливі переваги певним закладам, порушуючи рівновагу на ринку. Крім того, повідомляється, що клієнти витрачають на 12% більше під впливом фальшивих відгуків, що підкреслює фінансові наслідки цієї проблеми[1].

В умовах швидкого розвитку цифрових платформ та постійної появи нових закладів харчування існує нагальна потреба у більш достовірному, прозорому та орієнтованому на користувача підході до систем рекомендацій ресторанів. Мета - протидіяти розмиванню довіри та економічним

викривленням, спричиненим фейковими відгуками, тим самим відновлюючи цілісність та ефективність цих систем як для споживачів, так і для бізнесу. Це дослідження спрямоване на вирішення цих проблем шляхом розробки рекомендаційної системи, яка не тільки відображає уподобання користувачів, але й працює з вищим ступенем автентичності та надійності, що відрізняє її від існуючих платформ, які стикаються з цими проблемами.

1.2 Огляд існуючих рекомендаційних додатків

Поточний стан додатків для рекомендацій ресторанів включає кілька відомих платформ, кожна з яких має свої унікальні особливості, переваги та недоліки. Ці додатки відіграли важливу роль у формуванні досвіду відвідування ресторанів у цифрову епоху. Однак такі проблеми, як поширеність фальшивих відгуків і постійний приплив нових ресторанів, створюють значні проблеми для цих систем.

Google Maps [2] – карти Google перетворилися з простого навігаційного інструменту на більш комплексну платформу, яка зберігає інформацію про різноманітні точки інтересу, в тому числі й заклади харчування, та пропонує їх кінцевим користувачам на основі відгуків і фотографій, завантажених клієнтами. Ця функція дозволяє користувачам бачити рекомендації щодо закладів на вкладці "Огляд" і отримувати доступ до пов'язаних відгуків, де ці ресторани обговорюються іншими відвідувачами.

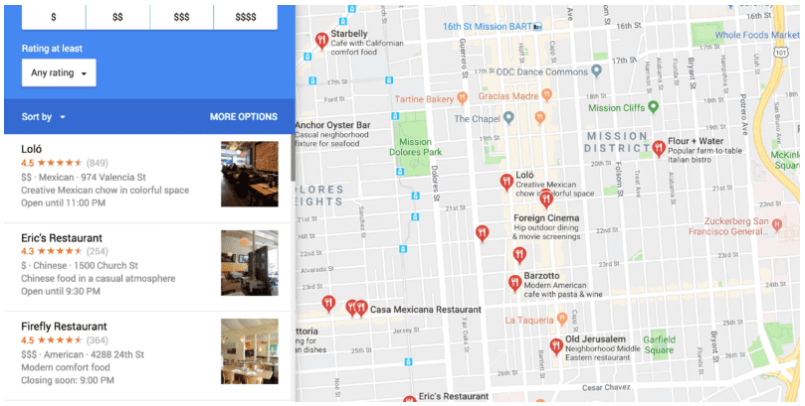

Рисунок 1. Веб-додаток Google Maps

Foursquare [3] – мобільний застосунок, який пропонує можливість швидко знаходити місцеві заклади, в тому числі ресторани, на основі уподобань користувача. Її система з часом навчається і адаптується до вподобань користувачів, пропонуючи більш персоналізовані пропозиції. Платформа також включає в себе відгуки та поради користувачів, які дають змогу отримати уявлення про заклади ще до їх відвідування.

У порівнянні з іншими веб-сайтами, Foursquare має менше відгуків та підприємств у своїй базі даних. Він також стикається з проблемами зі спам-повідомленнями і потенційною наявністю фальшивих відгуків, які можуть вводити користувачів в оману.

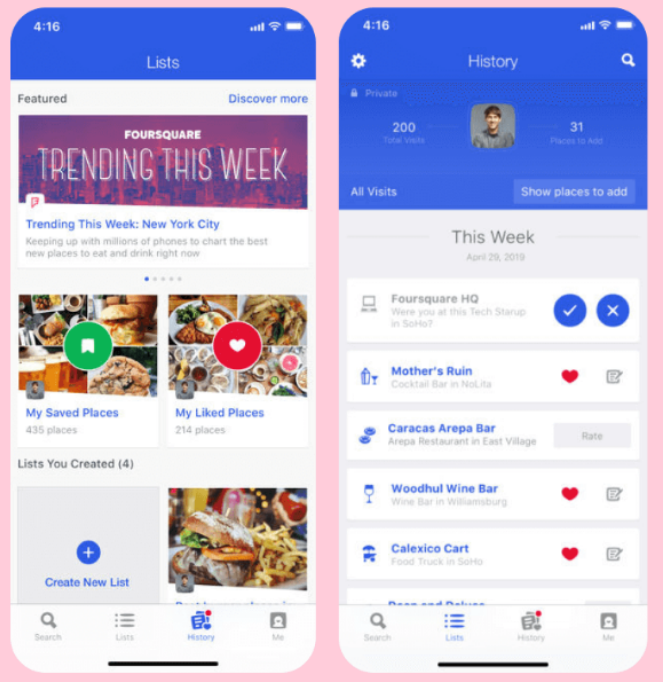

Рисунок 2. Мобільний застосунок Foursquare

Tripadvisor [4] – відома платформа, яка пропонує користувацькі відгуки та рейтинги для різних закладів, включаючи ресторани. Вона надає широкий спектр думок та досвіду від різних користувачів, що допомагає у прийнятті рішень.

TripAdvisor критикують за те, що він дозволяє залишати необґрунтовані анонімні відгуки, що робить його вразливим для фальшивих відгуків. Дослідження, проведене виданням “Which?” показало, що кожен сьомий відгук

про готелі з найвищим рейтингом у популярних туристичних напрямках мав ознаки фейку.

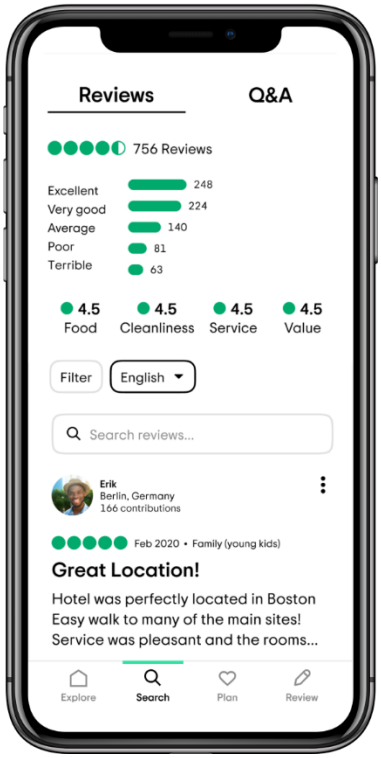

Рисунок 3. Мобільний застосунок Tripadvisor

Restaurant.com [5] – веб додаток, який пропонує унікальний підхід до рекомендацій ресторанів, зосереджуючись на економії та вигідних пропозиціях. Користувачі можуть шукати ресторани за різними критеріями та обирати різні дисконтні сертифікати для своїх страв. Основною перевагою Restaurant.com є його акцент на наданні грошових заощаджень відвідувачам ресторанів. Виклики для таких платформ зазвичай включають достовірність списків ресторанів і реальну вартість запропонованих пропозицій.

Отже, серед інструментів для пошуку закладів наведених вище є додатки різних рівнів та підходів. Проте їх об’єднує відсутність гарантій достовірної інформації. Система рекомендацій, заснована виключно на машинному навчанні, як запропоновано в цьому дослідженні, має на меті пропонувати більш точні, персоналізовані пропозиції, навчаючись на широкому спектрі джерел даних.

2 Огляд технологій розробки веб-додатків та моделей машинного навчання

2.1 Сучасні підходи та інструменти для створення застосунків

У сфері розробки додатків сучасні методології та технологічні інструменти відіграють вирішальну роль, особливо в розробці веб-додатків для таких цілей, як системи рекомендацій ресторанів. В основі цієї сучасної парадигми лежить клієнт-серверна архітектура, яка характеризується поділом додатку на два основні компоненти: клієнт і сервер. Ця архітектурна парадигма покращує масштабованість і керованість додатків.

Клієнт у цій архітектурі - це інтерфейс, призначений для користувача, зазвичай представлений у вигляді веб-браузерів або мобільних додатків. Він відповідає за представлення інформації користувачеві, ініціювання запитів до сервера та обробку відповідей сервера. Сервер, навпаки, працює віддалено, виконуючи основну частину обробки, зберігання даних та основних операцій додатку. Таке розмежування дозволяє ефективно розподіляти робоче навантаження, коли сервер виконує більш складні обчислювальні завдання, а клієнт зосереджується на взаємодії з користувачем і представленні даних.

У сфері інструментів і технологій розробки кілька з них є ключовими для створення сучасних веб-додатків:

- Мови програмування, такі як JavaScript або TypeScript, у поєднанні з фреймворками React, Angular або Vue.js, зазвичай використовуються для створення адаптивних та інтерактивних інтерфейсів на стороні клієнта. Для розробки на стороні сервера часто використовують такі мови, як .NET, Ruby або Java, у поєднанні з такими фреймворками, як ASP.NET, Rails або Spring, які пропонують комплексні інструменти для створення масштабованих веб-додатків.
- Інтеграція інтерфейсів прикладного програмування (API), зокрема RESTful сервісів, є стандартною практикою в сучасній розробці веб-додатків. Ці API забезпечують безперешкодний зв'язок між різними

програмними компонентами, полегшуючи інтеграцію різних систем, таких як бази даних, моделі машинного навчання та сторонні сервіси.

- Системи управління базами даних, що охоплюють як SQL, так і NoSQL бази даних, такі як SQL Server, MySQL або MongoDB, є невід'ємною частиною для ефективного зберігання та пошуку даних. Вибір бази даних, як правило, залежить від конкретних вимог програми до даних і потреб у масштабуванні.
- Системи контролю версій, яскравим прикладом яких є Git, незамінні для управління версіями коду та полегшення співпраці в командах розробників.
- Технології контейнеризації та віртуалізації, такі як Docker та Kubernetes, допомагають у розгортанні та масштабуванні додатків. Ці технології забезпечують портативність та узгодженість додатків у різних операційних середовищах.
- Хмарні сервіси, включаючи такі платформи, як AWS, Azure та Google Cloud Platform, пропонують масштабовану інфраструктуру та набір послуг, необхідних для створення, розгортання та управління додатками.

Використання цих передових підходів та інструментів дозволяє розробникам створювати складні, ефективні та масштабовані веб-додатки. Клієнт-серверна архітектура, що слугує фундаментальною основою, забезпечує чіткий розподіл обов'язків, дозволяючи додаткам ефективно виконувати складні завдання, такі як обробка даних у реальному часі, інтеграція моделей машинного навчання та управління взаємодією з користувачами.

2.2 Сучасні підходи створення моделей машинного навчання

У області машинного навчання та в контексті рекомендаційних систем, сучасні методології значно розвинулися, включаючи в себе передові алгоритми і методи обробки даних. Ці підходи сприяли підвищенню точності, ефективності та персоналізації рекомендаційних систем.

При розробці моделей машинного навчання для рекомендаційних систем зараз часто використовують поєднання різних алгоритмів. До них належать колаборативна фільтрація (collaborative filtering), фільтрація на основі контенту (content filtering) та гібридні методи, які поєднують сильні сторони обох підходів. Колаборативна фільтрація використовує дані про взаємодію користувачів для прогнозування уподобань на основі схожості між користувачами або об'єктами. З іншого боку, фільтрація на основі вмісту та покладається на особливості та характеристики елементів для надання рекомендацій.

Разом з передовими алгоритмами і методами обробки, розробка моделей машинного навчання для рекомендаційних систем також значною мірою спирається на сучасні технології, мови програмування і спеціалізовані бібліотеки. Ці інструменти мають вирішальне значення для реалізації складних методологій, необхідних для ефективних рекомендаційних систем.

Сучасні методології машинного навчання також наголошують на масштабованості та гнучкості. Моделі розроблені так, щоб ефективно обробляти зростаючі обсяги даних і користувачів, гарантуючи, що система рекомендацій залишається ефективною і швидко реагує при масштабуванні. Крім того, гнучкість для адаптації до змін у поведінці та вподобаннях користувачів є ключовим фактором при розробці сучасних моделей

Здатність обробляти та аналізувати дані в режимі реального часу - ще один важливий аспект сучасних моделей машинного навчання для рекомендаційних систем. Такий підхід дозволяє динамічно оновлювати рекомендації на основі взаємодії з користувачами в режимі реального часу та зворотного зв'язку, тим самим підвищуючи релевантність і своєчасність пропозицій.

Серед популярних мов програмування для побудови моделей машинного навчання можна виділити наступні:

- Python є домінуючою мовою в машинному навчанні завдяки своїй простоті та великій кількості бібліотек, які вона підтримує. Читабельність і лаконічний синтаксис роблять її ідеальною для швидкої розробки.

- R - ще одна мова, якій надають перевагу для статистичного аналізу та візуалізації даних, що має вирішальне значення на дослідницьких етапах розробки моделей машинного навчання.
- Java і Scala також використовуються, особливо в системах, де потрібна інтеграція з внутрішніми системами рівня підприємства.

2.3 Мова програмування Python

У сфері розробки рекомендаційних систем на основі алгоритмів машинного навчання мова програмування Python є квінтесенцією вибору, що вирізняється своєю універсальністю, широкою бібліотечною підтримкою та підтримкою спільноти. Популярність Python я пояснюється кількома ключовими характеристиками, які ідеально відповідають вимогам сучасних обчислювальних задач.

Синтаксис мови Python відрізняється своєю ясністю та читабельністю, що робить її доступною мовою для розробників різних рівнів кваліфікації. Ця простота прискорює процес розробки, оскільки дозволяє відносно легко переводити складні алгоритми в код. Простий синтаксис мови зменшує ймовірність помилок у процесі написання коду та покращує його підтримуваність, що є критично важливим аспектом в ітеративному процесі розробки моделей машинного навчання.

Одна з переваг Python полягає в його багатій екосистемі бібліотек і фреймворків, спеціально розроблених для машинного навчання та аналізу даних. Такі бібліотеки, як NumPy та Pandas, надають надійні інструменти для маніпулювання та аналізу даних, які є основою для попередньої обробки наборів даних для рекомендаційних систем. Для задач машинного та глибокого навчання такі бібліотеки, як Scikit-learn, TensorFlow та PyTorch, пропонують широкий функціонал для побудови та навчання складних моделей. Цей широкий спектр бібліотек не лише спрощує процес розробки, але й забезпечує розробникам доступ до найсучасніших інструментів.

Python має велику та активну спільноту, яка робить свій внесок у постійно зростаюче сховище модулів та бібліотек. Ця спільнота є безцінним ресурсом для розробників, пропонуючи обширну документацію, навчальні посібники та форуми для обговорення. Спільнота Python сприяє постійному вдосконаленню та інноваціям у її бібліотеках, тримаючи їх на передовій технологічного прогресу.

Адаптивність Python до нових тенденцій у машинному навчанні та науці про дані ще більше зміцнює його позиції як провідного вибору. Дизайн мови та екосистема бібліотек постійно розвиваються, враховуючи досягнення в галузі штучного інтелекту та машинного навчання, гарантуючи, що системи на основі Python залишатимуться актуальними та передовими.

Отже, вибір Python як мови програмування для розробки моделі машинного навчання рекомендаційної системи підкріплений його простотою, великими бібліотеками, підтримкою спільноти, а також можливістю інтеграції та масштабування. Ці характеристики роблять Python вдалим вибором для вирішення складних завдань, притаманних створенню складних, ефективних та адаптивних систем рекомендацій.

2.4 Мова програмування C#

Мова програмування C#, розроблена компанією Microsoft, є парадигмою сучасного об'єктно-орієнтованого програмування. Відома своєю надійністю та універсальністю, мова C#, що працює з .NET фреймворком та середовищем виконання CLR, чудово справляється з широким спектром завдань з розробки додатків - від простих десктопних додатків до складних веб-сервісів.

Принципи проектування C# наголошують на ясності, простоті та виразності, що полегшує розробку програмних систем, які легко підтримуються та масштабуються. Її об'єктно-орієнтовані можливості та багаті бібліотеки класів дозволяють розробникам створювати ефективні, безпечні та надійні додатки. У веб-розробці, особливо в рамках фреймворку ASP.NET, C#

забезпечує безшовну інтеграцію з серверною логікою, що дозволяє створювати динамічні веб-сторінки та сервіси.

Широка підтримка мовою сучасних парадигм програмування, включаючи асинхронне програмування та LINQ (Language Integrated Query), дозволяє розробникам ефективно вирішувати складні обчислювальні завдання. Ця можливість є особливо корисною в додатках, керованих даними, таких як рекомендаційні системи, де ефективність обробки та обробка даних має першорядне значення.

2.5 Фреймворк ASP.NET

ASP.NET, ключовий компонент фреймворку .NET від Microsoft, є потужним інструментом для створення веб-додатків та сервісів. Це серверний фреймворк веб-додатків, який дозволяє розробникам створювати веб-додатки, сервіси та API. Відомий своєю гнучкістю, безпекою та продуктивністю, ASP.NET підтримує створення веб-сторінок та сервісів, які можуть обслуговувати широкий спектр клієнтів, включаючи браузери та мобільні пристрої. Він пропонує повний набір інструментів і бібліотек, що полегшує швидку розробку і розгортання надійних, масштабованих веб-додатків. У контексті розробки рекомендаційної системи фреймворк ASP.NET, особливо його остання версія ASP.NET 8, стає потужною основою для бекенд-хостингу. Надійні можливості цього фреймворку роблять його особливо придатним для організації взаємодії між сервісом машинного навчання на основі Python та інтерфейсом користувача, що є критично важливим компонентом у системах рекомендацій.

ASP.NET 8 значно посилює аспекти безпеки та продуктивності, що мають вирішальне значення для рекомендаційних систем. Вбудовані функції безпеки фреймворку, такі як покращена підтримка захисту від підробок, захищають від поширених веб-вразливостей, забезпечуючи захист даних користувачів і цілісність системи. Таке посилення безпеки є особливо важливим для

рекомендаційних систем, які часто обробляють конфіденційні дані та уподобання користувачів.

З точки зору продуктивності, ASP.NET 8 забезпечує ефективне використання ресурсів і швидкий час відгуку (response time), навіть під навантаженням складних завдань з обробки даних, притаманних рекомендаційним системам. Таким чином, поєднання в ASP.NET 8 надійних заходів безпеки та підвищеної продуктивності робить його зразковим вибором для фреймворку внутрішньої частини рекомендаційних систем, забезпечуючи безпечне, ефективне та масштабоване середовище.

2.6 Мова програмування TypeScript

TypeScript, надбудова JavaScript, розроблена Microsoft, швидко набула популярності у сфері веб-розробки. Вона збагачує JavaScript статичною типізацією та об'єктно-орієнтованим програмуванням, пропонуючи більш структурований та масштабований підхід до створення складних додатків. Ключова перевага TypeScript полягає в його здатності забезпечувати перевірку типів під час компіляції, що значно зменшує кількість помилок під час виконання, що є вирішальним аспектом для розробки надійних та зручних для підтримки додатків.

У контексті фронтенд-розробки, особливо з такими фреймворками, як Angular, React та Vue.js, TypeScript полегшує створення надійних, масштабних додатків. Його сумісність з JavaScript забезпечує безперешкодну інтеграцію з існуючими бібліотеками та фреймворками JavaScript, підвищуючи продуктивність розробників. Для рекомендаційних систем потужна типізація та об'єктно-орієнтовані можливості TypeScript дозволяють розробляти складні інтерактивні користувацькі інтерфейси, які можуть ефективно обробляти та відображати дані, що обробляються внутрішніми системами.

Переваги мови програмування TypeScript:

- Статична типізація: Пропонує перевірку помилок під час компіляції, що може запобігти помилкам під час виконання.
- Об'єктно-орієнтоване програмування: Підтримує класи, інтерфейси та успадкування, що сприяє кращій організації та модульності.
- Простота рефакторингу: Безпечний рефакторинг коду з меншим ризиком внесення помилок.
- Широка підтримка IDE: Покращений досвід розробки завдяки інструментам автозавершення, навігації та налагодження.
- Сумісність з JavaScript: Код TypeScript компілюється в JavaScript, забезпечуючи сумісність з будь-яким середовищем JavaScript.
- Спільнота та екосистема: Потужна підтримка спільноти та інтеграція з популярними фреймворками, такими як Angular, React та Vue.js.

2.7 Фреймворк Angular

Angular, розроблений і підтримуваний компанією Google, є відомим фреймворком веб-додатків, який широко використовується для розробки динамічних односторінкових додатків (SPA). Архітектура Angular базується на високорівневому фреймворку, який використовує TypeScript, надаючи можливості статичної типізації та об'єктно-орієнтованого програмування для розробки інтерфейсів. Цей фреймворк відомий своєю надійністю, пропонуючи повний набір інструментів і функцій, включаючи двостороннє зв'язування даних, модульну розробку та ін'єкцію залежностей.

В контексті створення інтерфейсів для рекомендаційних систем, Angular перевершує інші фреймворки, надаючи структуроване та масштабоване середовище. Його компонентна архітектура дозволяє розробляти багаторазові елементи інтерфейсу, що полегшує створення складних користувацьких інтерфейсів з багатою інтерактивністю. Акцент фреймворку на тестуванні та підтримці в поєднанні з великою документацією та сильною спільнотою

розробників робить Angular надійним та ефективним вибором для створення складних веб-додатків.

Функція двостороннього зв'язування даних в Angular значно спрощує процес розробки, автоматично синхронізуючи компоненти моделі та представлення. Цей аспект особливо корисний для рекомендаційних систем, де взаємодія з користувачем часто призводить до оновлення інтерфейсу в реальному часі. Крім того, вбудовані сервіси Angular та бібліотеки HTTP-зв'язку забезпечують безперебійну інтеграцію з внутрішніми сервісами, що має вирішальне значення для отримання, відображення та оновлення даних на основі користувацьких уподобань та взаємодій.

Крім того, розширені можливості маршрутизації Angular покращують взаємодію з користувачем у SPA, забезпечуючи ефективну навігацію та ліниве завантаження компонентів. Це призводить до швидшої роботи додатків і більш чуйного інтерфейсу, що важливо для залучення користувачів до системи рекомендацій. Загалом, широкі можливості Angular роблять його ідеальним фреймворком для створення інтуїтивно зрозумілих, ефективних і масштабованих інтерфейсів для рекомендаційних систем.

2.8 СКБД SQL Server

Microsoft SQL Server - це система аналізу і керування базами даних в рішеннях електронної комерції, виробничих галузей і сховищ даних, створена компанією Microsoft. Серед основних переваг можна виділити наступні:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

3 Теоретичний опис побудови платформи

3.1 Створення серверного додатку у вигляді RESTful API

REST (Representational State Transfer) встановлює набір архітектурних принципів для розробки веб-сервісів, що зосереджені на управлінні системними ресурсами. Ці принципи включають механізми обробки та передачі станів ресурсів через протокол HTTP, що дозволяє різноманітним клієнтським програмам, створеним з використанням різних технологій, взаємодіяти з цими ресурсами. Протягом останніх років, REST здобула широке визнання як провідна методологія у сфері розробки веб-сервісів.

Засновником REST є Рой Філдінг, який представив цю концепцію в 2000 році у своїй докторській дисертації "Архітектурні стилі та дизайн мережевих програмних систем" в Каліфорнійському університеті в Ірвайні. У своїй роботі Філдінг описав набір архітектурних принципів для програмного забезпечення, які використовують Інтернет як платформу для розподілених обчислень.

Для імплементації REST-орієнтованого веб-сервісу необхідно дотримуватися чотирьох основних принципів:

- Явне застосування методів HTTP.
- Відсутність збереження стану (statelessness).
- Використання URI, що мімікрують структуру файлової системи.
- Передача даних у форматах XML, JavaScript Object Notation (JSON), або комбінації цих форматів.

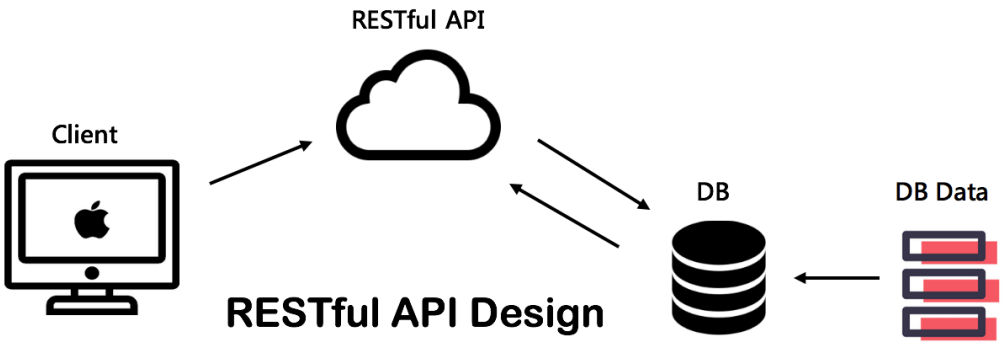

Рисунок 4. Схематичне зображення взаємодії з RESTful API

3.2 Реалізація серверної логіки з Onion архітектурою

Більшість конвенційних архітектурних парадигм стикаються з критичними викликами, пов'язаними з тісним зв'язком компонентів та розділенням обов'язків. Onion Architecture пропонує рішення для проблем, характерних для трьох-рівневих та багаторівневих (n-ярусних) архітектур, забезпечуючи ефективне розподілення функціональних завдань. Шари цієї архітектури взаємодіють між собою через інтерфейси.

Основою "цибулевої" архітектури є принцип інверсії управління. Вона структурована як ряд концентричних шарів, що спрямовані до центрального ядра, яке представляє домен. Відмінною рисою цієї архітектури є її незалежність від рівня даних, на відміну від традиційних багаторівневих архітектур, орієнтованих на доменні моделі.

У контексті традиційної архітектури, рівень користувальницького інтерфейсу взаємодіє з бізнес-логікою, яка, у свою чергу, взаємодіє з рівнем даних, при цьому всі шари тісно пов'язані та залежні один від одного. У трьох-рівневих та багаторівневих архітектурах відсутня незалежність між шарами, що ускладнює розуміння та підтримку таких систем. Одним із основних недоліків традиційних архітектур є надмірне взаємозалежне зв'язування компонентів.

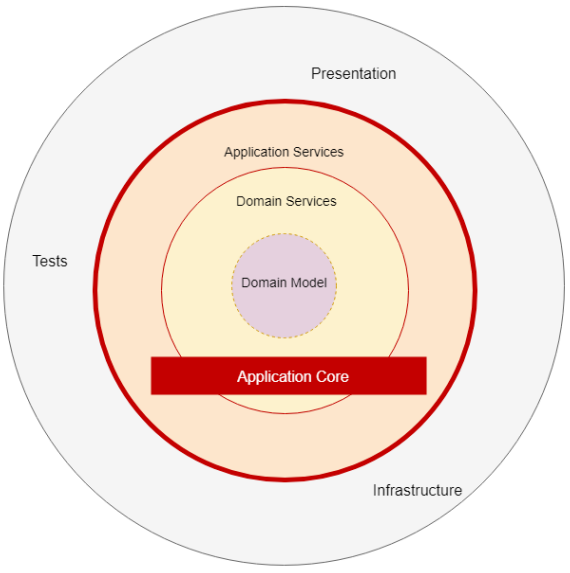

Рисунок 5. Схема Onion архітектури

3.3 Аспекти інверсії керування та використання Dependency Injection

Впровадження залежностей (Dependency Injection) являє собою процес інтеграції зовнішньої функціональності в програмний компонент. Цей процес є конкретним застосуванням концепції "інверсії управління" для управління залежностями компонентів. Згідно з одним з основних принципів розробки програмного забезпечення SOLID, а саме принципом єдиної відповідальності, об'єкт повинен делегувати створення та управління необхідними йому залежностями зовнішньому механізму, спеціально призначеному для цієї мети.

Традиційно, якщо об'єкту потрібен доступ до певного сервісу, він самостійно визначає спосіб доступу до цього сервісу, будь то безпосереднє посилання на місцезнаходження сервісу або використання «сервіс-локатора» для отримання необхідного типу сервісу. Проте, використовуючи впровадження залежностей, об'єкт лише визначає властивість для зберігання посилання на тип потрібного сервісу. При створенні об'єкта посилання на відповідний тип сервісу автоматично інтегрується в цю властивість, використовуючи зовнішній механізм управління залежностями.

Такий підхід забезпечує вищу гнучкість, оскільки спрощує створення альтернативних реалізацій сервісу та їх конфігурацію, наприклад, через конфігураційні файли, без необхідності внесення змін у об'єкти, що використовують цей сервіс. Це особливо корисно для модульного тестування, дозволяючи легко інтегрувати "заглушки" сервісів у тестовані об'єкти.

4 Постановка завдання та програмна реалізація

4.1 Постановка задачі

Метою цієї кваліфікаційної роботи є розробка комплексної системи рекомендацій для ресторанів, використовуючи можливості машинного навчання та веб-технологій. Ця система покликана допомогти користувачам знаходити ресторани, які відповідають їхнім вподобанням та потребам. Для досягнення цієї мети необхідно вирішити такі завдання:

- Збір набору даних з відкритих джерел, зокрема, з Google Map Reviews, щоб зібрати різноманітні та широкі відгуки користувачів і характеристики ресторанів. Чим більше та якісніше будуть дані, тим точніше буде працювати модель машинного навчання, тому цей етап є фундаментально важливим.
- Оцінка та вибір відповідного алгоритму машинного навчання. Цей крок передбачає навчання моделі на зібраному наборі даних для забезпечення точних та персоналізованих рекомендацій ресторанів.
- Розробка веб-хостингового додатку, що слугуватиме внутрішньою інфраструктурою. Цей додаток відповідатиме за обробку даних, запитів користувачів та взаємодію з моделлю машинного навчання.
- Створення інтерфейсного додатку, розробленого за сучасними принципами UI/UX. Цей додаток забезпечить інтуїтивно зрозумілий та зручний інтерфейс для взаємодії користувачів з рекомендаційною системою, отримання пропозицій та надання зворотного зв'язку.

Веб-додаток, що слугуватиме точкою оркестрування, використовуватиме ASP.NET для внутрішньої частини, забезпечуючи надійну продуктивність та безпеку. Інтеграція між моделлю машинного навчання та веб-додатком буде ключовою, а для розробки та запуску алгоритмів машинного навчання буде використовуватися Python. Angular слугуватиме як бібліотека для розробки користувацького інтерфейсу.

4.2 Створення датасету та методи збирання даних

У основі будь-якого алгоритму машинного навчання та штучного інтелекту фундаментом, та безпосередньо, найпершою і найважливішою задачею є створення комплексного набору даних та розробка надійної методології їх збору. Для вирішення цієї задачі існують різні методи, кожен з яких пропонує унікальну інформацію та типи даних:

- API та веб-сервіси: Часто використовуються для доступу до структурованих даних з відомих платформ.
- Краудсорсинг: Цей метод використовує можливості спільноти для збору різноманітних даних.
- Опитування та форми зворотного зв'язку: Збираються прямі відгуки користувачів, щоб зрозуміти їхні вподобання та досвід.
- Купівля даних: Набори даних можна придбати у спеціалізованих постачальників даних.
- Аналіз соціальних мереж: Дані з соціальних платформ дають уявлення про громадську думку та тенденції.
- Співпраця з бізнесом: Співпраця з відповідними компаніями може дати ексклюзивні дані.

У цій кваліфікаційній роботі датасет складається з даних, витягнутих з Google Maps, з фокусом на заклади харчування в місті Чернівці та відгуки клієнтів. Збір цієї інформації був здійснений за допомогою методики Scraping, коли програмне забезпечення емулює реального користувача, що користується браузером, й далі витягує всю необхідну інформацію з розмітки HTML.

Ці скрипти були розроблені для завантаження та методичного вилучення релевантних даних, включаючи назви ресторанів, місцезнаходження, типи кухні, оцінки клієнтів та текстові відгуки. Процес вилучення регулювався структурованим підходом, що забезпечило широке охоплення різних типів закладів харчування та багату компіляцію відгуків клієнтів.

Ця методологія дозволила накопичити різноманітний і великий набір даних, необхідний для навчання нюансованої моделі машинного навчання, здатної надавати персоналізовані рекомендації щодо ресторанів.

Крім вилучення, методологія збору даних включає етапи очищення та попередньої обробки даних.

Результатом став датасет закладів міста Чернівці та їх відгуків з такими характеристиками:

- 500 й більше закладів;
- 100’000 й більше відгуків.

4.3 Оцінка та вибір відповідного алгоритму машинного навчання

У пошуках відповідного алгоритму машинного навчання для системи рекомендацій ресторанів важливо критично оцінити наявні на ринку алгоритми. Сфера машинного навчання пропонує різноманітний набір алгоритмів, кожен з яких має свої сильні сторони та можливості застосування:

- Колаборативна фільтрація (Collaborative filtering): Цей алгоритм надає рекомендації на основі схожості поведінки користувачів. Він дуже ефективний у сценаріях, де є багато даних про уподобання користувачів.
- Фільтрація на основі вмісту (Content-Based filtering): Пропонує заклади харчування (або інші речі, в залежності від сфери), схожі на ті, які користувач вподобав у минулому, значною мірою покладаючись на аналіз характеристик товарів.
- Гібридні методи: Поєднують сильні сторони спільної роботи та фільтрації на основі контенту та уподобань користувачів, часто надаючи більш точні та надійні рекомендації.
- Підходи глибокого навчання: Нейронні мережі, зокрема згорткові та рекурентні нейронні мережі, використовуються для складного

розпізнавання образів, що підходить для великих і динамічних наборів даних.

- Обробка природної мови (NLP): Методи NLP використовуються для аналізу та інтерпретації відгуків користувачів, витягуючи цінну інформацію для персоналізованих рекомендацій.

4.3.1 Колаборативна фільтрація

Колаборативна фільтрація (КФ) - це широко використовувана техніка в рекомендаційних системах, яка в першу чергу фокусується на прогнозуванні уподобань користувача на основі вподобань інших користувачів. Він працює за принципом, що користувачі, які погодилися в минулому, погодяться і в майбутньому щодо певних пунктів.

Існує два основних типи КФ:

- Спільна фільтрація "користувач-користувач": Цей метод знаходить користувачів, схожих на цільового користувача, і рекомендує елементи, які сподобалися цим користувачам.
- Спільна фільтрація "товар-товар": Замість того, щоб шукати схожих на користувача, він знаходить схожих на товар. Якщо користувачеві сподобався товар, система рекомендує схожі товари.

4.3.2 Матрична факторизація

Матрична факторизація - це специфічна техніка в КФ, особливо ефективна для роботи з великими, розрідженими наборами даних. Вона розкладає матрицю взаємодії між користувачем і товаром на матриці меншої розмірності, що представляють латентні фактори користувачів і товарів. Такі методи, як сингулярна декомпозиція (SVD), широко використовуються для факторизації матриць. Цей підхід допомагає виявити приховані особливості, що лежать в основі взаємодії між користувачами і товарами, покращуючи якість рекомендацій.

Моделі матричної факторизації відомі своєю масштабованістю і здатністю обробляти великі набори даних, що робить їх придатними для сценаріїв з великою кількістю користувачів і товарів.

Матрична факторизація є особливо потужною завдяки своїй здатності перетворювати взаємодію користувача та об'єкта на представлення прихованих факторів. Ці фактори можуть представляти приховані характеристики об'єктів або уподобання користувачів, які не є доступними в явному вигляді. Наприклад, у системі рекомендацій ресторанів ці фактори можуть відповідати типу кухні, ціновому діапазону або вподобанням щодо атмосфери.

Ключова перевага використання матричної факторизації полягає в її здатності передбачати невідомі рейтинги шляхом реконструкції вихідної матриці. Такі алгоритми, як SVD, адаптовані для цих завдань, щоб впоратися з відсутніми значеннями, типовими для реальних наборів даних. Розбиваючи матрицю користувацьких елементів на латентні ознаки, він фіксує основну структуру даних, що дає змогу робити точніші прогнози.

Однак, серед викликів - вирішення проблем "холодного старту" (нові користувачі або об'єкти з обмеженою кількістю даних про взаємодію) та динамічна природа вподобань користувачів. Передові методи, такі як інтеграція часової динаміки або використання гібридних моделей з підходами, заснованими на контенті, можуть підвищити ефективність матричної факторизації в колаборативній фільтрації.

4.3.3 Метод найменших квадратів (Alternating Least Squares)

Метод найменших квадратів (ALS) - це алгоритм машинного навчання, який зазвичай використовується в рекомендаційних системах, особливо ефективний для наборів даних з неявним зворотним зв'язком. Неявний зворотний зв'язок - це дані, зібрані пасивно від взаємодії користувачів, такі як перегляди, кліки або історія покупок, на відміну від явного зворотного зв'язку, такого як рейтинги або відгуки.

ALS - це варіант методів матричної факторизації, що використовується в колаборативній фільтрації. Він вирішує проблему неявного зворотного зв'язку, який є численним, але менш інформативним, ніж явний зворотний зв'язок. На відміну від явного зворотного зв'язку, неявний зворотний зв'язок не є прямим показником уподобань користувача, оскільки йому бракує чітких негативних сигналів і він часто має викривлений розподіл.

Алгоритм ALS працює шляхом декомпозиції матриці взаємодії користувач-об'єкт на дві матриці меншої розмірності, що представляють латентні фактори для користувачів та об'єктів. Ключовою відмінністю ALS, особливо для неявного зворотного зв'язку, є спосіб обробки відсутніх даних. ALS припускає, що всі відсутні дані є негативними сигналами, але з нижчим рівнем достовірності, ніж позитивні взаємодії. Алгоритм чергує фіксацію користувацьких факторів і розв'язання для факторів об'єкта, і навпаки, мінімізуючи помилку найменших квадратів у прогнозах.

ALS особливо добре справляється з великими наборами даних завдяки тому, що його можна розпаралелювати. Він може ефективно виявляти приховані фактори в даних, що призводить до точних рекомендацій. ШНМ також пом'якшує надмірне припасування за допомогою регуляризації, що дуже важливо з огляду на розрідженість і високу розмірність наборів даних неявного зворотного зв'язку.

Підсумовуючи, можна сказати, що ALS - це надійний алгоритм для рекомендаційних систем з неявним зворотним зв'язком. Його здатність працювати з великими розрідженими наборами даних і виявляти значущі латентні фактори робить його популярним у сценаріях, де відсутні явні уподобання користувачів. Однак для ефективного застосування цього методу необхідно ретельно враховувати його припущення та потенційні обмеження.

4.3.4 Підбір алгоритму до створеного датасету

У процесі оцінки набору даних, отриманого з Google Reviews для системи рекомендацій ресторанів, було помічено, що в ньому переважають позитивні відгуки. Такий перекіс у бік позитивних відгуків є загальною характеристикою багатьох платформ користувацького контенту, де користувачі з більшою ймовірністю залишають відгуки, коли їхній досвід є позитивним. Також було помічено, що середня кількість відгуків на одного унікального клієнта – 2 заклади харчування.

Враховуючи цю характеристику набору даних, алгоритм Alternating Least Squares (ALS) виявився найефективнішим на етапі оцінки. Здатність ALS обробляти неявний зворотний зв'язок, який притаманний таким позитивно зміщеним наборам даних, зробила його належним вибором для цієї кваліфікаційної роботи. Для реалізації ALS було обрано бібліотеку Python "implicit". Ця бібліотека спеціально розроблена для сценаріїв неявного зворотного зв'язку і пропонує ефективну та масштабовану реалізацію алгоритму ALS, що робить її ідеальним вибором для обробки нюансів вилученого набору даних.

Варто відмітити, що алгоритм Alternating Least Squares та бібліотека implicit пропонує переконливе рішення проблеми холодного старту, яка є поширеною проблемою в рекомендаційних системах. Проблема холодного старту виникає, коли нові користувачі або елементи вводяться в систему без достатньої кількості даних про взаємодію з об’єктами рекомендацій, що створює ситуацію, коли модель нічого не може сказати про користувача, про якого нічого не знає.

У цьому проекті було застосовано стратегічний підхід для пом'якшення цієї проблеми. В рамках процесу входження в систему користувачам пропонується вибрати кілька ресторанів, які їм подобаються (див. розділ 5). Цей початковий вхід надає цінні дані для ефективної інтеграції нових користувачів у систему. Примітно, що бібліотека “implicit” полегшує додавання цих нових взаємодій між користувачем та елементами без необхідності перенавчання всієї

моделі. Ця функція гарантує, що додаток зберігає свою продуктивність і забезпечує безперебійну роботу користувачів. З іншого боку, якщо є потреба додати нові заклади харчування, бібліотека також надає цю можливість без необхідності робити повне тренування моделі знову. Здатність динамічно додавати нові дані, зберігаючи при цьому загальну цілісність і швидкість реакції системи, є значною перевагою, що сприяє безперешкодній інтеграції нових користувачів у платформу рекомендацій.

4.4 Архітектура системи

Технічна архітектура системи рекомендацій ресторанів - це багатокомпонентний дизайн, що гармонізує різні технології для надання цілісного та ефективного сервісу.

Веб-хост .NET: В основі лежить веб-хост .NET, який виступає в ролі оркестратора системи. Він забезпечує інтерфейс між користувацьким Angular-додатком та внутрішнім сервісом рекомендацій на Python. Хост .NET керує запитами API, маршрутизує дані між базою даних SQL та сервісом Python, а також виконує іншу логіку на стороні сервера.

База даних SQL Server: Ця база даних є ключовою для зберігання профілів користувачів, інформації про ресторан та даних про взаємодію. Вона слугує механізмом постійного зберігання, що дозволяє ефективно отримувати дані та керувати ними на хості .NET.

Рекомендаційний сервіс Python: Цей сервіс відповідальний за тренування, доповнення моделі та надання рекомендацій.

Клієнтський додаток Angular: Цей фронтенд-додаток призначений для взаємодії з клієнтами. Створений на Angular, він забезпечує чуйний та інтуїтивно зрозумілий користувацький інтерфейс, що дозволяє користувачам переглядати рекомендації, надавати відгуки та персоналізувати свій досвід. Додаток Angular взаємодіє з веб-хостом .NET для отримання даних і відображення їх користувачеві.

Схему зв’язку між поданими трьома елементами наведено нижче у рис. 6.

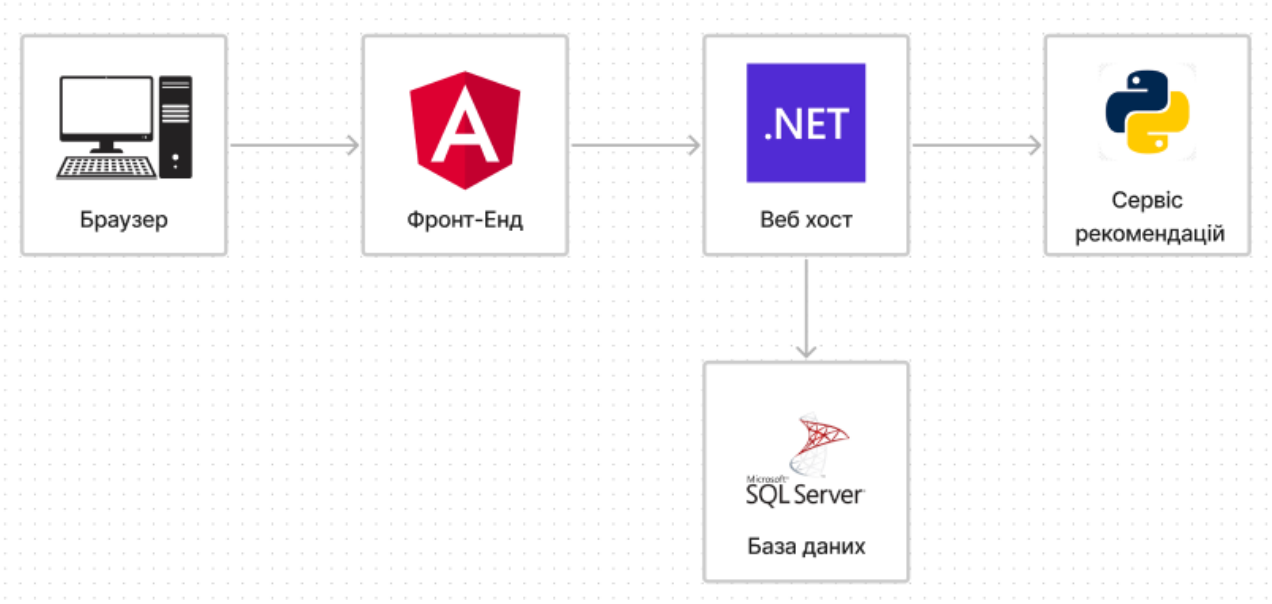

Рисунок 6. Архітектура компонентів системи

Технічна архітектура системи рекомендацій ресторанів розроблена таким чином, щоб забезпечити безперебійний зв'язок між її різними компонентами за допомогою HTTP та RESTful API. Цей стандартний протокол гарантує, що взаємодія між веб-хостом .NET, базою даних SQL Server, додатком Angular UI та системою рекомендацій на Python є послідовною, надійною та безпечною.

Використання HTTP та RESTful API дозволяє системі обробляти запити та обмінюватися даними в клієнт-серверній архітектурі без стану, що має вирішальне значення для масштабованості та гнучкості. Такий підхід дозволяє кожному компоненту працювати незалежно, але узгоджено, забезпечуючи модульність та інтегрованість системи. Такий дизайн має важливе значення для ефективного функціонування такої складної системи, як сервіс рекомендацій ресторанів, де обробка даних у реальному часі та взаємодія з користувачем мають першорядне значення.

У поточній конфігурації кожен сервіс системи розміщений на окремому хостингу. Такий незалежний хостинг забезпечує гнучкість і полегшує

управління на початкових етапах життєвого циклу системи. Однак у майбутньому планується перехід до більш масштабованого та ефективного хостингу шляхом докеризації сервісів.

Докеризація дозволить інкапсулювати кожну частину системи у власний контейнер, що зробить систему більш портативною та узгодженою в різних середовищах. Для управління цими контейнерами можуть використовуватися інструменти оркестрування, такі як Docker Compose або Kubernetes. Ці інструменти надають розширені можливості для автоматичного розгортання, масштабування та управління контейнерними додатками, тим самим оптимізуючи операційну ефективність та надійність системи у виробничому середовищі.

4.4.1 Структура Бази Даних.

Структура бази даних системи рекомендацій ресторанів побудована на основі фреймворку ASP.NET Identity, який надає набір таблиць за замовчуванням, призначених для надійної аутентифікації та авторизації користувачів (див розділ 4.4.5). Ці таблиці включають AspNetUsers для профілів користувачів, AspNetRoles для управління ролями, AspNetUserRoles для асоціювання користувачів з ролями, AspNetUserClaims для ідентифікації на основі претензій та AspNetUserLogins для зовнішніх постачальників автентифікації.

Було створено додаткову спеціальну таблицю FoodPlaces для зберігання всіх закладів харчування, яка містить поля для назв, описів, місцезнаходження, типів кухні, рейтингів та інших відповідних атрибутів.

Схема бази даних нормалізована, що усуває надмірність і забезпечує цілісність даних. Ця нормалізація полегшує розширюваність системи, дозволяючи розширювати її в майбутньому, наприклад, додавати нові атрибути або таблиці, не порушуючи цілісності існуючої структури.

Дизайн бази даних відповідає принципам, які передбачають майбутнє зростання та розширення функціональності. Наприклад, якщо додаток почне

збирати додатковий контент, створений користувачами, такий як відгуки, коментарі або інші форми зворотного зв'язку, існуючу структуру можна буде легко розширити. Для зберігання цієї інформації можна додати нові таблиці або стовпці, або розширити існуючі таблиці, такі як FoodPlaces, включивши до них поля для відгуків та оцінок користувачів.

Більше того, зв'язки між цими новими точками даних та існуючими сутностями можуть бути визначені за допомогою зовнішніх ключів, які підтримують цілісність посилань і підтримують складні запити. Нормалізована природа бази даних гарантує, що такі розширення можуть відбуватися без порушення поточної структури даних і взаємозв'язків, зберігаючи загальну продуктивність і надійність системи. Детальну схему бази даних додатку можна побачити на рис. 7.

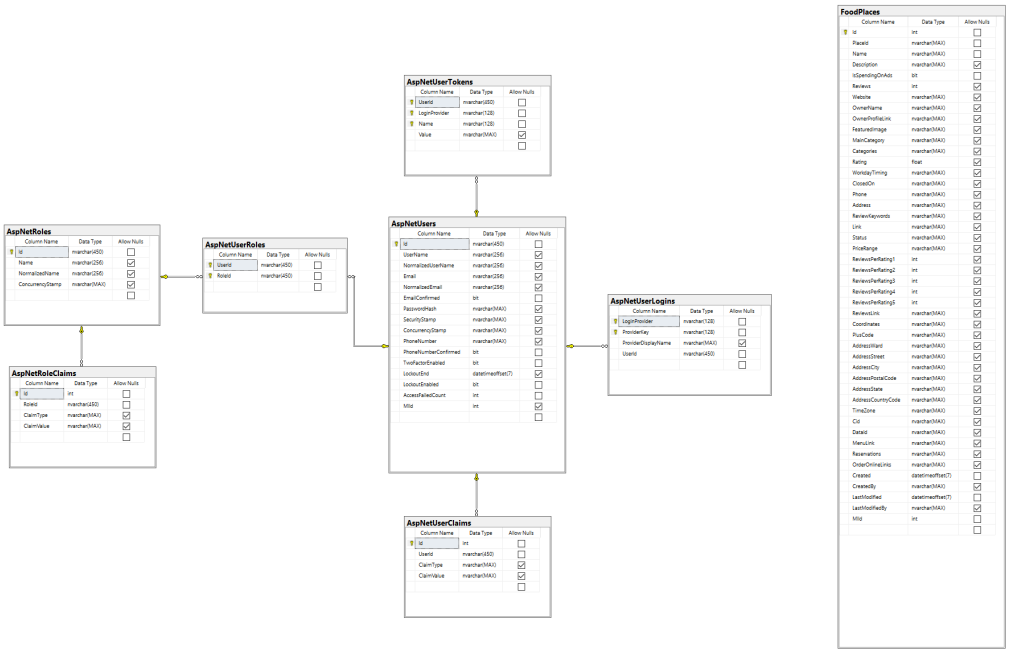

Рисунок 7. Схема бази даних додатку

4.4.2 Зв’язок з базою даних. Зв'язок з базою даних у системі рекомендацій ресторанів здійснюється виключно за допомогою хост-додатку ASP.NET. Такий сфокусований підхід гарантує, що всі операції з даними централізовано виконуються в одній точці, що підвищує безпеку та цілісність

даних. Хост ASP.NET використовує бібліотеку Entity Framework Core (EF Core), сучасний об'єктно-реляційний маппер (ORM), щоб полегшити цю взаємодію.

ORM-бібліотеки, такі як EF Core, абстрагують складність прямих маніпуляцій з базами даних, тим самим мінімізуючи ризики SQL-ін'єкцій, оскільки вони використовують параметризовані запити. Вони керують з'єднаннями та транзакціями з базами даних, сприяючи створенню безпечних та підтримуваних шаблонів доступу до даних у додатках.

EF Core слугує рівнем абстракції над базою даних SQL, дозволяючи програмі виконувати CRUD-операції, використовуючи строго типізовані об'єкти C#. ORM значно спрощує процес взаємодії з базою даних, забезпечуючи підхід до оновлення схеми бази даних, що ґрунтується на коді.

EF Core пропонує значні переваги у продуктивності та вважається дуже зручним для розробників. Вона спрощує операції з базами даних, транслюючи вирази LINQ (Language Integrated Query) у запити SQL, дозволяючи розробникам писати логіку бази даних на C#. Ця ORM також включає механізми кешування, відмовостійкості з'єднань, пакетної обробки та асинхронного виконання запитів, які сприяють її ефективності.

Первинне налаштування зв’язку з базою даних за допомогою бібліотеки EF Core створюється за допомогою створення спеціального класу, що наслідується від класу DbContext. Налаштування цього класу зазвичай відбувається коли веб додаток запускається, й вся конфігурація, наприклад адреса база даних та логін і пароль вказується у файлі Startup.cs.

Приклад:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>, IApplicationDbContext
{
   public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

   public DbSet<FoodPlace> FoodPlaces => Set<FoodPlace>();

   protected override void OnModelCreating(ModelBuilder builder)
   {
      builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

      base.OnModelCreating(builder);
   }
```

}

Entity Framework Core включає потужну функцію, відому як міграції, яка дозволяє розробникам розвивати схему бази даних з часом. У міру зростання додатку і зміни вимог, міграції дозволяють розробникам застосовувати інкрементні оновлення до бази даних без втрати даних. За допомогою набору команд розробники можуть додавати, модифікувати або відкочувати зміни, а EF Core генерує відповідні SQL-скрипти. Цей процес не тільки допомагає версіонувати схему бази даних разом з кодом програми, але й забезпечує чіткий шлях для оновлення бази даних у виробничому середовищі, гарантуючи, що зміни застосовуються послідовно і надійно.

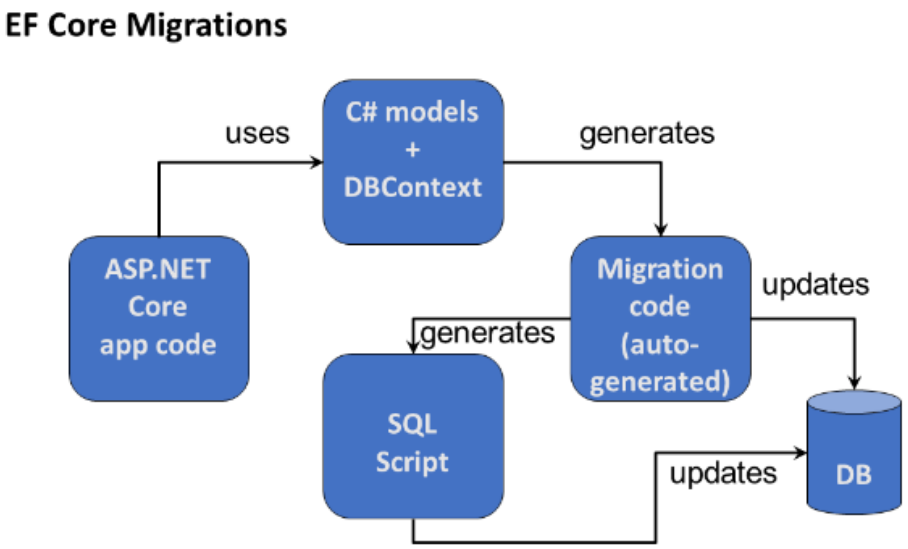

Рисунок 8. Схематичне зображення міграцій в EF Core

Взаємодія додатку з базою даних ретельно спроектована відповідно до принципів Onion Architecture, що забезпечує чіткий розподіл завдань і акцент на основній бізнес-логіці. В основі цієї архітектури лежить прикладний рівень (Application Layer), який містить інтерфейси, що визначають контракт на операції з базою даних. Ці інтерфейси навмисно не залежать від сховища, що забезпечує більшу гнучкість і адаптивність в управлінні збереженням даних.

Конкретні реалізації цих інтерфейсів знаходяться на рівні інфраструктури. Саме тут здійснюються фактичні виклики до бази даних, використовуючи EF Core для перетворення бізнес-операцій у транзакції з даними. Потім ці реалізації передаються назад на прикладний рівень за допомогою ін'єкції залежностей (Dependency Injection, DI). Така інверсія управління не тільки відокремлює логіку додатку від проблем доступу до даних, але й полегшує тестування та супровід модулів, що відповідає сучасним практикам розробки програмного забезпечення (див розділ 3).

4.4.3. Веб ендпоінти. Бізнес логіка. Історично та тривалий час під час розробки веб застосунків за допомогою ASP.NET фреймворка використовувався підхід MVC – модель, представлення та контролер. Цей підхід був і залишається одним з головним під час створення RESTful API, проте все частіше критикується із-за потреби написання шаблонного коду, що раз у раз повторюється. У цій кваліфікаційній роботі був використаний новітній метод, що був випущений у 2021 році з появою .NET 6 – мінімальні ендпоінти (Minimal Endpoints).

Цей метод забезпечує спрощений підхід для обробки HTTP-запитів у прості методи C# з використанням делегатів. Мінімальні ендпоінти зменшують шаблонний код, який зазвичай пов'язаний з налаштуванням контролерів, дозволяючи створювати кінцеві точки HTTP з мінімальними налаштуваннями. Мінімальні ендпоінти можуть бути поєднані в групи, та мають рівноцінні можливості налаштувань, як й класична модель контролерів.

Приклад мінімального ендпоінту, що працює з сутністю закладів харчування:

```
public class FoodPlaces : EndpointGroupBase
{
   public override void Map(WebApplication app)
   {
      app.MapGroup(this)
         .RequireAuthorization()
         .MapGet(GetAllFoodPlaces)
         .MapPost(SaveOnboardingPreferences);
   }

   public async Task<FoodPlacesListDto> GetAllFoodPlaces(ISender sender)
   {
      return await sender.Send(new GetAllFoodPlacesQuery());
   }

   public async Task<IResult> SaveOnboardingPreferences(ISender sender,
SaveUserInitialPreferencesCommand command)
   {
      await sender.Send(command);
      return Results.NoContent();
   }
```

Після досягнення хоста ASP.NET ці запити перенаправляються на прикладний рівень за допомогою патерну проектування Mediator. Медіатор слугує диспетчером повідомлень, який розділяє класи і впорядковує комунікацію, особливо в системах зі складною бізнес-логікою. Він спрямовує запити до відповідних обробників, підвищуючи модульність додатку. У цій архітектурі кожен вхідний HTTP-запит розглядається як запит або команда - кожен зі своїм спеціальним обробником. Запит отримує дані без зміни стану системи, тоді як команда змінює стан системи або викликає побічні ефекти. Обробники запитів і обробники команд відповідають за виконання відповідної бізнес-логіки, забезпечуючи чітке розділення між операціями читання даних і операціями запису даних. Таке розмежування спрощує розуміння поведінки системи та сприяє створенню більш зручного в обслуговуванні та масштабованого програмного забезпечення.

Однією із нагальних потреб будь-якого веб-додатку є валідація вхідних даних, отриманих з HTTP запиту. Це може бути проста валідація щодо максимальних значень тих чи інших параметрів, чи більш комплексна як, наприклад, перевірка чи є достатньо коштів на рахунку клієнта для здійснення

фінансової операції. Часом ці перевірки потребують великої кількості коду та додаткових сервісів, й тим самим можуть затуманювати основну бізнес-логіку.

Завдяки використання Onion архітектури, використання Mediator патерну та розділення всіх операцій в системі на запити та команди, як окремі сутності, з'являється можливість й виділити також й валідаторів.

В архітектурному дизайні цієї роботи кожен запит або команда можуть бути пов'язані з відповідним валідатором, використовуючи можливості FluentValidation або подібних бібліотек. Такий підхід відокремлює логіку валідації від бізнес-операцій, дотримуючись принципу єдиної відповідальності. Валідатори гарантують, що вхідні дані відповідають визначеним критеріям до виконання будь-якої бізнес-логіки, що має вирішальне значення для підтримки цілісності даних і надійності системи. Цей рівень валідації виконує роль воротаря, запобігаючи поширенню недійсних даних у системі та забезпечуючи виявлення помилок на ранніх стадіях обробки запитів. Абстрагуючи валідацію від обробників, кодова база залишається чистою і зосередженою на основних функціональних можливостях.

Приклад валідатора збереження початкових уподобань користувача можна побачити нижче. В цьому валідаторі ми перевіряємо, що кількість вподобаних ресторанів має бути в межах від 0 до 10, й також, що унікальний ідентифікатор кожного вибраного місця має бути не пустим.

```
public class SaveUserInitialPreferencesCommandValidator
   : AbstractValidator<SaveUserInitialPreferencesCommand>
{
   public SaveUserInitialPreferencesCommandValidator()
   {
      RuleFor(x => x.PlacesIds).Must(x => x.Any());
      RuleFor(x => x.PlacesIds).Must(x => x.Count() <= 10);
      RuleFor(x => x.PlacesIds).ForEach(x => x.NotEmpty());
   }
}
```

Якщо перевірка пройшла та результат негативний, веб хост не буде виконувати ту чи іншу команду або запит, й натомість автоматично надасть HTTP відповідь з необхідним кодом помилки та повною інформацією про невірні параметри запиту.

4.4.4 Безпека додатку.

Безпека додатку є першочерговим завданням. Веб-хост .NET несе відповідальність за автентифікацію та авторизацію. Система безпеки цієї системи побудована на надійному фундаменті, який забезпечують ASP.NET Identity та Duende Identity Server.

ASP.NET Identity - це система членства, яка додає функціональність входу в додатки .NET, полегшуючи управління користувачами, паролями, даними профілю, ролями, вимогами, токенами та іншим. Duende Identity Server (раніше IdentityServer4) - це фреймворк OpenID Connect і OAuth 2.0 для ASP.NET Core, який розширює можливості авторизації, дозволяючи реалізувати автентифікацію на основі токенів.

OAuth 2.0 - це галузевий стандарт протоколу авторизації, що широко використовується такими компаніями як Google, Facebook, X.com, Reddit та іншими. Він фокусується на простоті для розробки та використання, забезпечуючи при цьому специфічні потоки авторизації для веб-додатків, десктопних додатків, мобільних телефонів та інших пристроїв. У контексті цієї системи рекомендацій ресторанів OAuth 2.0 полегшує інтеграцію Google Sign-In, дозволяючи користувачам авторизуватися за допомогою своїх облікових записів Google. Ця інтеграція не тільки спрощує користувацький досвід, пропонуючи звичний і безпечний спосіб входу, але й підвищує безпеку додатку, використовуючи надійні механізми аутентифікації Google. Схематичне зображення протоколу Oauth 2.0 дивіться на рис. 9.

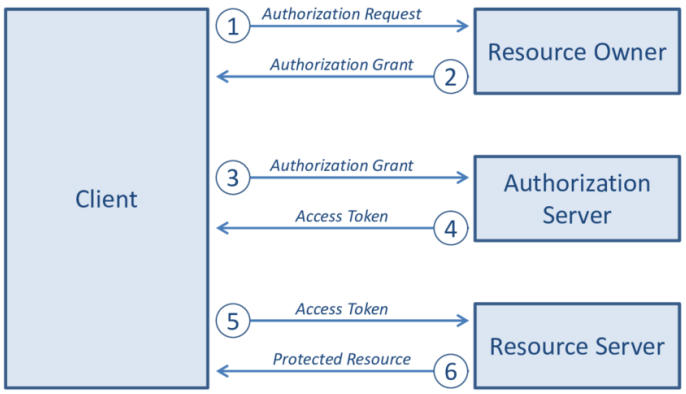

Рисунок 9. Протокол OAuth 2.0

4.4.5 Клієнтський додаток.

Клієнтський додаток системи рекомендацій ресторанів розроблено з використанням Angular, ефективного фреймворку для створення динамічних веб-додатків. Роль Angular у цій архітектурі є ключовою, оскільки він відповідає за всі взаємодії на стороні клієнта.

В архітектурі Angular, кожен розроблений компонент має один вимогливий файл та три додаткові. Основний файл, що є обов'язковим, це файл TypeScript, який включає в себе весь функціонал компоненту та управління його життєвим циклом. Другий файл TypeScript використовується для налаштування тестувань компоненту. Третій файл — це HTML, який містить візуальну розмітку та код сторінки. Четвертий, CSS файл, містить стилі для компоненту. Зазвичай, кожен компонент відповідає за одну веб-сторінку.

Розглянемо детальніше один з компонентів. У коді зазначеному нижче вказано сигнатуру, назву компонента RecommendationsDashboardComponent, що відповідальний за перегляд рекомендацій.

```
@Component({
  selector: 'app-recommendations-dashboard',
  templateUrl: './recommendations-dashboard.component.html',
  styleUrls: ['./recommendations-dashboard.component.css'],
})
export class RecommendationsDashboardComponent {
  public user: UserInfoDto;

  constructor(private client: CurrentUserInfoClient, private router: Router) {
    client.getCurrentUserInfo().subscribe({
      next: function (result) {
        this.user = result;

        if (this.user.onboardingIsDone == true) {
          router.navigate(['onboarding']);
        }
      },
    });
  }
}
```

Цей компонент, позначений декоратором @Component, є важливим елементом архітектури Angular, що інкапсулює шаблон HTML, стиль і пов'язану з ним логіку TypeScript. Він використовує прив'язку даних, основну концепцію Angular, для безперешкодного відображення користувацької інформації в інтерфейсі. Реалізація ін'єкції залежностей очевидна в ін'єкції сервісів CurrentUserInfoClient і Router, що демонструє дизайн Angular в управлінні залежностями.

Компонент також використовує бібліотеку RxJS для обробки асинхронних операцій, що є поширеним шаблоном у сучасних веб-додатках для управління потоками даних та подіями. Використання підписки на Observable, що повертається методом getCurrentUserInfo(), є прикладом такого підходу.

Крім того, логіка умовної маршрутизації всередині компонента підкреслює можливості динамічної навігації, притаманні Angular. Це реалізовано за допомогою router.navigate, який перенаправляє користувача до різних представлень додатку на основі конкретних умов, в даному випадку - статусу користувача.

Приклад розмітки компоненту та перевикористання на інших у RecommendationsDashboard:

```
<div class="flex flex-col h-screen">
  <div class="overflow-auto pb-16">
    <div class="grid grid-cols-1 lg:grid-cols-4 gap-4 p-4">
      <app-onboarding-palce-card
        *ngFor="let item of foodPlaces"
        [place]="item"
        class="flex flex-col h-full"
      ></app-onboarding-palce-card>
    </div>
  </div>
  <div class="fixed bottom-0 left-0 right-0 px-8 py-4 w-full">
    <button
      [disabled]="isButtonDisabled"
      class="bg-primary text-white disabled:text-black">
      Continue
    </button>
  </div>
</div>
```

5 Функціонал додатку

5.1 Перший вхід. Реєстрація. Авторизація

Додаток вимагає, щоб користувачі входили в систему, перш ніж взаємодіяти з його функціями, оскільки система рекомендацій персоналізує пропозиції на основі індивідуальних профілів та уподобань користувачів. Наразі додаток підтримує автентифікацію виключно через Google Sign-In, забезпечуючи спрощену та зручну процедуру входу. Цей метод не тільки зручний для користувачів, оскільки дозволяє використовувати існуючий обліковий запис Google, але й підвищує безпеку та зменшує потребу в додатковому управлінні паролями. Початковий вхід або реєстрація - це одноразовий процес, який надає доступ до персоналізованих функцій додатку, гарантуючи, що рекомендації будуть адаптовані та релевантні для кожного користувача. Після успішної реєстрації, дані про користувача записуються в систему та користувач перенаправляється на сторінку вибору закладів до вподоби (див п. 5.2).

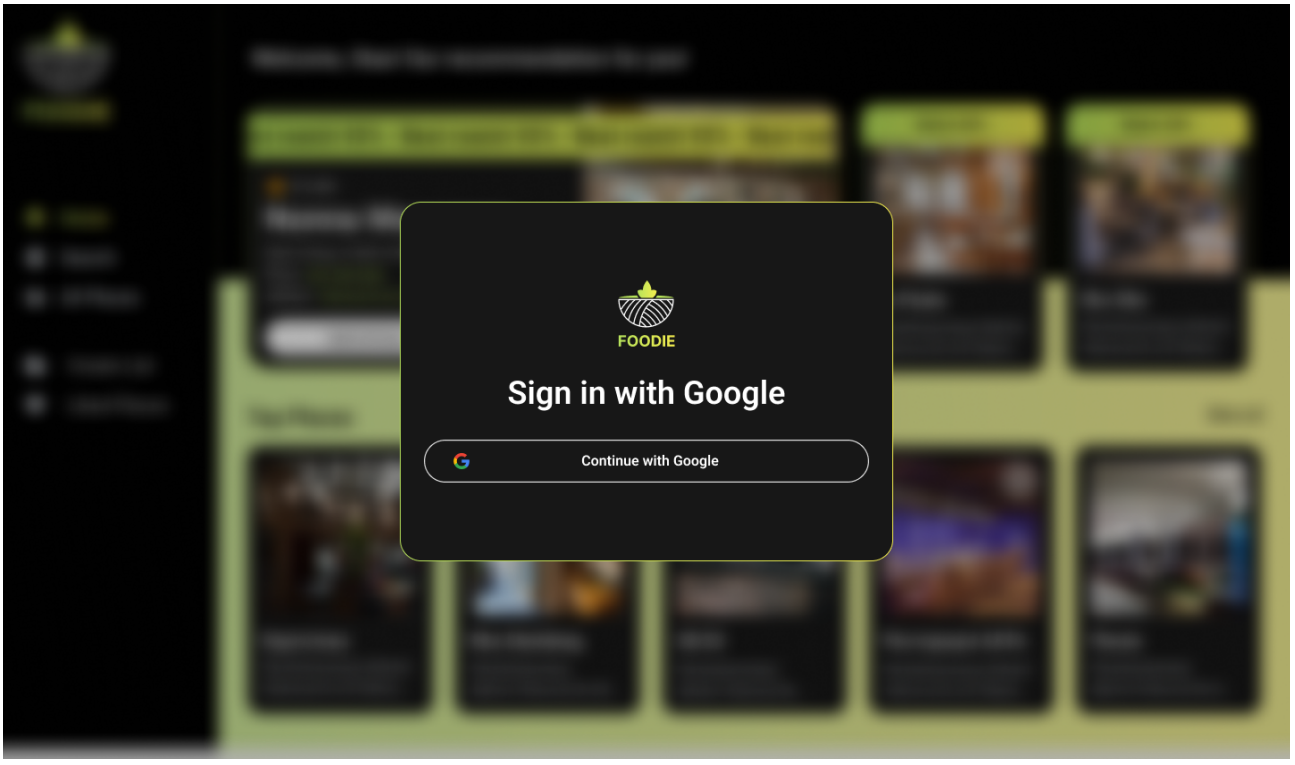

Рисунок 10. Сторінка авторизації

5.2 Сторінка початкового вибору закладів

Початковий процес збору уподобань - це важливий етап, на якому користувачам пропонується вибрати щонайменше три ресторани, які вони відвідували і які їм сподобалися. Цей вибір є не просто вказівкою на вподобання, він слугує важливими вхідними даними для моделі машинного навчання. Інформація, зібрана під час реєстрації, використовується для адаптації алгоритму рекомендацій до унікальних смаків користувача та його історії відвідування ресторанів, забезпечуючи базовий набір даних, на основі якого можна генерувати персоналізовані пропозиції ресторанів. Цей процес гарантує, що перший досвід користувача з додатком буде релевантним і цікавим, створюючи підґрунтя для інтелектуальної та персоналізованої подорожі з рекомендаціями.

Під час онбордінгу користувачам пропонується добірка закладів харчування міста Чернівці, відсортованих за популярністю. Це відображення покликане допомогти користувачам пригадати та обрати свої улюблені ресторани. Кожен заклад містить назву, адресу та візуальне зображення ресторану.

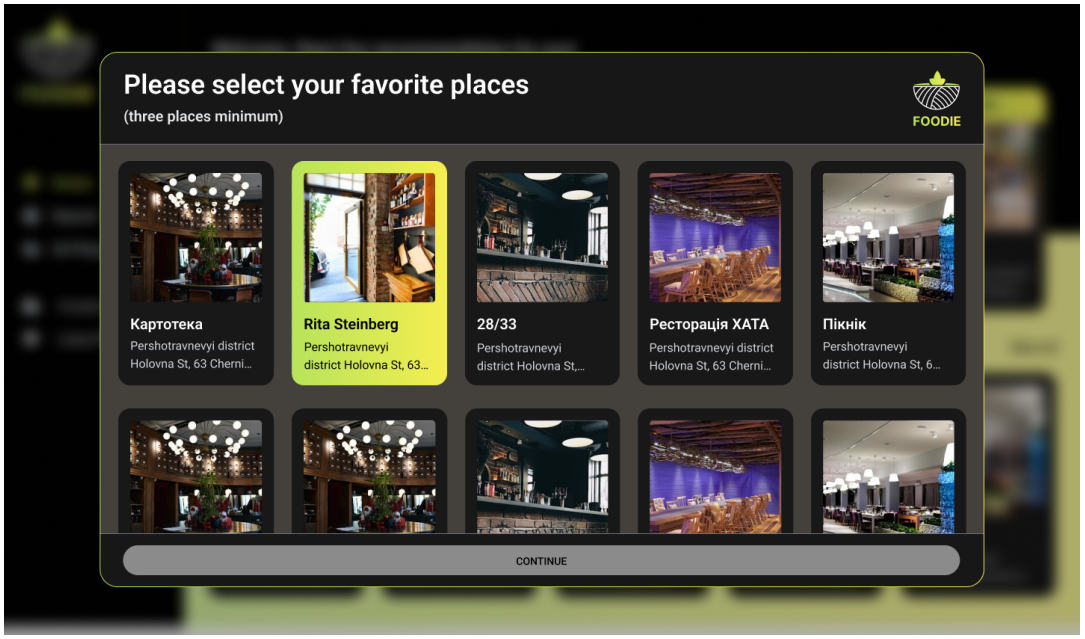

Рисунок 11. Сторінка початкового вибору закладів

5.3 Сторінка перегляду рекомендацій

Сторінка з рекомендаціями слугує центром персоналізованого підходу до вибору ресторану, влучно використовуючи алгоритми машинного навчання, щоб задовольнити індивідуальні смаки та уподобання. Ця сторінка розділена на окремі розділи, кожен з яких ретельно розроблений, щоб підвищити залученість користувачів і спростити процес прийняття рішень при виборі ресторану.

Після входу користувачів вітають по імені, що підкреслює персоналізований характер додатку. Центральним елементом сторінки є найбільш рекомендований ресторан, який демонструється на видному місці з відсотком збігів. Ця рекомендація є втіленням прогностичних здібностей додатку, обрана за допомогою розробленої моделі машинного навчання, яка аналізує вподобання користувача та безліч інших даних, щоб з високою достовірністю передбачити потенційну задоволеність користувача від відвідування ресторану.

Під основною рекомендацією на сторінці відображаються три найкращі ресторани, унікально підібрані відповідно до смакового профілю користувача. Кожен ресторан супроводжується відсотком збігу, що відображає ступінь відповідності вподобанням користувача. Макет чистий і привабливий, з високоякісними зображеннями закладів. Користувачі можуть взаємодіяти з кожною рекомендацією, отримуючи доступ до детального перегляду.

У нижній частині інформаційної панелі є розділ "Найкращі місця", в якому ресторани ранжуються за їхньою популярністю серед усіх користувачів додатку. Цей сегмент надає ширшу перспективу, дозволяючи користувачам вийти за межі своїх персоналізованих рекомендацій і побачити, куди стікається колективний смак інших людей. Показники популярності можуть бути отримані з комбінації користувацьких оцінок, частоти рекомендацій та загальної взаємодії користувачів з профілями ресторанів.

Сторінка з рекомендаціями є прикладом синергії між дизайном, орієнтованим на користувача, і передовою аналітикою даних, пропонуючи

переконливий, персоналізований досвід, який резонує з окремими користувачами, водночас надаючи вікно в ресторанні вподобання ширшої спільноти.

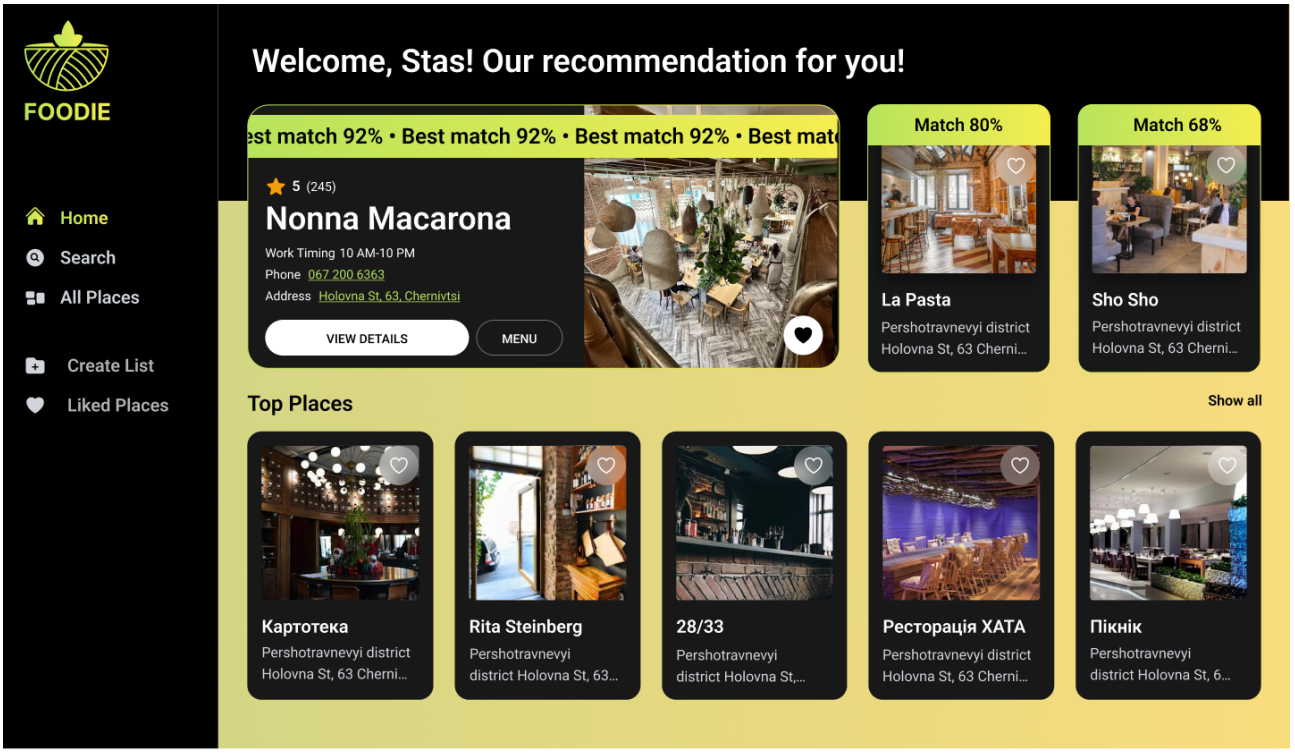

Рисунок 12. Сторінка рекомендацій

Презентація кожного ресторану у вигляді карток на сторінці панелі рекомендацій містить необхідну інформацію для користувача, а саме:

- Назва ресторану: Ідентичність кожного закладу відображається на видному місці, закріплюючи кожну рекомендацію чітким ярликом для легкого пошуку.
- Візуальна привабливість через фотографію: Фотографічне зображення дає змогу зазирнути в атмосферу та кулінарну пропозицію ресторану, підвищуючи чуттєве очікування користувача.
- Інформація про місцезнаходження: Адреса кожного ресторану чітко вказана, що дозволяє користувачам оцінити близькість і відповідно спланувати свій візит.

- Індикатори довіри громади: Рейтинг, отриманий безпосередньо з Google Maps, є надійним показником довіри, що відображає колективну думку відвідувачів ресторану.
- Інформація про роботу: Інформація про графік роботи інформує користувачів про години роботи, що є критично важливим для планування візитів і розуміння доступності ресторану.
- Прямий контакт: Надання номера телефону забезпечує прямий зв'язок із закладом, полегшуючи бронювання та запити.
- Підтвердження адреси: Повторна згадка адреси гарантує відсутність плутанини щодо місця розташування ресторану, що особливо корисно для швидкого пошуку або при плануванні логістики.
- Відсоток збігу: Можливо, найбільш важливим є те, що відсоток збігу рекомендацій - прямий результат роботи моделі машинного навчання - кількісно оцінює відповідність ресторану унікальним уподобанням користувача, діючи як персоналізований путівник у процесі прийняття рішень.

Цей набір даних не лише інформує, але й залучає користувачів, дозволяючи їм робити усвідомлений вибір ресторанів з упевненістю та зручністю. Він являє собою гармонійне поєднання якісних і кількісних даних, спрямованих на покращення кулінарних досліджень користувачів.

Висновки

Під час виконання кваліфікаційної роботи для досягнення мети опрацьовано такі завдання:

- Проведено ретельне дослідження сучасних методів та підходів у галузі машинного навчання та розробки веб-додатків у сфері систем рекомендацій;
- Засвоєно теоретичні та практичні навички у веб розробці;
- Набуто теоретичні та практичні навички в галузі машинного навчання, що забезпечило міцний фундамент для проекту;
- Зібрано вичерпний набір даних, необхідний для навчання моделі машинного навчання;
- Впроваджено алгоритм машинного навчання орієнтованого на систему рекомендацій;
- Розроблено зручний веб-додаток, який легко інтегрує навчену модель машинного навчання, надаючи користувачам інтуїтивно зрозумілий інтерфейс для отримання персоналізованих пропозицій щодо ресторанів;
- апробовано декількома знайомими, та виправлено помилки виявлені під час апробації.

Створений веб-додаток не лише втілює конвергенцію машинного навчання та навичок веб-розробки, але й є значним кроком вперед у застосуванні штучного інтелекту в повсякденних процесах прийняття рішень, особливо в ресторанній та кулінарній індустріях. Потенціал системи для модернізації та розширення відкриває шляхи для подальших досліджень і розробок, залишаючи можливість для постійного вдосконалення точності та зручності використання системи рекомендацій закладів.

Список використаної літератури

1. The State of Fake Online Reviews Statistics And Trends [Електронний ресурс] : стаття – Режим доступу : https://www.businessdit.com/fake-reviews-statistics/
2. Веб Застосунок для пошуку закладів Google Maps [Електронний ресурс] : веб застосунок. – Режим доступу : https://www.google.com/maps
3. Веб Застосунок для пошуку закладів Foursquare [Електронний ресурс] : веб застосунок. – Режим доступу : https://foursquare.com/city-guide
4. Веб Застосунок для пошуку закладів Tripadvisor [Електронний ресурс] : веб застосунок. – Режим доступу : https://www.tripadvisor.com/
5. Веб Застосунок для пошуку закладів Restaurant [Електронний ресурс] : веб застосунок. – Режим доступу : https://www.restaurant.com/
6. Огляд методу найменших квадратів [Електронний ресурс] : інтернет ресурс. – Режим доступу : https://sophwats.github.io/2018-04-05-gentle-als.html
7. Документація мови програмування TypeScript [Електронний ресурс] : онлайн документація. – Режим доступу : https://www.typescriptlang.org/
8. Документація фреймворку Angular [Електронний ресурс] : онлайн документація. – Режим доступу : https://angular.io/guide/lifecycle-hooks
9. Документація SQL Server [Електронний ресурс] : онлайн документація. – Режим доступу : https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16

Додатки

Додаток А – Скрапінг інформації про один заклад з Google Maps

```
def extract_data(input_str, link):
    data = parse(input_str)
    categories = get_categories(data)
    place_id = get_place_id(data)
    order_online_links = get_order_online_link(data)
    thumbnail = get_thumbnail(data)
    coordinates = get_gps_coordinates(data)
    images = get_images(data)
    description = get_description(data)
    status = get_open_state(data)
    plus_code = get_plus_code(data)
    reservations = get_reservations(data)
    menu = get_menu(data)
    owner = get_owner(data)
    time_zone = get_time_zone(data)
    complete_address = get_complete_address(data)
    reviews_link = get_reviews_link(data)
    if reviews_link is None:
        gl = complete_address['country_code']
        hl = get_hl_from_link(link)
        query = extract_business_name(link)
        reviews_link = generate_google_reviews_url(place_id,query , 0, hl, gl)
        pass

    price_range = get_price_range(data)
```

```
    reviews_per_rating = get_reviews_per_rating(data)
    cid = get_cid(data)
    data_id = get_data_id(data)
    about = get_about(data)
    title = get_title(data)

    hours = get_hours(data)
    if hours:
        hours = reorder_hours_list(hours)
    else:
        hours = []

    rating = get_rating(data)
    reviews = get_reviews(data)
    phone = get_phone(data)
    address = get_address(data)
    website = get_website(data)
    main_category = get_main_category(data)
    user_reviews = get_user_reviews(data)

    review_keywords = get_review_keywords(data)
```

Додаток Б – Обробка датасету відгуків

```
# %%
import pandas as pd

# %%
```

```
df = pd.read_csv("../google-reviews-scraper/data/newest_gm_reviews.csv", dtype={ 'id_review': str, 'caption': str, 'relative_date':str, 'retrieval_date': str, 'rating': float, 'username': str, 'n_review_user': str, 'n_photo_user': str, 'url_user': str, 'url_source': str })


# %%

from urllib.parse import urlparse, urlencode, parse_qs, urlunparse


def remove_query_parameter(url, query_parameter):
    # Parse the URL
    parsed_url = urlparse(url)

    # Parse the query parameters
    query_params = parse_qs(parsed_url.query)

    # Remove the 'rclk' parameter
    if query_parameter in query_params:
        del query_params[query_parameter]

    # Construct the modified URL
    modified_url = urlunparse(
                        (parsed_url.scheme, parsed_url.netloc, parsed_url.path, parsed_url.params, urlencode(query_params, doseq=True), parsed_url.fragment)
    )

    return modified_url


# %%

import urllib.parse
```

```
df["url_source"] = df["url_source"].apply(lambda u: remove_query_parameter(u, 'authuser'))

df["url_source"] = df["url_source"].apply(lambda u: remove_query_parameter(u, 'hl'))

df["url_source"] = df["url_source"].str.replace("https://www.google.com/maps/place/", "")

df["url_source"] = df["url_source"].apply(lambda u: urllib.parse.unquote(u))

df["url_source"]


# %%

df["url_user"] = df["url_user"].str.replace("https://www.google.com/maps/contrib/", "", regex=False)

df["url_user"] = df["url_user"].str.replace("/reviews?hl=en", "", regex=False)

df["url_user"]


# %%

import maya

def relative_date_to_exact(relative_date):

    return maya.when(relative_date).datetime().strftime("%Y-%m-%d")


# %%

df["user_id"] = df["url_user"].factorize()[0]

df["place_id"] = df["url_source"].factorize()[0]

# df["date"] = df["relative_date"].apply(relative_date_to_exact)

df["date"] = df["relative_date"]


# %%

final_df = df[['user_id', 'place_id', 'rating', 'date', 'url_source', 'url_user']]

final_df['rating'] = final_df['rating'].astype(int)


# %%
```

```
final_df.to_csv('ratings.csv', index=False)
```