Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мультимедійних систем

Кваліфікаційна робота

Освітній ступінь – бакалавр

На тему: **«Розробка веб-платформи для підбору та контролю раціону харчування та фізичних навантажень»**

Виконала студентка 4-го року навчання

Спеціальності «Комп’ютерні науки»

Романенко Марія Олегівна

Керівник Борозенний С. О.

Старший викладач

Київ 2023

Календарний план виконання роботи

№	Назва етапу	Термін виконання	Примітка
1.	Отримання завдання на кваліфікаційну роботу	Вересень 2022 р.	
2.	Огляд технічної літератури за темою роботи	Листопад 2022 р.	
3.	Вибір інструментів для виконання роботи	Грудень 2022 р.	
4.	Розробка веб-застосунку	Січень – лютий 2023 р.	
5.	Написання текстової частини	Березень 2023р.	
6.	Надання роботи керівнику для перевірки	Квітень 2023р.	
7.	Внесення правок згідно зауважень керівника	Квітень 2023р.	
8.	Створення презентації для захисту	Травень 2023р.	
9.	Корегування роботи після попереднього захисту	Травень 2023р.	
10.	Захист кваліфікаційної роботи	30 травня 2023 р.	

Зміст

Анотація

Дана кваліфікаційна робота присвячена розробці веб-платформи для контролю раціону харчування та фізичних навантажень з використанням фреймворку Spring Boot та створенні клієнтського інтерфейсу до нього за допомогою React. А також, порівняльному аналізу різних типів рекомендаційних систем разом з підходами до їх реалізації та створення рекомендаційної системи як частини веб-додатку, для здійснення користувачам рекомендацій щодо раціону харчування згідно їхніх фізичних характеристик.

Вступ

Розробка веб-додатків являється невід’ємною складовою сучасного цифрового світу. Не зважаючи на велике різноманіття мобільних застосунків, веб-додатки володіють досить великою кількістю переваг та являються могутнім інструментом для підприємств, організацій та індивідуальних розробників, щоб надати широкий спектр послуг та зручний інтерфейс для користувачів.

Головними перевагами веб-додатків є :

- Кросплатформеність
- Простота впровадження оновлень
- Більша доступність
- Знижені вимоги до пристрою.

Розробка веб-додатку вимагає комплексного підходу, який включає в себе аналіз вимог, проектування інтерфейсу, програмування, тестування та впровадження.

Одним з найбільш популярних інструментів для створення веб-додатків являється Spring Boot. Spring Boot – це потужний фреймворк на базі мови

програмування Java, який надає широкий спектр інструментів та бібліотек для швидкої та ефективної розробки веб-додатків. Він надає стандартизовану конфігурацію та автоматичне управління залежностями, що дозволяє розробникам зосередитися на бізнес-логіці свого додатку, замість витрачання часу на налаштування середовища розробки. Spring Boot також пропонує розширені можливості для роботи з базами даних, обробки запитів та створення API. Він дозволяє швидко створювати веб-додатки з підтримкою різноманітних веб-технологій.

У контексті створення веб-додатків одним із найбільш популярних інструментів для розробки клієнтського інтерфейсу є React. React це бібліотека JavaScript з відкритим вихідним кодом. Його основна концепція полягає в розбитті користувацького інтерфейсу на компоненти, які можна повторно використовувати та легко управляти.

Тож, об’єктом дослідження даної кваліфікаційні роботи є створення веб-додатків та рекомендаційних систем.

Предметом дослідження – технології для створення веб-додатків, доступні в межах фреймворку Spring Boot, а також різні типи рекомендаційних систем та підходи до їх реалізації.

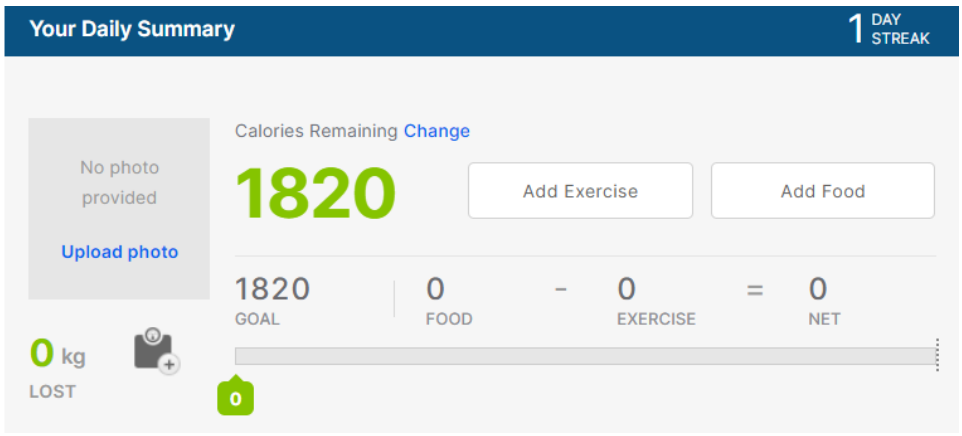
Результатом роботи стане створена веб-платформа, на якій користувачам надаватиметься можливість вести облік витрачених та спожитих калорій, слідкувати за збалансованістю свого харчування. А також користуватися послугами рекомендаційної системи, котра буде пропонувати раціон харчування, що відповідає особистій нормі калорій.

Розділ 1. Аналіз та порівняння існуючих платформ.

1.1 Огляд доступних для використання аналогічних платформ.

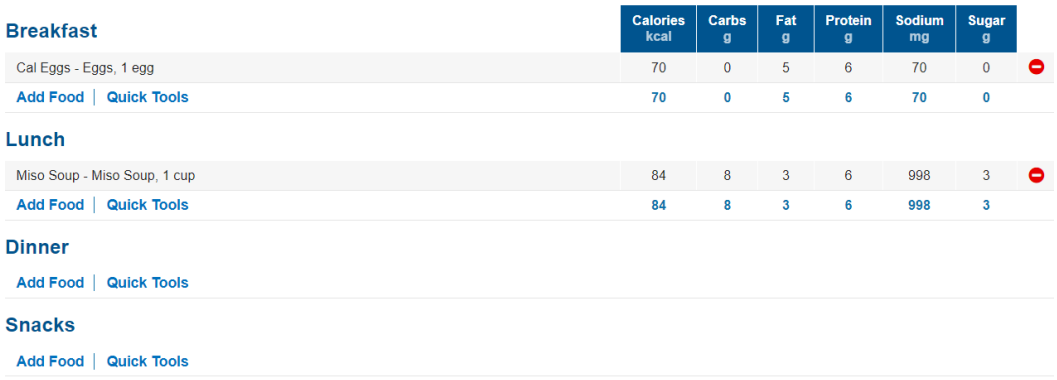
Оскільки тема здорового та збалансованого харчування є досить актуальною в наш час, знайти платформу з аналогічним призначенням досить легко. Існують варіанти як мобільних додатків, котрі можна знайти в Google Play Market або Apple App Store, так і велике різноманіття веб-додатків. Пропоновані аналоги відрізняються кількістю доступного функціоналу, зручністю інтерфейсу та вартістю доступу. Багато з додатків мають безкоштовну версію, проте в ній доступний лише обмежений функціонал.

1) MyFitnessPal – веб сайт для відстеження стану здоров'я та фізичної форми, контролю спожитих та витрачених калорій. Також доступний додаток для завантаження на смартфон. На сайті пропонується досить широкий перелік можливостей, проте усі вони доступні лише при придбанні підписки.



Daily Nutrition Goals	EDIT
Calories	1820
Carbohydrates 228 g	50 %
Fat 61 g	30 %
Protein 91 g	20 %
Custom Daily Goals	No

Fitness	EDIT
Calories Burned / Week	0 Calories
Workouts / Week	0 workouts
Minutes/Workout	0
Exercise Calories	On

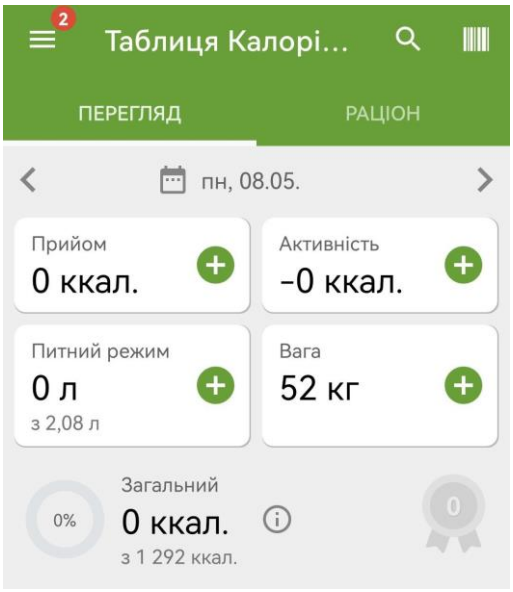


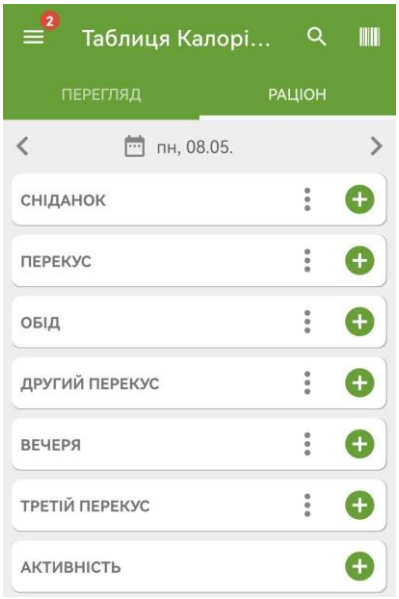
Breakfast	Calories kcal	Carbs g	Fat g	Protein g	Sodium mg	Sugar g
Cal Eggs - Eggs, 1 egg	70	0	5	6	70	0
Add Food | Quick Tools	70	0	5	6	70	0
Lunch						
Miso Soup - Miso Soup, 1 cup	84	8	3	6	998	3
Add Food | Quick Tools	84	8	3	6	998	3
Dinner						
Add Food | Quick Tools						
Snacks						
Add Food | Quick Tools						

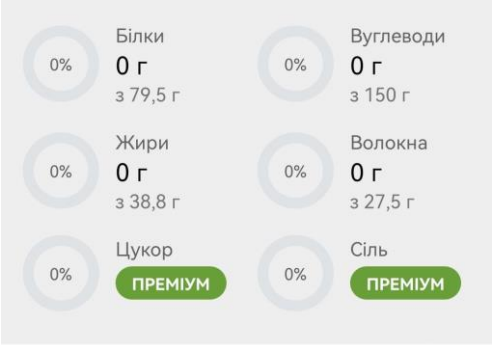
На скріншотах інтерфейсу даного веб-додатку, можна помітити, що користувачу доступні можливості відстеження співвідношення поживних речовин у стравах, котрі вживалися. Також можливий трекінг витрачених калорій під час фізичних навантажень. Також, розробники додали можливість запрошувати друзів та спільно слідкувати за своїм прогресом, що може стати хорошою мотивацією для дотримання здорового харчування на постійній основі. Інтерфейс додатку зручний та зрозумілий, усі статистичні дані відображаються у вигляді таблиць та діаграм, що значно полегшує сприйняття.

2) Таблиця калорійності – калорії

Цей мобільний застосунок доступний для завантаження в Google Play Market. Виробник зазначає, що головною функцією цього застосунку є підрахунок харчової цінності продуктів. Користувачі додатку мають можливість зберігати страви зі свого раціону, відстежувати статистику щодо кількості спожитих та витрачених калорій та налаштовувати свій раціон залежно від особистих фізичних характеристик та цілей. Додаток має зрозумілий інтерфейс, без зайвих елементів. Проте, безкоштовна підписка не відкриває доступу до всіх можливостей додатку. За бажанням, можна придбати преміум версію.







1.2. Постановка задачі

Розробити веб-застосунок, котрий поєднуватиме усі корисні функції наведених вище аналогів. Створити максимально зручний для користувача інтерфейс, без зайвих елементів. Також, доповнити платформу додатковим функціоналом, таким як рекомендаційна система, котра буде пропонувати користувачеві збалансований раціон харчування, залежно від його вподобань та фізичних характеристик.

Висновки

Порівнюючи наведені додатки не було виявлено чітких недоліків або переваг, адже вони мають однакове призначення та простір для впровадження додаткового функціоналу не особливо великий. Проте, в обох наведених прикладах був відсутній елемент рекомендаційної системи, котрий може зробити використання додатку значно приємнішим, адже підбір різноманітного та одночасно збалансованого раціону часто стає нелегким завданням для користувачів. Саме тому, було вирішено розробити веб-платформу, котра поєднуватиме усі корисні функції наведених аналогів, а також матиме систему рекомендацій для підбору страв.

Варто зазначити, що мобільні додатки можуть видатися більш зручними, за рахунок того що телефон майже завжди знаходиться під рукою та користування додатком не вимагає великих витрат часу. Проте, беззаперечною перевагою веб-додатку є його доступність з будь якого пристрою з веб-браузером та підключенням до інтернету. Також веб-додатки не вимагають жодних дій від користувача для отримання нових або ж виправлених функцій у разі оновлення застосунку виробниками.

Розділ 2. Архітектура веб-додатків

2.1. Монолітна архітектура

Монолітна архітектура вважається базовою та традиційною. Всі компоненти додатку розробленого на базі монолітної архітектури тісно пов’язані один з одним та працюють як одне ціле. Відповідно, для роботи програмного забезпечення усі його компоненти мають працювати коректно та компілюватися. Більш того, якщо виникає потреба змінити або оновити певний компонент програми, усі інші елементи мають бути перекомпільовані та протестовані. Тож, відразу можна перейти то недоліків монолітної архітектури:

- Процес внесення змін до коду програмного забезпечення або оновлення деяких компонентів займає досить багато часу та обмежує гнучкість та швидкість команди розробників.
- Некоректна робота одного з модулів може вплинути на роботу усієї програми.
- Внесення мінімальних змін до програми вимагає повторного розгортання цілого моноліту.
- Відсутність або обмежені можливості масштабування окремих компонентів.

Але існують і деякі переваги монолітних архітектур, саме тому все ще велика кількість програм розробляється з використанням цієї парадигми:

- Більш проста розробка. Оскільки весь застосунок знаходиться в єдиній кодовій базі, розробникам легше розуміти систему в цілому.
- Спрощений процес розгортання.
- Більш швидке та просте тестування.

Монолітна архітектура може бути застосована при розробці невеликих програм, або командами з обмеженими ресурсами. Однак для більших та

складніших додатків краще робити вибір на користь модульної або мікросервісної архітектури, адже це надасть кращу маштабованість, гнучкість та надійність.

2.2 Мікросервісна архітектура

Цей підхід полягає у розробці веб-застосунків, як сукупності декількох невеликих, незалежних один від одного сервісів, кожен з яких виконує певні функції. Мікросервісна архітектура дозволяє команді розробників ізолювати сервіси та працювати окремо над кожним з них, таким чином використовуючи децентралізований підхід до розробки застосунків. Кожен сервіс відповідає за виконання конкретного завдання та комунікує з іншими за допомогою простих API. Серед перевах мікросервісної архітектури можна назвати наступні:

- Кожен мікросервіс може бути змінений, оновлений або масштабований без впливу на інші частини застосунку, адже є повністю незалежною частиною.
- Надійність та стійкість, оскільки вихід із ладу одного з сервісів не загрожуватиме решті системи.
- Можливість обирати різноманітні мови програмування, засоби розробки та технології для різних сервісів.
- Швидке масштабування.
- Можливість перевикористання окремих сервісів в інших проектах.

Попри безліч переваг мікросервісної архітектури, слід пам’ятати також про можливі недоліки:

- Розробка мікросервісного застосунку є досить складним процесом, котрий вимагає вправного управління командою розробників, хорошого координування та комунікації між членами команди.

- Складнощі у налагодженні та тестуванні, спричинені тим, що кожний сервіс має окремий механізм логування.
- Додаткові потреби у захисті та забезпеченні інформаційної безпеки веб-застосунку.

Висновки

Перш ніж обрати будь-який архітектурний підхід для розробки проекту, необхідно ретельно розглянути його вимоги та плани майбутнього розвитку, щоб уникнути проблем з обслуговуванням в міру зростання програми.

Оскільки проект, розроблений в межах виконання даної кваліфікаційної роботи, характеризується відносно простими вимогами та невеликим розміром було вирішено розробляти його на базі монолітної архітектури. Це забезпечило простіший процес розробки, та допомогло зосередитися на створенні програми в цілому без накладних витрат на керування окремими службами або мікросервісами.

Розділ 3. Вибір інструментів та технологій для розробки

3.1. Back-end

3.1.1. Spring Boot

Для розробки серверної частини застосунку було вирішено використовувати Spring Boot. Spring boot – це фреймворк з відкритим вихідним кодом на основі мови програмування Java, орієнтований на створення мікросервісних застосунків. Його розробниками є Pivotal Team та він використовується для створення автономних Spring додатків.

Головними перевагами Spring Boot є:

- Авто-конфігурація: Spring Boot здатний здійснювати автоматичну конфігурацію застосунку, базуючись на вказаних залежностях, що зводить до мінімуму кількість конфігураційних xml-файлів.
- Spring Boot стартери: це набір певних дескрипторів залежностей необхідної функціональності, котрі ви можете включити в свій додаток. До прикладу, «spring-boot-starter-web» включає в себе усі необхідні залежності для створення веб-додатку.
- Вбудовані сервери: Spring Boot містить у собі вбудовані сервери, такі як Tomcat, Jetty, Undertow, котрі не вимагають додаткового встановлення для використання. Таким чином, доступна можливість створювати автономні застосунки без потреби розгортання на окремому сервері. [1]

Розглянувши деякі переваги використання Spring Boot для розробки застосунків, важливо зазначити як саме вони працюють. Так званою точкою входу у застосунок є анотація @SpringBootApplication. Вона розташована в класі, котрий містить метод main. Ця анотація поєднує у собі наступні три анотації:

- @Configuration позначає клас котрий є джерелом визначень компонентів (beans), або містить пов’язані з бінами методи конфігурації
- @EnableAutoConfiguration надає дозвіл здійснювати автоматичну конфігурацію на основі заданих залежностей
- @ComponentScan вказує на компоненти, конфігурації та сервіси котрі необхідно сканувати під час ініціалізації.

3.1.2 Spring Security

Питання інформаційної безпеки займає ледь не найважливіше місце при розробці веб-додатку. Адже користувач довіряє застосунку свої особисті дані і вони обов’язково мають бути під надійним захистом. Основною ціллю заходів безпеки у застосунках є запобігання крадіжок або зміни інформації та вихідних кодів.

Аутентифікація – це процес перевірки того, хто є користувачем. Вона стосується підтвердження ваших облікових даних, якими зазвичай являються ім’я користувача та пароль. Залежно від рівня безпеки застосунку можна виділити декілька факторів автентифікації:

- Базова автентифікація: є найпростішою та вимагає від користувача лише один пароль для підтвердження його особистості.
- Двофакторна автентифікація: відбувається у два кроки. Для підтвердження особистості, користувачу необхідний не лише пароль, а й деяка інформація, котру знає лише він.
- Багатофакторна автентифікація: найбільш надійний метод перевірки особистості, котрий містить у собі два або більше рівнів безпеки, які є незалежними один від одного і тому забезпечують максимально надійний захист даних.[2]

Авторизація – це процес перевірки того, до чого користувач має доступ. Зазвичай вона відбувається після успішної аутентифікації. Поширеною

практикою є надання користувачам застосунку різних прав доступу до ресурсів. До прикладу, адміністратор платформи може мати значно більше прав та можливостей ніж вільний користувач і саме авторизація визначає до якої інформації отримає доступ користувач згідно політики застосунку.[3]

Для захисту своєї веб-платформи я вирішила використовувати фреймворк під назвою Spring Security. Це стандартний фреймворк для аутентифікації та авторизації.

Підключення Spring Security до проекту можливе за допомогою стартеру org.springframework.boot:spring-boot-starter-security. Щоб зрозуміти принцип роботи Spring Security, необхідно розглянути усі етапи безпекових заходів.

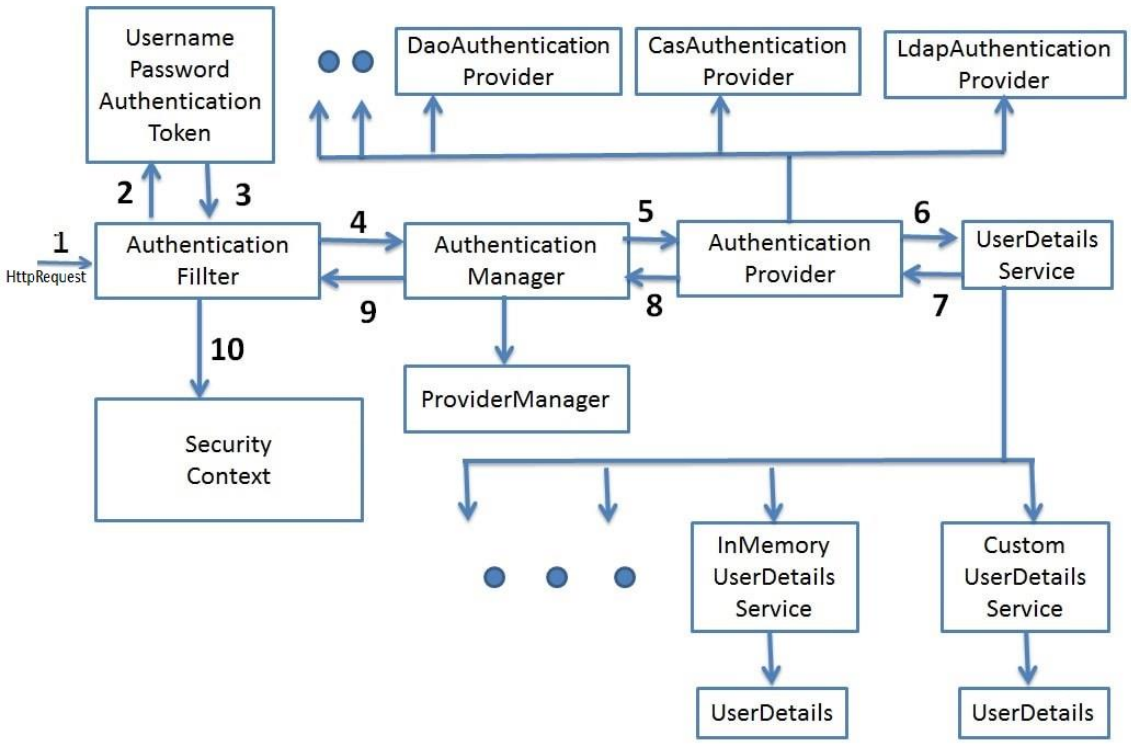


Рисунок 1.1. Архітектура Spring Security

1) В основі Spring Security лежить ланцюжок фільтрів, який відповідає за перехоплення та обробку HTTP запитів. Кожен фільтр відповідає за окреме завдання пов’язане з безпекою, наприклад автентифікацію, авторизацію, керування сеансами та захист CSRF.
2) Після того, як запит був перехоплений та оброблений відповідним фільтром, створюється об’єкт Authentication з отриманих імені користувача та пароля, котрі потім слугують для створення UsernamePasswordAuthenticationToken.
3) AuthenticationManager – це інтерфейс, котрий містить один єдиний метод authenticate, який власне і відповідає за процес аутентифікації. Проте, оскільки це лише інтерфейс, фактична реалізація методу автентифікації здійснюється завдяки ProviderManager.
4) ProviderManager містить у собі список об’єктів AuthenticationProviders, і за допомогою вище згаданого методу він викликає метод authenticate відповідного AuthenticationProvider, щоб при успішній аутентифікації отримати у відповідь об’єкт Authentication. [4]
5) UserDetailsService – важлива частина Spring Security, котра являє собою інтерфейс з лише одним методом:

```
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException;
```

Його метою є отримання інформації про користувача з бази даних або будь-якого іншого джерела інформації, необхідної для здійснення автентифікації та авторизації.

Для передачі даних для автентифікації в моєму застосунку використовується JSON Web Token або JWT. JWT – це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519.[5]

JWT складається з трьох частин:

- Заголовок (Header) – містить метадані про тип токена та криптографічний алгоритм, котрий використовується для підпису токена.
- Корисне навантаження (payload) – містить фактичні дані для передачі за допомогою маркера. Ці дані також називають «претензії» (claims). Вони можуть містити таку інформацію, як ідентифікаційні дані користувача, ролі, дозволи, термін дії тощо.
- Підпис (signature) – потрібен для перевірки цілісності токена та гарантії того, що він не підроблений. Підпис створюється шляхом кодування заголовку та корисного навантаження з подальшим підписанням їх за допомогою секретного ключа.

Коли користувач проходить процес автентифікації, JWT зазвичай генерується сервером та надсилається клієнту, де потім зберігається в локальному сховищі (local storage) або сховищі сеансів (session storage).

3.1.3 Spring Data JPA

Spring Data JPA є частиною Spring Framework, яка забезпечує зручний спосіб взаємодії з реляційними базами даних за допомогою Java Persistence API (JPA). JPA реалізує концепцію ORM. Існує декілька реалізацій цього інтерфейсу, найбільш популярною являється Hibernate.

Ключовими аспектами Spring Data JPA є:

- Можливість використання анотацій для відображення об’єктів Java у таблиці бази даних. Прикладами таких анотацій є @Entity, @Table, @Column.
- Інтерфейс JPA Repository, котрий містить набір базових CRUD операцій а також надає можливість надсилати деякі запити до бази даних, дотримуючись певних умов іменування, уникаючи явного написання запитів мовою SQL.

- Підтримка JPQL (Java Persistence Query Language) для написання запитів, а також можливість написання звичайних SQL запитів.
- Для створення складних запитів із динамічними предикатами та умовами існують такі інструменти, як Spring Data Specification та Criteria API.
- Вбудовані інструменти для сортування та пагінації результатів запитів.

3.1.4 Вибір СКБД

Для зберігання даних свого застосунку я обрала MySQL. MySQL – це реляційна система управління базами даних з відкритим кодом, яка використовує мову структурованих запитів SQL для зберігання та управління даними. MySQL проста у встановленні та використанні, саме тому вона чудово підходить як для невеликих веб-сайтів, так і для складних корпоративних систем.

3.2. Front-end

Головним інструментом для створення клієнтського інтерфейсу мого застосунку став React. React – це бібліотека JavaScript, розробниками якої є компанія Meta, та котра активно використовується для створення односторінкових застосунків.

Головними особливостями React є:

- Компонентна структура: архітектура React основана на компонентах, що дозволяє розбити користувацький інтерфейс на окремі незалежні частини. Компоненти можна комбінувати та повторно використовувати, зберігаючи код організованим і простим. Цей підхід є дуже зручним, адже можна з легкістю додавати, оновлювати або видаляти компоненти, не зачіпаючи при цьому інші елементи застосунку.

- Віртуальний DOM (Document Object Modal): React використовує віртуальний DOM, що дозволяє ефективно оновлювати необхідні елементи сторінки без безпосередньої взаємодії з реальними подіями DOM. Бібліотека ReactDOM забезпечує зберігання віртуального DOM в пам’яті та його синхронізацію з реальним DOM. Ця особливість забезпечує високу продуктивність додатку.
- JSX (JavaScript XML): це розширення синтаксису JavaScript, яке дозволяє писати HTML код безпосередньо в JavaScript. Це дозволяє описувати структуру та вигляд компонентів у декларативний спосіб. Також JSX дозволяє вбудовувати JavaScript вирази в код у фігурних дужках. Це дає змогу доступатися до змінних та виконувати задану логіку JavaScript безпосередньо в JSX.
- Властивості (props): властивостями в React називають дані, що передаються в компоненти з батьківських компонентів. Це полегшує процес перевикористання компонентів.
- Стани (states): кожен компонент може мати власні стани. Станом називають об’єкт, котрий зберігає певну інформацію і ця інформація може змінюватися протягом життєвого циклу компонента. Для управління станами існують спеціальні функції, напряму змінювати стан компонента не можливо.

Висновки

У цьому розділі були розглянуті технології, використані під час розробки платформи.

Back-end:

1) Spring boot надає широкі можливості для створення веб-додатків, спрощуючи процес розробки і розгортання.

2) Основні технології, які використовуються в Spring Boot включають Spring MVC для обробки веб-запитів, Spring Data JPA для роботи з базами даних та Spring Security для забезпечення безпеки додатків.
3) Spring Boot забезпечує автоматичну конфігурацію проектів, що дозволяє швидко почати розробку без необхідності ручного налаштування багатьох компонентів.

Front-end:

1) Використання React дозволяє створювати ефективні та динамічні веб-інтерфейси, які реагують на зміни даних в реальному часі без перезавантаження сторінки.
2) React використовує компонентний підхід до розробки, що спрощує управління станами та полегшує повторне використання коду.
3) Однією з ключових переваг React є віртуальний DOM, який дозволяє ефективно оновлювати лише необхідні частини сторінки, забезпечуючи високу продуктивність додатків.

Розділ 4. Рекомендаційні системи

4.1 Призначення рекомендаційних систем

Рекомендаційні системи – один з найбільш популярних та затребуваних напрямів інтелектуального аналізу та машинного навчання. Алгоритми рекомендаційних систем часто використовуються на різноманітних веб-платформах, таких як інтернет-магазини, соціальні мережі, відеохостинги для того щоб підбирати персоналізовані рекомендації користувачам, базуючись на їхніх особистих вподобаннях, історії пошуку, зацікавленості у окремих продуктах тощо.

4.2 Типи рекомендаційних систем

4.2.1. Колаборативна фільтрація (collaborative filtering)

Цей алгоритм збирає інформацію та аназілує дії багатьох користувачів, намагаючись знайти схожість у їхніх діях та вподобаннях. Базуючись на попередніх діях багатьох користувачів, він передбачає найбільш ймовірні варіанти взаємодії майбутніх користувачів. Основна ідея колаборативної фільтрації полягає в тому, що якщо декілька користувачів мають схожі смаки або вподобання, то ймовірно, що їхні смаки будуть збігатися й при наступних взаємодіях з об'єктами, тому доцільно буде пропонувати таким користувачам схожі рекомендації.

Колаборативна фільтрація не потребує інформації про характеристики окремих об'єктів. Замість цього, увага акцентується на групі схожих користувачів та їхній взаємодії з об'єктами.

Розрізняють декілька технік типів колаборативної фільрації:

- Memory-based

- Model-based.

4.2.1.1 Memory-based

Даний підхід основується на використанні даних про взаємодію користувачів з об’єктами, котра зберігається в базі даних. В залежності від логіки здійснення рекомендацій він може бути розділений на дві секції:

1) User-item filtering:

Першим кроком при застосуванні цього алгоритму є визначення найбільш схожих за вподобаннями користувачів. З цією метою може бути використаний коефіцієнт кореляції Пірсона, для якого кожен користувач представляється як вектор у n-вимірному просторі, після чого відбувається обчислення відстані між активним користувачем та усіма іншими для знаходження найбільш подібних. Також популярною метрикою для обчислення подібності є косинусна подібність. До прикладу, подібність між двома користувачами u та u` може бути обрахована, як косинус кута між двома їхніми векторами:

$$sim(u,u') = cos(\theta) = \frac{r_u \cdot r_{u'}}{\|r_u\|\|r_{u'}\|} = \sum_i \frac{r_{ui} r_{u'i}}{\sqrt{\sum_i r_{ui}^2}\sqrt{\sum_i r_{u'i}^2}}$$

Оцінки користувача u` для об’єкту і можуть бути передбачені шляхом зважування суми оцінок цього фільму іншими користувачами u`.

$$\hat{r}_{ui} = \sum_{u'} sim(u,u') r_{u'i}$$

Також, оцінки мають бути нормалізовані згідно загальної кількості оцінок інших користувачів (u`):[7]

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u,u') r_{u'i}}{\sum_{u'} |sim(u,u')|}$$

2) Item-item filtering

Цей алгоритм відрізняється від попереднього тим, що в першу чергу відбувається пошук найбільш схожих елементів, на основі векторів рейтингів з попередньо отриманих оцінок користувачів. Після чого, активному користувачу здійснюються рекомендації об’єктів, з найбільшим ступенем схожості до тих, якими він вже цікавився.

Обидва типи memory-based фільтрації не використовують технік машинного навчання, усі розрахунки для створення рекомендацій здійснюються лише за допомогою арифметичних операцій та застосування метрик подібності.

4.2.1.2 Model based filtering

Наступний підхід полягає у проектування та розробці моделей з використанням алгоритмів машинного навчання, щоб передбачати оцінки користувачів на ще не оцінені об’єкти. Дані моделі працюють використовуючи інформацію про історію взаємодій представлену у матрицях, застосовуючи різноманітні алгоритми, такі як матрична факторизація, кластеризація та глибинне навчання.

- Методи матричної факторизації, такі як розкладання сингулярних значень(SVD) і чергування найменших квадратів(ALS), розкладають матрицю взаємодії користувача з елементом на матриці прихованих факторів меншої розмірності. Ці приховані фактори представляють деякі характеристики користувачів та об’єктів, що дозволяє надавати персоналізовані рекомендації.
- Алгоритми кластеризації використовуються для групування користувачів або об’єктів на невеликі кластери, на основі їх схожості, що дозволяє отримати більш таргетовані рекомендації в кожному кластері. Для вимірювання подібності можуть бути використані ті самі

метрики подібності, що й для memory-based систем. Зазвичай, кластеризація є проміжним етапом та результуючі кластери можуть бути використані для виконання наступних кроків та подальшого аналізу даних.

- Нейронні мережі можна використовувати для безпосереднього моделювання взаємодії між користувачем та елементом. Цей підхід зазвичай передбачає створення окремого рівня для представлення користувачів і елементів у вигляді щільних векторів у просторі нижньої вимірності. Ці вектори фіксують приховані фактори або особливості, які представляють уподобання користувача та характеристики предмета. Потім ці вектори передаються на наступні рівні нейронної мережі, щоб вивчати закономірності та робити прогнози. Модель навчена мінімізувати помилку передбачення між спостережуваними взаємодіями між користувачем і елементом та прогнозованими оцінками чи вподобаннями. [8]

Головною перевагою колаборативної фільтрації є те, що для неї не потрібно жодної інформації про користувачів або об’єкти. Рекомендації здійснюються лише на основі дій користувачів, що робить цей спосіб універсальним для різних типів об’єктів, таких як фільми, музика, товари тощо. До того ж, чим більше люди взаємодіють з заданими об’єктами, тим точніше та ефективніше буде працювати рекомендаційна система, особливо, якщо перелік продуктів та користувачів є фіксованим. Проте існує зворотня сторона цієї переваги – так званий, холодний старт(cold start problem). Ця проблема заключається в тому, що неможливо створити доцільну рекомендацію щойно зареєстрованому користувачеві, або ж рекомендувати щойно створений об’єкт, з котрим ще ніхто не взаємодіяв. Саме це є головним недоліком колаборативної фільтрації і для боротьби з ним розробники використовують декілька стратегій. Наприклад, випадкові рекомендації новим користувачам

або ж рекомендування нових товарів найбільш активним покупцям. Також, для нових користувачів можна використовувати інші методи рекомендаційних систем, допоки не набереться необхідна кількість взаємодій для застосування колаборативного методу.

4.2.2. Рекомендації на основі контенту (content-based filtering)

Даний тип рекомендаційних систем здійснює генерування рекомендацій на основі певного контенту, а саме різноманітних характеристиках об'єктів, ключових слів присутніх в їхньому описі тощо. Головно ідеєю контент базованих рекомендаційних систем є те, що якщо користувач вподобав деякий об'єкт, то скоріш за все він буде зацікавлений у інших об'єктах зі сходими характеристиками. Зрозуміло, що здійснення таких рекомендацій можливе лише якщо продукти мають чіткий набір характеристик і вже доступна інформація про зацікавленість користувача у певних продуктах. Поширеною практикою при створенні системи даного типу є створення профілю користувача, де зберігатиметься уся інформація про взаємодію користувача з заданими об'єктами, наприклад переходи на сторінки, виставлення оцінок та лайків, відгуки тощо. Чим більшу історію взаємодій з об'єктами матиме користувач, тим точнішими будуть рекомендації.

Існує декілька підходів до сворення контент-базованих рекомендаційних систем і далі буде описано класичні методи, котрі застосовуються найчастіше:

1) Модель векторного простору.

   Головною ідеєю цього підходу є представлення профілів об'єктів та користувачів у вигляді векторів в багатовимірному просторі. Першим кроком при реалізації цього методу є вилучення характеристик об’єктів, необхідних для генерації рекомендацій. Після цього, кожен об'єкт представляється у вигляді n-вимірного вектору, де кожен вимір відповідає окремому терміну, а n – це загальна кількість використаних термінів.

Зазвичай, в представленні ознак присутні бінарні індиктори (0 або 1) для позначення наявності або відсутності терміну. Також, можуть бути використані більш просуноті техніки, такі як TF-IDF(Term Frequency and Inverse Document Frequency).

TF(Term Frequency) вимірює, як часто термін з'являється в певному документі. Термін частоти обчислюється за формулою:

$$\text{TF} = \frac{n_i}{\sum_k n_k},$$

Де n_i є число входжень слова в документ, а в знаменнику – загальна кількість слів в документі.

Проте, в цьому процесі існує один недолік. Знаходячи терміни з великим відсотком присутності TF ігнорує важливість термінів котрі зустрічаються рідше. Це може негативно вплинути на якість результатів, адже терміни котрі зустрічаються в документах рідко можеть бути не менш важливими. Для вирішення цієї проблеми існує IDF(Inverse Document Frequency). IDF обчислюється за формулою:

$$\text{IDF} = \frac{|D|}{|d_i \supset t_i|},$$

де |D| - кількість документів колекції;

$|d_i \supset t_i|$ - кількість документів, в яких зустрічається слово t_i (коли $n_i \neq 0$

Використання IDF зменшує вагу шировживаних слів та збільшує вагу слів котрі зустрічаються рідко. Таким чином, відбувається нормалізація терміну частоти.

Наступним етапом є створення профілю користувача. Його уподобання також представляються у вигляді вектора, який будується на основі елементів, з якими користувач взаємодіяв або ставив позитивні оцінки. Цей

вектор створюється шляхом агрегації ознак елементів за допомогою створення матриць зв'язків між користувачем та елементами.

Маючи вектори об'єктів та користувача можна переходити до розрахунку подібності. Найчастіше для цього використовуються такі метрики подібності, як косинусна або Евклідова відстань. Косинус кута між вектором елемента та вектором профілю користувача надає змогу виннячити ступінь подібності та таким чином підібрати елементи для майбутніх рекомендацій.

Генерація рекомендацій відбувається шляхом ранжування елементів на основі значень їх подібності. Елементи з найвищими оцінками вважаються найбільш релевантними та обираються для рекомендацій користувачеві.

2) Метод класифікації

Також, для створення рекомендацій можуть бути використані алгоритми класифікації, такі як байєсівські класифікатори або моделі дерева рішень. До прикладу, кожен рівень дерева рішень може бути використаний для фільтрації різноманітних вподобань користувача для здійснення більш точного вибору.

Перевагами контент-базованих рекомендаційних систем є:

- Оскільки рекомендації здійснюються персонально, на основі даних користувача, цей тип рекомендаційних систем не потребує жодної інформації про інших користувачів, що забезпечує легку масштабованість для роботи з великою кількістю клієнтів.
- Щойно створені об’єкти можуть пропонуватися користувачеві, оскільки для здійснення рекомендацій потрібні лише їхні характеристики. Це зменшує прояв, так званої проблеми холодного старту(cold start problem).

Недоліки контент-базованих рекомендаційних систем:

- Оскільки системи, що базуються на контенті, зосереджені на пошуку елементів, подібних до тих, якими вже цікавився користувач, вони не здатні ознайомити користувачів з чимось новим та несподіваним.
- Якщо смаки користувача часто змінюються, системі може бути важко ефективно охоплювати усі зміни уподобань, що може спричинити генерацію менш точних рекомендацій з часом.

Висновки

У цьому розділі були розглянуті різні підходи та моделі, що застосовуються для створення ефективних рекомендацій. На підставі проведеного аналізу можна зробити наступні висновки:

- Контент-базовані системи здійснюють рекомендації завдяки аналізу характеристик та властивостей об'єктів. Вони виявляють подібність між об'єктами на основі спільних атрибутів та забезпечують рекомендації, котрі враховують особисті вподобання користувача. Системи даного типу добре працюватимуть, коли властивості об'єктів легко визначити та описати.

- Системи на основі колаборативної фільтрації: базуються на аналізі поведінки користувачів та виявленні спільних патернів між ними. Моделі колаборативної фільтрації поділяються на дві категорії. Перші використовують схожість між користувачами та об'єктами для здійснення рекомендацій, а другі використовують статистичні або машинні моделі для прогнозування рекомендацій. Колаборативні фільтри зазвичай мають високу точність, але можуть страждати від проблеми «холодного старту», коли новий користувач або об'єкт не володіють достатньою кількістю даних для здійснення рекомендацій.

Розділ 5. Створення додатку

5.1 Архітектура Back-end

Першим етапом розробки було створення Spring Boot проекту та додавання до нього необхідних залежностей. Для реалізації бізнес логіки проекту та виконання різноманітних операцій з даними були створені класи згідно стандартної архітектури Spring Boot. Стандартний застосунок створений з використанням Spring Boot зазвичай складається з чотирьох шарів:

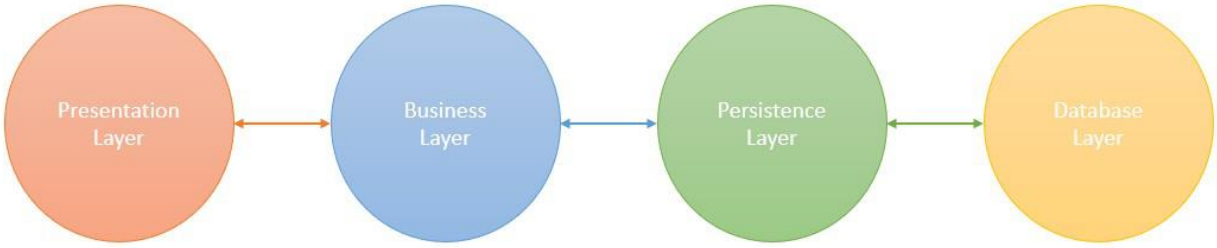


Рис. 5.1

1) Presentation Layer – верхній шар застосунку. Він обробляє HTTP запити та здійснює аутентифікацію. Також, він відповідає за конвертацію JSON об’єктів в об’єкти Java та навпаки. Після аутентифікації запитів вони передаються на наступний рівень.
2) Business Layer – в ньому міститься уся бізнес логіка застосунку. Також, він відповідальний за валідацію та авторизацію.
3) Persistence Layer – містить усю логіку, пов’язану з базою даних. Він відповідає за конвертацію Java об’єктів в рядки таблиці бази даних та навпаки.
4) Database Layer – тут міститься база даних застосунку, наприклад MySQL, MongoDB тощо. Цей рівень відповідає за здійснення CRUD (create, retrieve, update, delete) операцій. [6]

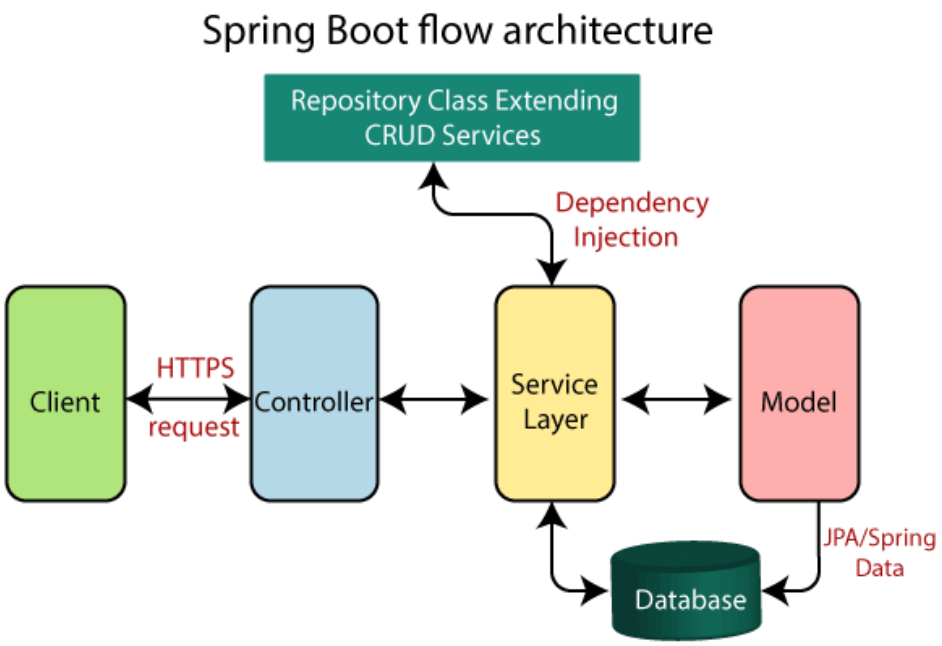


Рис. 5.2

Наведені на рисунку рівні програми містять різні біни, і для автоматичного їх виявлення, необхідно застосовувати відповідні анотації. Прикладами таких анотацій є @Service, @Repository, @Component. Також, для позначення класів, котрі відповідатимуть за обробку HTTP запитів була використана анотація @RestController.

Важливим кроком у розробці будь-якого застосунку є визначення сутностей та зв'язків між ними, котрі потім будуть спроектовані у таблиці бази даних. Кожна сутність являє собою окрему таблицю, рядками якої стануть екземпляри цієї сутності. Тож, для визначення сутностей була використана анотація @Entity, а для встановлення необхідних зв’язків анотації @OneToMany, @ManyToOne. Приклад реалізації зв’язку «один до багатьох»:

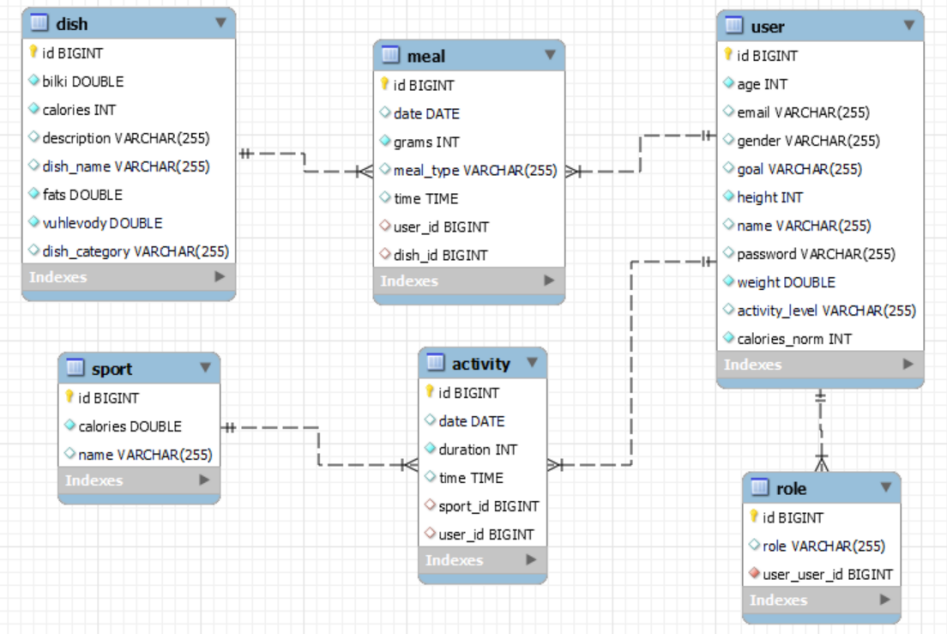
Клас User:

```
@OneToMany(mappedBy = "user")
private List <Meal> meals;
```

Клас Meal:

```
@ManyToOne
@JoinColumn(name = "user_id")
private User user;
```

В результаті була отримана така схема даних, згенерована в СКБД MySQL:



Для взаємодії з об’єктами бази даних та виконання різноманітних запитів, необхідних для реалізації бізнес-логіки проекту, були використані функції, доступні в межах Spring Data JPA. А саме,

- Можливість визначати запити за допомогою іменування методів.

```
List<Meal> findByUserAndDate(User user, LocalDate date);
List<Meal> findByUserAndDateAndMealType(User user, LocalDate
date, MealType mealType);
```

До прикладу, назви цих методів будуть проаналізовані і після цього автоматично згенеруються відповідні запити SQL для пошуку страв заданого користувача або з заданими характеристиками.

- Написання запитів мовою SQL в коді, завдяки анотації @Query:

```
@Query("SELECT m FROM Meal m " +
       "WHERE m.user = :user " +
       "AND m.date IN (" +
       "    SELECT m1.date FROM Meal m1 " +
       "    WHERE m1.user = :user " +
       "    GROUP BY m1.date " +
       "    HAVING COUNT(DISTINCT m1.mealType) = (SELECT COUNT(DISTINCT m2.mealType) FROM Meal m2 WHERE m2.user = :user)) " +
       "ORDER BY m.date")
List<Meal> findMealsForUserWithAllTypesPresent(@Param("user") User user);
```

Ось приклад такого запиту, в якому відбувається пошук списку страв, котрі входили до повноцінного денного раціону заданого користувача.

Для безпечного користування застосунку необхідно було подбати про захист особистої інформації користувачів. Для цього був використаний фреймворк Spring Security.

Короткий опис усіх компонентів Spring Security:

- Головна реалізації безпеки знаходиться в класі WebSecurityConfig, котрий є розширенням WebSecurityConfigurerAdapter. В цьому класі містяться конфігурації HttpSecurity для налаштування cors, csrf, керування сесіями а також правила доступу до захищених ресурсів.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;
    private final CustomPasswordEncoder passwordEncoder;
    private final JwtFilter jwtFilter;
```

Метод configure дозволяє налаштувати правила доступу для різних URL.

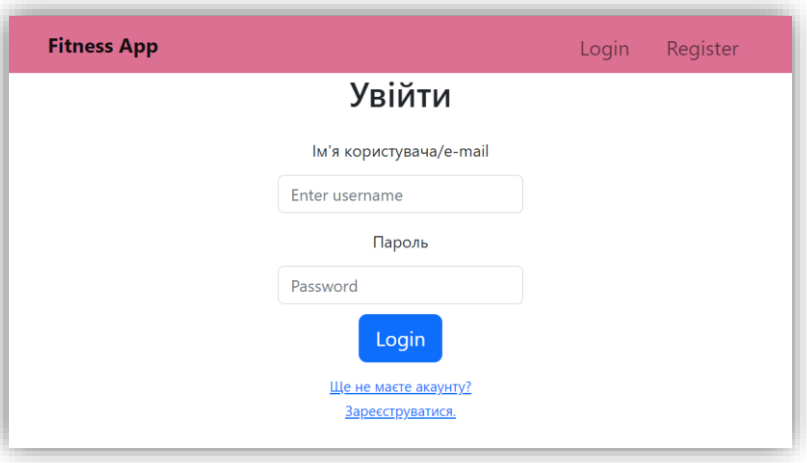
```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http = http.csrf().disable().cors().disable();

    http = http
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and();
    http = http.exceptionHandling()
            .authenticationEntryPoint((request, response, exception)->{
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, exception.getMessage());
            }).and();
     http.authorizeHttpRequests()
             .antMatchers( ...antPatterns: "/api/auth/**").permitAll()
             .anyRequest().authenticated();
     http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
}
```

- Інтерфейс UserDetailsService містить у собі метод для завантаження користувача за його ім’ям і повернення об’єкту UserDetails, який потім буде використаний для здійснення автентифікації та валідації.
- UserDetails містить необхідну інформацію для створення об’єкту автентиіфкації, таку як ім’я, пароль та роль користувача. Логіка мого застосунку потребує наявності лише однієї ролі – звичайного користувача. Усім, хто не пройшов процес автентифікації буде доступна лише сторінка для реєстрації та входу в акаунт.
- Для роботи із JWT створений клас JwtFilter, що розширяє клас OncePerRequestFilter. Вся логіка фільтру, а саме аналіз та валідація JWT, налаштування терміну його валідності, завантаження деталей користувача, тощо, відбувається в методі doFilterInternal і спрацьовує лише один раз для кожного запиту.
- Паролі користувачів зберігаються в базі даних в зашифрованому вигляді. Для цього використовується наявний у Spring Security шифрувальник паролів – BCryptPasswordEncoder.

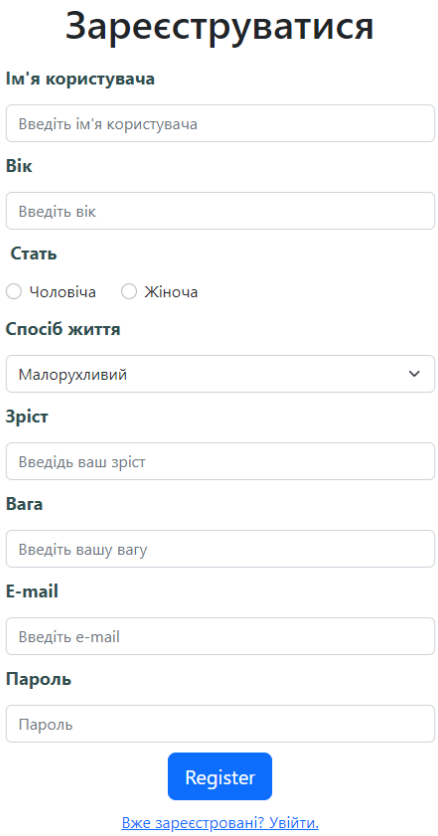
5.3 Створення клієнтського інтерфейсу

5.2.1 Перелік можливостей доступних користувачу

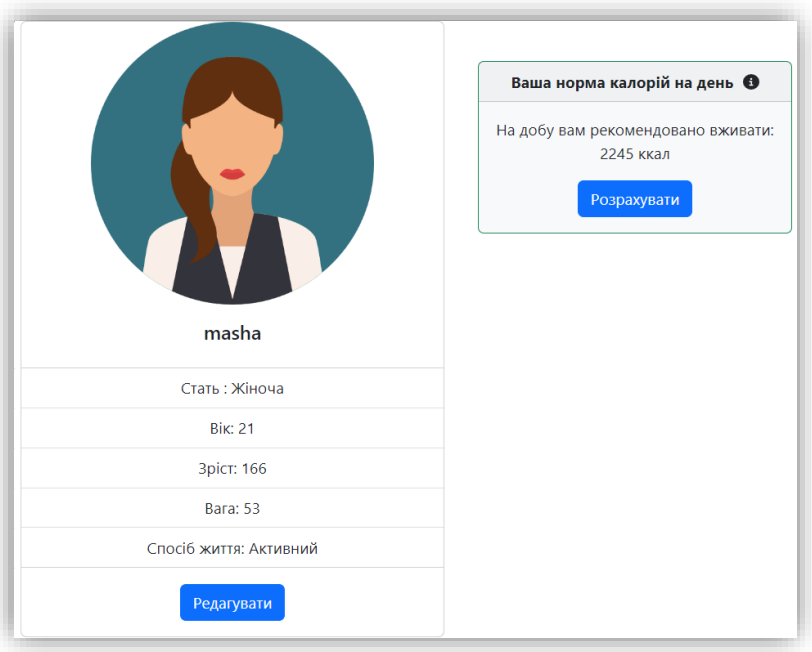
Перше, що бачить користувач при відкритті веб-застосунку це сторінка авторизації, котра дозволяє увійти в свій акаунт.



Якщо ж, акаунт ще не був створений, необхідно перейти на сторінку реєстрації за наявним посиланням:



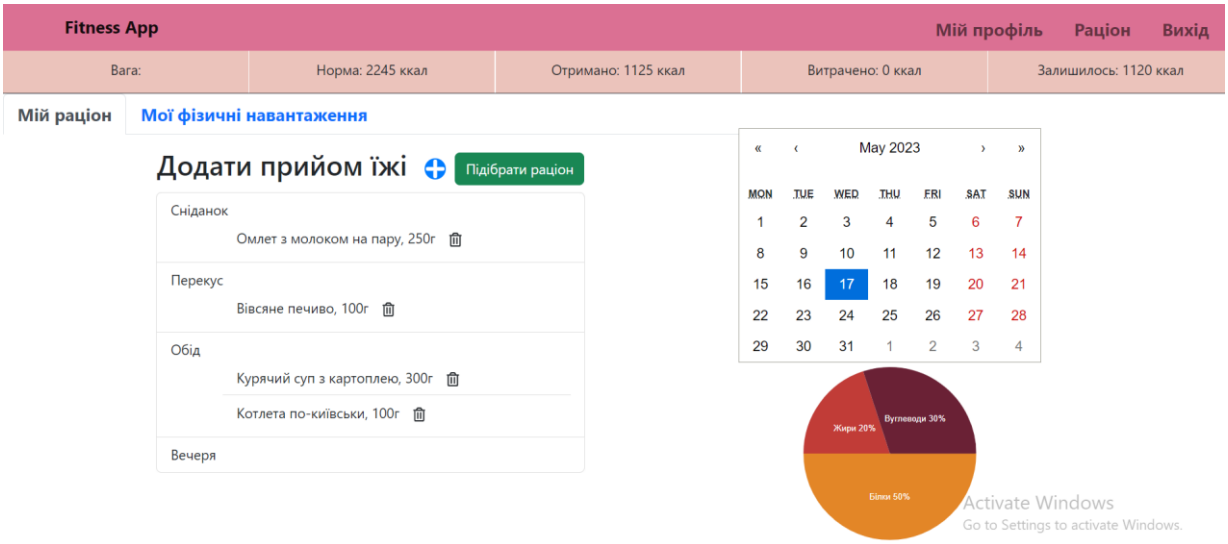
Після успішної реєстрації та авторизації користувач потрапляє на сторінку свого профілю, де він має змогу змінити дані профілю за потреби, а також значення особистої норми калорій на добу, розрахованого за спеціальною формулою.



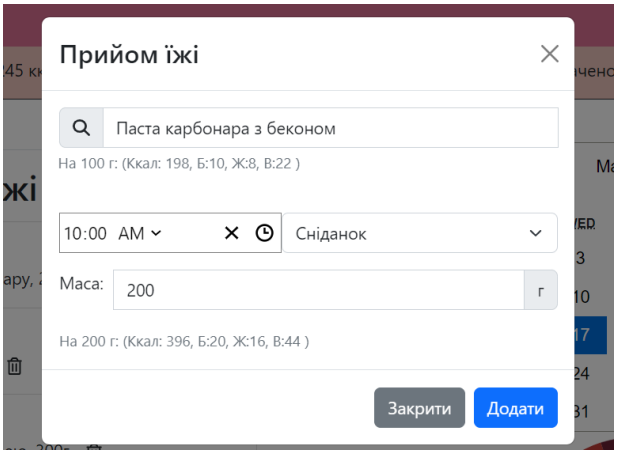
Доступ до основної логіки застосунку відбувається на наступній сторінці. Нижче буде наведено перелік можливостей, котрі отримає користувач після успішного заповнення свого профілю та розрахунку норми калорій:

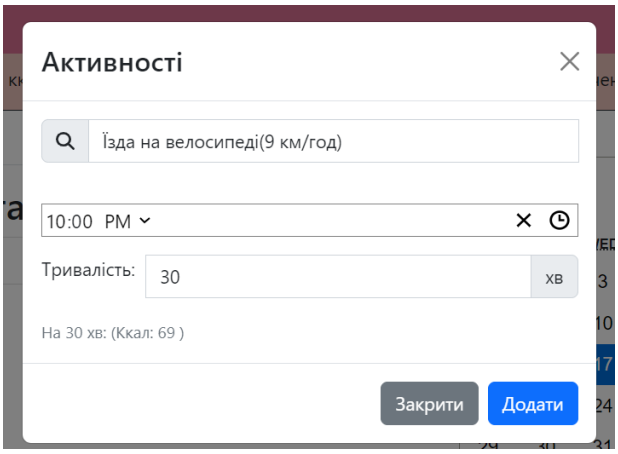
1) Можливість обирати із запропонованого переліку страв ті, которі вживав користувач для розрахунку їх харчової цінності та контролю дотримання обрахованої норми калорій.
2) Можливість обирати із запропонованого переліку різноманітні види спорту та активності, для розрахунку витрачених за день калорій.
3) Перегляд історії харчування та фізичних навантажень.
4) Отримання співвідношення мікроелементів (білків жирів та вуглеводів), котрі входять до денного раціону у вигляді діаграми.
5) Отримання рекомендацій щодо раціону харчування, на основі денної норми калорій та страв котрі вже були присутні в раціоні користувача.

Інтерфейс головної сторінки має такий вигляд:



Для додавання страв та фізичних навантажень на сторінці присутні наступні елементи:





5.2.1 Опис додаткових інструментів використаних під час розробки інтерфейсу

5.2.1.1 State Manager

Оскільки логіка застосунку вимагає постійної взаємодії з даними користувача, необхідно було забезпечити доступ до цих даних з усіх компонентів. React надає можливість використовувати спеціальні інструменти для зручної організації станів програми та централізованого управління станами (state managers). State managers дозволяють створити

сховище, в якому будуть зберігатися усі необхідні стани, і доступ до них можна буде отримати з будь-якого компонента програми.
Найпопулярнішими інструментами для керування станами є Redux, MobX, Recoil, Zustand.
При розробці мого застосунку було вирішено використовувати бібліотеку MobX. Вона забезпечує просте та гнучке управління станами програми за допомогою впровадження спостережуваних об'єктів, які автоматично відстежують і поширюють зміни в інтерфейсі користувача. Для того щоб позначити компоненти, в котрих будуть використовуватися стани сховища існує ключове слово “observable”. Коли будь-який стан спостережуваного об’єкта змінюється, MobX автоматично запускає повторний рендеринг відповідних компонентів. MobX є досить простим у використанні, допомагає зменшити кількість шаблонного коду та полегшує процес доступу до даних.
У сховищі зберігається об’єкт поточного користувача а також булева змінна, котра вказує чи аутентифікований користувач.

```
import {makeAutoObservable} from 'mobx';

class UserStore {
    constructor() {
        this._isAuth=false;
        this._currentUser = null;
        makeAutoObservable(this);
    }
    setCurrentUser(user) {
        this._currentUser = user;
    }
    setIsAuth(bool){
        this._isAuth = bool;
    }
    get currentUser(){
        return this._currentUser;
    }
    get isAuth(){
        return this._isAuth;
    }
}
export default UserStore;
```

5.2.1.2 Комунікація з сервером

Отримання даних із сервера та виконання різноманітних операцій з ними здійснюється за допомогою HTTP-запитів. З'єднання з сервером в моєму застосунку відбувалося за допомогою Fetch API. Це легкий низькорівневий API який забезпечує базові функції для надсилання запитів, дозволяє налаштовувати заголовки запитів і методи. Для обробки асинхронних операцій він використовує Promises. До прикладу, так виглядає запит на оновлення профілю користувача.

5.3 Реалізація рекомендаційної системи

5.3.1 Вибір підходу до реалізації

Для рекомендування страв користувачу було вирішено створити контент-базовану рекомендаційну систему. Цей вибір обумовлений декількома причинами:

- Доступність характеристик страв: система рекомендацій на основі вмісту використовує чітку інфомацію про характеристики страв. В даному випадку, це вміст макроелементів, колорійність та категорія. Це дозволяє створити профіль кожної страви та обчислювати значення їхньої схожості.
- Прозорість: рекомендації на основі вмісту часто є більш прозорими у порівнянні з іншими підходами. Оскільки рекомендації грунтуються на відомих характеристиках страв, користувачу легше зрзуміти та інтерпретувати чому були зроблено певні рекомендації.

- Незалежність від поведінки користувачів: контент-базовані рекомендаційні системи не вимагають інформації про інших користувачів або їхні вподобання. Це стало у нагоді, адже створення великої бази користувачів є досить довгим процесом.
- Спеціальні рекомендації: підходи, що базуються на контенті, можуть бути особливо корисними, коли є необхідність надавати рекомендації на основі конкретних дієтичних вимог, харчових цілей або уподобань. Враховуючи вміст макроелементів і калорій, система може запропонувати страви, які відповідають конкретним потребам користувача, наприклад страви з низьким вмістом вуглеводів, з високим вмістом білка або вегетаріанські страви.

5.3.1 Реалізація рекомендаційної системи

Створення рекомендаційної системи складалося з наступних кроків:

1) Створення профілю користувача та збереження його фізичних характеристик, необхідних для розрахунку денної норми калорій.
2) Розрахунок денної норми калорій за допомогою спеціальної формули.
3) Оскільки рекомендації будуються на основі схожості страв, користувач має самостійно скласти свій раціон на декілька днів, дотримуючись заданої норми калорій та співвідношення макроелементів.
   Після цього відкривається можливість скористатися послугами рекомендаційної системи. Для пошуку готового раціону створений запит зі сторони серверу, котрий повертає список страв для кожного прийому їжі. Система буде здійснювати рекомендаціїї, шукаючи страви подібні до тих, що є в цьому списку.

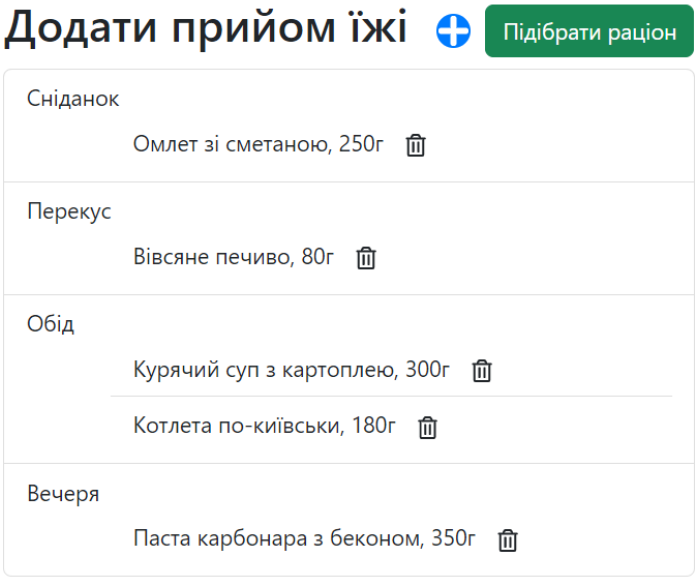
```
@Query("SELECT m FROM Meal m " +
        "WHERE m.user = :user " +
        "AND m.date IN (" +
        "    SELECT m1.date FROM Meal m1 " +
        "    WHERE m1.user = :user " +
        "    GROUP BY m1.date " +
        "    HAVING COUNT(DISTINCT m1.mealType) = (SELECT COUNT(DISTINCT m2.mealType) FROM Meal m2 WHERE m2.user = :user)) " +
        "ORDER BY m.date")
List<Meal> findMealsForUserWithAllTypesPresent(@Param("user") User user);
```

4) Нормалізація даних: представлення характеристик страв у вигляді векторів. Також, на цьому етапі необхідно переконатися, що кожен макроелемент має однакову вагу при обчисленні відстані.
5) Обчислення подібності страв. Для цього була використана традиційна формула відстані між двома точками – Евклідова відстань. Формула відстані між двома стравами виглядає наступним чином:

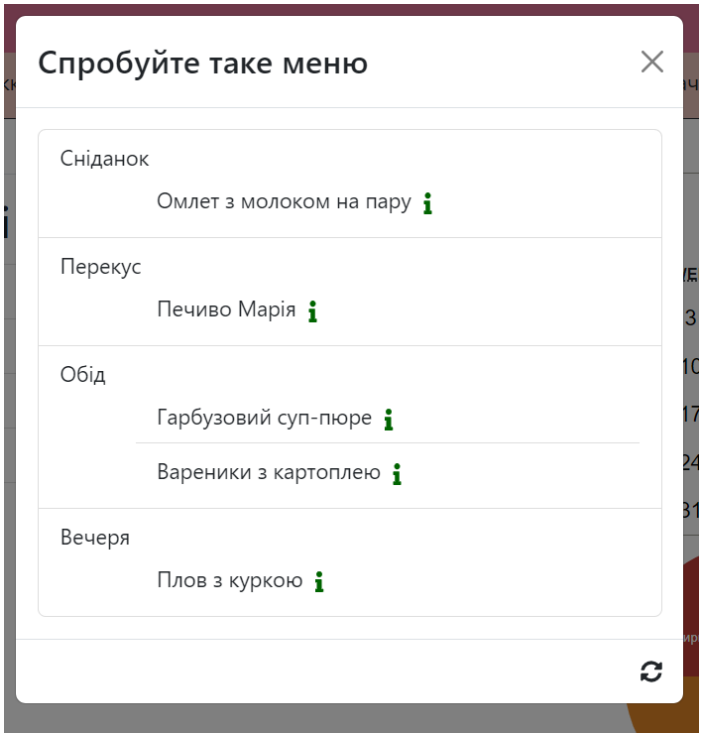
$$\sqrt{(A_{calories} - B_{calories})^2 + (A_{protein} - B_{protein})^2 + (A_{fat} - B_{fat})^2 + (A_{carb} - B_{carb}$$

6) Останнім кроком є визначення найбільш подібних страв, котрі будуть утворювати збалансований раціон харчування та рекомендування їх користувачу. В першу чергу будуть рекомендуватися найбільш подібні страви до тих, які вже вживав користувач. При повторних запитах рекомендації можуть бути менш якісними, але ця проблема може бути вирішена розширенням бази даних та наявністю достатньої кількості страв зі схожими характеристиками.

Рекомендації здійснюються за запитом користувача.



При натисканні на кнопку «Підібрати раціон», з'являється модальне вікно з готовим раціоном.



Якщо користувач хоче переглянути інший варіант раціону, потрібно лише натиснути на кнопку в нижній частині вікна. Після цього згенерується новий набір страв.

Основною метою даної рекомендаційної системи є урізноманітнення раціону харчування шляхом підбору різних страв зі схожими поживними цінностями.

Висновки

У ході виконання даної кваліфікаційної роботи було виконано усі поставленні задачі, а саме:

- Розглянуто особливості створення веб-додатків за допомогою фреймворку Spring Boot, описано використані інструменти та технології.
- Створено веб-додаток, котрий дозволяє користувачам розраховувати особисту норму калорій на основі фізичних характеристик, контролювати баланс макроелементів та кількість калорій у денному раціоні, підраховувати кількість калорій, витрачених під час різноманітних фізичних навантажень.
- Досліджено різні типи рекомендаційних систем та підходи до їх реалізації.
- Створено контент-базовану рекомендаційну систему для генерації рекомендацій страв, що входять до денного раціону.

Існує досить великий простір для розвитку даного проекту та розширення його функціоналу. Наприклад, модифікація рекомендаційної системи шляхом впровадження методів колаборативної фільтрації для здійснення більш точних рекомендацій, або ж збільшення кількості функціоналу, доступного користувачам, зокрема планується додати можливість комунікації між користувачами в межах застосунку для обміну ідеями та спільною роботою над своїм харчуванням.

Список використаної літератури

1. Java Basics: What Is Spring Boot? | JRebel by Perforce. *JRebel by Perforce*. URL: https://www.jrebel.com/blog/what-is-spring-boot#spring-boot-overview (date of access: 21.05.2023).
   Spring Boot - Introduction. *Online Courses and eBooks Library*. URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm (date of access: 21.05.2023).
2. Siddiqui A. Authentication vs Authorization. *Medium*. URL: https://medium.datadriveninvestor.com/authentication-vs-authorization-716fea914d55 (date of access: 21.05.2023).
3. Authentication vs. Authorization. *Auth0 Docs*. URL: https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization#what-are-authentication-and-authorization- (date of access: 21.05.2023).
4. Understand Spring Security Architecture and implement Spring Boot Security | JavaInUse. *Home | JavaInUse*. URL: https://www.javainuse.com/webseries/spring-security-jwt/chap3 (date of access: 21.05.2023).
5. Учасники проектів Вікімедіа. JSON Web Token – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 21.05.2023).
6. Spring Boot Architecture - javatpoint. *www.javatpoint.com*. URL: https://www.javatpoint.com/spring-boot-architecture (date of access: 21.05.2023).
7. Grover P. Various Implementations of Collaborative Filtering. *Medium*. URL: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0 (date of access: 21.05.2023).
8. Su X., Khoshgoftaar T. M. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*. 2009. Vol. 2009. P. 1–19. URL: https://doi.org/10.1155/2009/421425 (date of access: 21.05.2023).

Список використаних зображень

2.1 https://www.javainuse.com/series-2-2-min.jpg

5.2 https://static.javatpoint.com/springboot/images/spring-boot-architecture2.png

5.1 https://media.geeksforgeeks.org/wp-content/uploads/20220306155306/Fig47.jpg