МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему

Платформа для координації спортивних занять з елементами move-to-earn

Виконав: студент групи КІ-22
спеціальності
123 «Комп'ютерна інженерія»
Азьмук Н. А.

Науковий керівник канд. техн. наук, доцент

Розломій І.О.

Рецензент канд. техн. наук, ст. викл.

Стабецька Т.А.

Черкаси, 2024

ЗМІСТ

ВСТУП

У сучасному світі розвиток технологій інформаційних систем відкриває безліч можливостей для покращення якості життя людей. Однією з таких можливостей є використання інтерактивних платформ для спортивних занять з елементами move-to-earn, які стають дедалі останніми днями.

Актуальність теми. Фізична активність та спорт є важливими складовими здорового способу життя. За останні роки відбулося значне зростання популярності спортивних заходів та тренувань, що спонукає до пошуку нових технологій для координації цих занять. Розробка платформи для координації спортивних занять з елементами move-to-earn є актуальною темою дослідження в галузі комп'ютерних наук. Move-to-earn - це підхід до спорту, в якому за досягнення певних результатів у тренуваннях користувачі отримують винагороду. Це дозволяє поєднувати фізичну активність з отриманням матеріальних стимулів, що є мотивуючим фактором для більшості людей.

Мета і задачі розробки. Мета полягає в реалізації проекту, що дасть змогу багатьом спортсменам різних видів спорту структурувати та отримувати статистику стосовно тренувань, кооперуватись та надавати важливу інформацію про тренувальний процес тренеру. Фактично даний додаток являє свого роду щоденник тренувать з елементами соціальної мережі та взаємодії з тренером. Окрім цього, метою є створення інтуїтивно зрозумілого та простого в управлінні інтерфейсу, який забезпечить користувачам комфортну та легку навігацію по додатку. В плани входить реалізувати функції що дозволять спортсменам бути завжди в курсі свого тренувального розкладу та не пропускати жодного тренування. Окрім того, передбачається можливість налаштування персоналізованого тренувального плану, який буде розроблений з урахуванням індивідуальних особливостей спортсмена, його мети та поточного рівня фізичної підготовки.

Для успішної реалізації даного програмного продукту необхідно виділити основні завдання:

1. Дослідити предмету область додатків для координації спортивних занять.

2. Виділити основні переваги та недоліки існуючих додатків для чіткого розуміння очікувань ринку.
3. Дослідити існуючі інструменти для реалізації та інтеграції спортивних занять в рамках інформаційної системи.
4. Виконати проектування програмного забезпечення відповідно до вимог.
5. На основі розробленої архітектури програмно реалізувати програмний продукт.
6. Виконати тестування роботи додатку.

Об’єктом розробки є процес планування та аналізу спортивних тренувань.

Предметом розробки є веб-додаток що реалізує можливості планування спортивних тренувань з їх подальшим аналізом.

Практична цінність продукту полягає в можливості покращення процесу тренувань, убезпечення спортсменів від травм та збільшення контролю над тренувальним процесом.

1. Огляд та порівняльний аналіз аналогів додатків спортивних платформ для кооперації

1.1 Дослідження додатку збереження занять – Runkeeper

Даний додаток було розроблено компанією ASICS, що є відомим виробником взуття, одягу, екепірування, в тому числі і для бігу [1]. Це означає що компанія вже має аудиторію та ресурси щоб розвивати додаток. Версії додатку наявні на Adnrois та IOS.

Рис. а)

Рис. б)

Рисунок 1.1 а – Екран тренувань додатку. б – Екран користувача

Функціонал додатку достатньо широкий. Є можливість записувати тренування в реальному часі, слідкувати за показниками організму під час тренування та маршрутом.

В даному програмному забезпеченні (ПЗ) наявні можливості встановлення цілей, аналізу тренувань, перегляду статистики. Також присутня соціальна складова: є можливість переглядати результати інших користувачів та свої.

Тобто, можна зробити висновок, що даний додаток позіціонує себе як трекер для тренувань, що має певні функції аналізу та підрахування статистики. Це не відповідає основній цілі додатку що є темою завдання, так як основна мета цілком відрізняється. До недавнього часу Runkeeper мав Health Graph API.

Використання цього API дозволило би надати користувачам, що вже користуються Runkeeper функціонал автоматичної синхронізації тренувань, що може значно спростити процес використання додатку.

Як висновок аналізу даного додатку можна зазначити, що домен двох застосунків є однаковим, але фактично проблеми вони вирішують різні. Також корисно переглянути статистику, яку підраховує додаток та тримати цю інформацію в фокусі при розробці власного рішення для підрахунку статистики.

1.2 Аналіз Runner's Log

Програма Runner's Log має безліч спільного з попередньою. Тут як і в попередньому прикладі основний акцент зроблено саме на тренуваннях [2].

Рисунок 1.2 – Екран статистики

Можна відзначити акцент на статистиці:

Рисунок 1.3 – Вигляд екрану календаря

Список функціоналу:

– використання вбудованої карти для навігації під час тренувань;

– синхронізація iPhone та iPad, використовуючи iCloud;
– перегляд календаря тренувань;
– відображення статистики;
– категоризація тренувань за тегами;
– імпорт та експорт даних у форматі CSV. Імпорт GPX файлів;

Як висновок можна знову повторити тезу про те, що призначення даного додатку є зовсім іншим. Цікавим є функціонал імпорту GPX файлів так як це може стати ультимативним рішенням як завантажувати тренування в систему.

1.3 Аналіз Running Diary

Знову додаток, що фактично повторює функціонал попередніх двох. Running Diary [3] використовується для запису дистанції бігу, кроків і встановлення цілей. Мета додатку — змусити користувача продовжувати працювати через додаток, щоб створити здоровий фізичний та психічний стан. Тобто, з тез що сформовані командою додатку можна зробити висновок, що справа йде про трекер активностей, а не про щоденник. Додаток розповсюджується на платній основі за ціною 9.99USD. Наявний функціонал встановлення цілей (рис. 1.4.а), та отримати статистику (рис. 1.4.б).

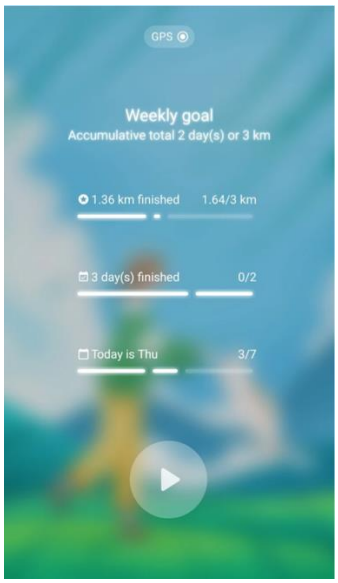


Рис. а)

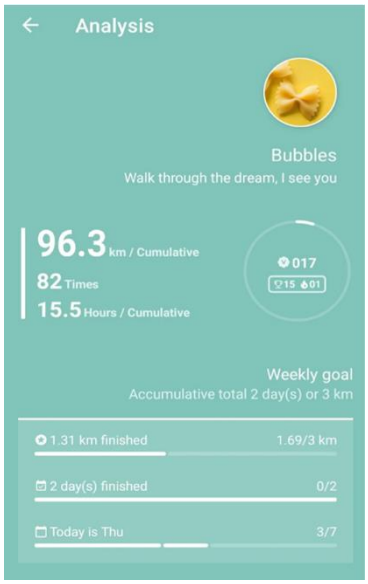


Рис. б)

Рисунок 1.4 а – Вигляд тижневої цілі. б – Вигляд статистики

Висновок стосовно даного додатку є досить передбачуваним: трекер активностей цілком відпрізняється від щоденнику та планеру для тренувань, але

слід взяти до уваги ціну додатку. Після розгляду кількості покупок можна зробити висновок, що 10 USD за додаток, що копіює функціонал попередніх готові заплатити досить мало людей, тому в теорії ціна преміум версії додатку що розробляється має бути меншою: від 3 до 5 долларів на місяць.

1.4 Аналіз Strava

Strava [4] – американський інтернет-сервіс для відстеження фізичних вправ, який включає функції соціальних мереж. В основному він використовується для їзди на велосипеді та бігу з використанням даних GPS. Strava використовує безкоштовну модель з деякими функціями, доступними лише в плані платної підписки. Сервіс був заснований у 2009 році Марком Гейні та Майклом Хорватом і розташований у Сан-Франциско, Каліфорнія.

Strava записує дані про діяльність користувача, якими потім можна поділитися з читачами користувача або публічно. Якщо діяльність поширюється публічно, Strava автоматично групує дії, які відбуваються в один і той же час і в одному місці (наприклад, участь у марафоні, спортивному або груповому заїзді). Записана інформація про діяльність може включати підсумок маршруту, висоту, швидкість (середня, мінімальна, максимальна), час (загальний час і час руху), потужність та частоту серцевих скорочень. Інтерфейс додатку де наведено груповий забіг (рис. 1.5.а).

Діяльність можна записувати за допомогою мобільного додатка або з пристроїв сторонніх розробників, таких як Garmin, Google Fit, Suunto і Wahoo. Діяльність також можна вводити вручну через веб-сайт Strava. Strava Metro, програма, призначена для міських планувальників, використовує велосипедні дані користувачів Strava у підтримуваних містах і регіонах. У травні 2022 року Strava оголосила, що компанія придбала додаток для запобігання травм Recover Athletics. Strava включила послуги та широкий спектр ресурсів із запобігання травмам у вигляді вмісту Recover Athletics у свій пакет підписки, включаючи персоналізовані плани підготовки та попередження травм.

Strava включає в себе функції соціальних мереж, які дозволяють користувачам публікувати свої вправи для підписників. Поряд з GPS-картою своїх тренувань користувачі також можуть публікувати фотографії. Після цього підписники можуть коментувати дописи та висловлювати «подяки» у вигляді кнопки «подобається». Beacon це функція, яка дозволяє користувачам Strava ділитися своїм місцезнаходженням у режимі реального часу з ким завгодно, а також призначати інших контактних осіб із безпеки під час тренування. Користувачі також можуть вводити віртуальні тренування на платформі. Користувачі підписки мають доступ до персоналізованих планів реабілітації та запобігання травмам Recover Athletics. Нижче наведено частковий приклад статистики, що надає Strava для тренуваня (рис. 1.5.б).

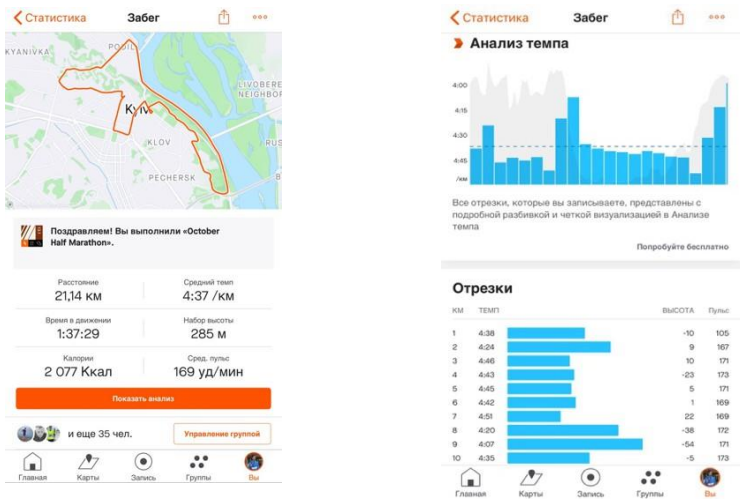


Рис. а). Рис. б)

Рисунок 1.5 а – Вигляд тренування. б – Вигляд статистики тренування

В довершення до всього додаток має API, використовуючи який можна забезпечити 99% потенційний користувачів функціоналов автоматичної синхронізації тренувань, тому що аудиторія що цікавиться бігов, має на меті тренуватись за планом у більшості випадків має активний акаунт Strava.

Переваги Strava:

1. Широка база користувачів
2. Різноманітні види спорту
3. Інтеграція з популярними провайдерами спортивних пристроїв

1.5 Постановка задачі на розробку

В наслідок розгляду наведених вище додатків, та завдяки дослідженню ринку, було сформовано спільну порівняльну таблицю:

Таблиця 1.1 – Порівняльна характеристика досліджуваних додатків

Назва додатку	Переваги	Недоліки
Runkeeper	GPS функціонал; Соціальні можливості (додавання друзів, груп); Наявний додаток для мобільних платформ	Збої в роботі GPS; Немає планів тренувань; Для основного функціоналу необхідна платна підписка
Runner’s Log	GPS фунціонал; Можливість створювати плани тренувань; Наявність мобільного додатку	Відсутність соціальної складової; Обмежена можливість кастомізації планів тренувань та статистики
Strava	Складова соціальної мережі; інтеграція з фітнес додатками; варіативна статистика	Відсутня можливість кооперації з тренерами; відсутня можливість повідомляти про свій стан та метрики

Дана таблиця являє собою результат порівняння характеристик досліджуваних додатків. Це дає змогу знайти важливі місця вразливості існуючих додатків та під час розробки виправити недоліки які існують в даній сфері. Таким чином розроблений комплекс буде являти собою унікальний компонент що здатен конкурувати з лідерами індустрії.

Висновки до розділу

В рамках розробки додатку був проведений аналіз ринку та існуючого програмного забезпечення, що вирішує проблеми сфери застосування. Аналіз

дав можливість виявити, що рішень які повністю покривають описаний функціонал, на ринку поки що немає, або вони настільки не популярні, що знайти їх важко. Вдалося виявити той факт, що цільовий додаток не має містити в собі функціоналу фітнес трекеру, так як для цього є вже досить рішень, які повністю задовольняють потреби користувачів. В тому числі завдяки аналізу вдолося виявити вдалі приклади підрахунку статистики, проаналізувати орієнтовну ціну платної підписки та знайти сервіси інтеграція з якими може значно покращити процес використання додатку.

На основі проведеного дослідження можна сформувати список базових вимог для додатку що є темою роботи:

1. Імплементація можливості створення планів тренувань, їх редагування та поширення
2. Реалізація базових соціальних складових для об’єднання тренерів і спортсменів
3. Виконання редактору планів тренувань
4. Відображення статистики виконання плану та результату тренувань
5. Реалізація move-to-earn складової для забезпечення інтересу
6. Інтеграція з платіжним сервісом для реалізації моделі підписок

Реалізація наведених задач дасть змогу розробити програмний продукт який здатний до конкуренції на сучасному ринку.

2. Аналіз і проектування системи

2.1 Виявлення, дослідження та аналіз вимог

В розрізі вимог слід розуміти, що сучасні системи мають тенденцію змінюватися під впливом ринку та користувачів. Тому необхідно виділити основні вимоги, які сформують фундамент майбутньої системи та дозволять їй мати достатню гнучкість щоб відповідати сучасним вимогам.

Аналізуючи аналоги та ринок подібних додатків в цілому було сформовано наступний список вимог до системи:

1. Система повинна являти собою веб-додаток доступний для перегляду та взаємодії в браузері. Це зумовлено простотою розробки та розповсюдження, відсутністю проблем з платформами та специфікою додатку. Специфіка полягає в тому що додаток має на меті бути щоденником, який зручно використовувати через персональний комп’ютер.
2. Дані користувачів мають зберігатись в хмарі, це дає гарантувати цілісність даних з точки зору відсутності локальних копій і необхідності синхронізації.
3. Користувач повинен мати змогу створювати акаунт в системи. Для створення акаунту достатньо поштової адреси та паролю.
4. Користувач повинен мати змогу завантажувати персональні тренування (біг), редагувати їх та переглядати статистику.
5. Користувач повинен мати змогу обрати доступний план тренувань або створити власний.
6. План тренувань може бути публічний або приватний. Питання приватності вирішує творець плану.
7. План тренувань повинен відображатись поряд з тренуваннями щоб користувач мав змогу переглядати які тренування входять в план та стан їх виконання користувачем.
8. Користувач має отримувати бонусні монети за відповідність тренувань планові. Ця функція має на меті мотивувати спортсменів.
9. Користувач має мати змогу повідомляти про свій фізичний стан кожного дня. Ця інформація необхідна для аналізу та покращення процесу тренувань.

10. Користувач має мати змогу переглядати статистику своїх тренувань, дистанцію, кількість тренувань та інші показники.

2.1.1 Представлення вимог за допомогою діаграми прецедентів системи

Діаграми прецедентів є важливим інструментом для аналізу та проектування систем, оскільки вони дозволяють визначити, які дії можуть виконуватися в системі та які актори (користувачі або інші системи) взаємодіють з системою.

У контексті розробки платформи для координації спортивних занять з елементами move-to-earn, можна виділити наступні прецеденти:

1. Реєстрація користувача. Даний прецедент дозволяє новим користувачам зареєструватися в системі. Користувач повинен ввести особисті дані, такі як електронну адресу та пароль для входу в систему.
2. Аутентифікація користувача. Даний прецедент дозволяє користувачам авторизуватися в системі. Користувач повинен ввести свій логін та пароль для входу в систему.
3. Створення тренування. Даний прецедент дозволяє користувачам створювати нові тренування з елементами move-to-earn. Користувач повинен ввести опис тренування, дату та час проведення та відстань.
4. Створення плану тренувань. Даний прецедент дозволяє користувачам створювати план тренувань що визначає механізм та послідовність виконання різних тренувань.
5. Вибір плану тренувань. Даний прецедент описує процес вибору конкретного плану для подальшого слідування в тренуваннях. Це дозволяє користувачеві поступово і стабільно розвиватись як спортсмен.

Планується мати тільки одного актора, користувача додатку, тому діаграма прецедентів буде розглядати тільки його взаємодію з системою. При побудові діаграми було використано 4 основні відношення-асоціації, включення, розширення та узагальнення.

На діаграмі (рис. 2.1) було візуалізовано список користувацьких вимог та їх відношення один з одним за допомогою вище згаданих відношень. Також на діаграмі продемонстровано основні користувацькі сценарії дій, що значно полегшило подальшу розробку.

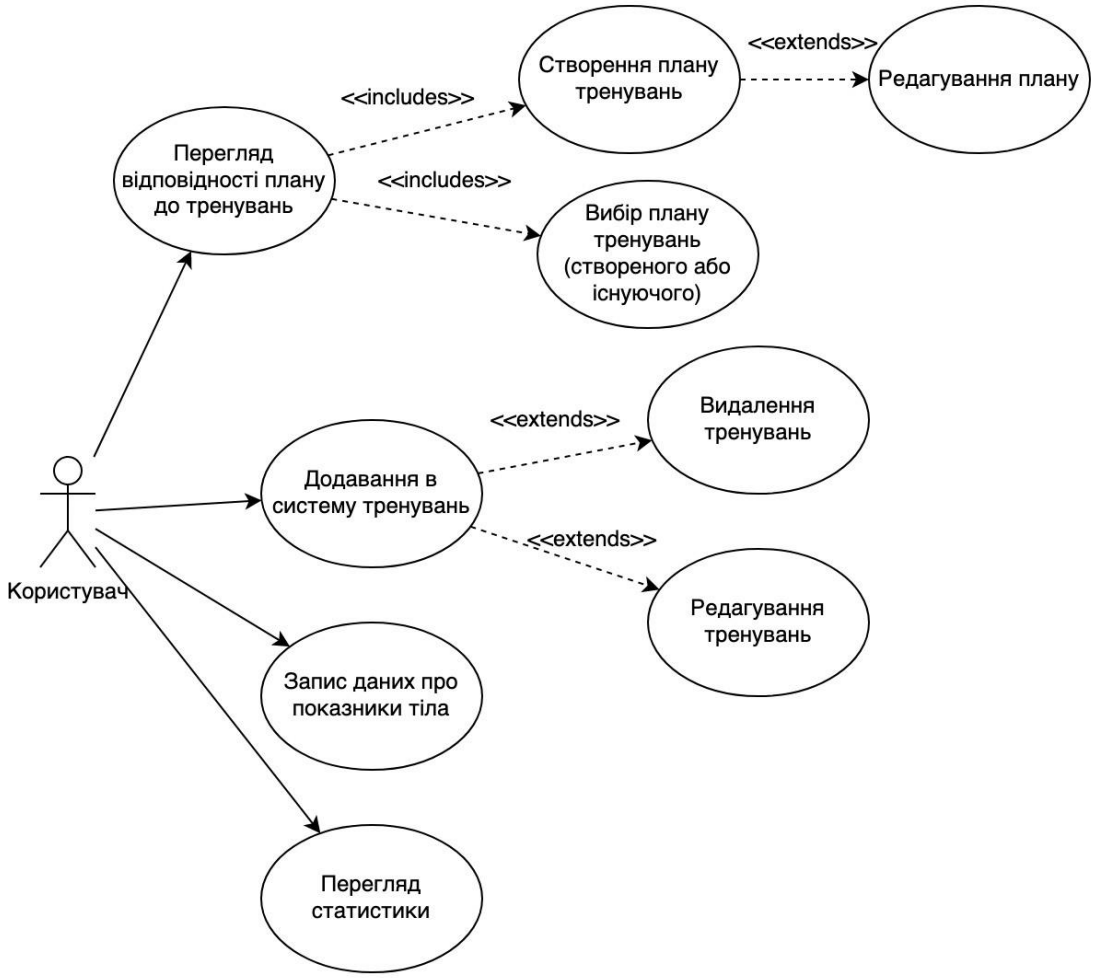


Рисунок 2.1 –Діаграма прецидентів системи

2.2 Розробка логічної моделі додатку

2.2.1 Представлення архітектури за допомогою діаграми класів

На даному етапі проектування необхідно спланувати та розробити діаграму класів системи. Розробка діаграми класів є важливим етапом проектування програмного забезпечення, оскільки дозволяє візуалізувати структуру системи та її компонентів. Це допомагає розробникам та іншим зацікавленим особам краще зрозуміти функціональність системи та взаємозв'язки між її компонентами. Діаграма класів дозволяє описати класи, їх

атрибути та методи, а також взаємозв'язки між класами, зокрема агрегацію, композицію, спадковість та залежності. Це дає змогу зрозуміти, які класи відповідають за яку функціональність, як вони взаємодіють між собою та які дані передають один одному. Крім того, діаграма класів допомагає уникнути помилок при розробці програмного забезпечення, так як дозволяє виявити потенційні проблеми та помилки ще на етапі проектування, коли виправлення їх менш витратне. Також вона дозволяє легше розуміти код програмного забезпечення та змінювати його у майбутньому.

Для розуміння представлення діаграми класів даного додатку необхідно визначитись з основними компонентами системи так як ці компоненти і є складовими діаграми. З аналізу вимог можна виділити такі сутності як: тренування, план тренувань, користувач, та сутність яка буде зберігати інформацію про стан спортсмена. Кожна з цих сутностей має визначені атрибути та методи, які будуть реалізовувати їх функціональність у системі. За допомогою діаграми класів буде можливо проілюструвати взаємозв'язки між цими сутностями, а також відображення їх взаємодії в рамках системи. Діаграма класів дозволить зосередитись на абстракції від реалізації та націлитись на функціональність системи. При розробці будь-якої системи, в тому числі й даного додатку, розуміння її архітектури є ключовим для успішної реалізації функціональності та забезпечення її якості. Використання діаграм класів є важливим етапом проектування, що дозволяє уникнути помилок при подальшому розробленні та зменшити кількість витраченого часу на тестування. Крім того, діаграма класів дозволяє легко змінювати архітектуру системи та додавати нові компоненти в неї без необхідності змінювати всю структуру системи. Це дозволяє бути більш гнучкими при розробці та змінювати систему відповідно до потреб користувачів.

З огляду на наведені вище тези було побудовано діаграму класів яка описує систему що є темою даної роботи (рис. 2.2).

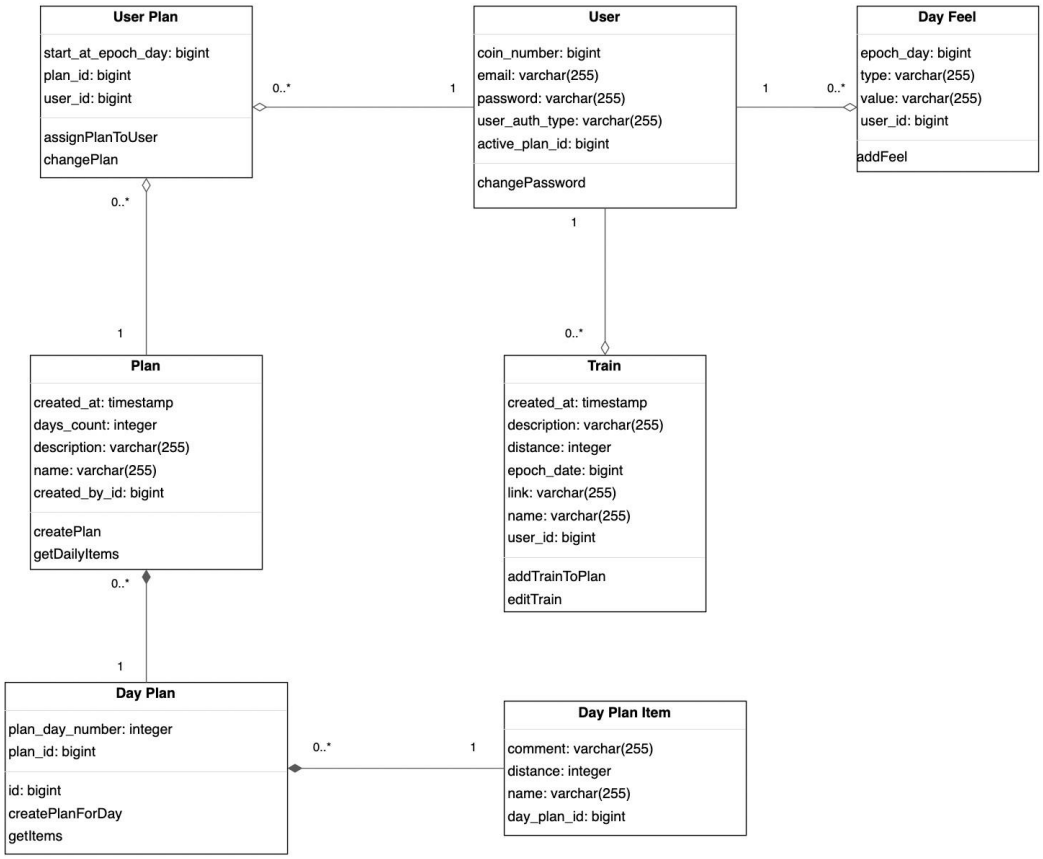


Рисунок 2.2 –Діаграма класів системи

На даній діаграмі зображено основні класи системи та базові операції що вони виконують.

Клас користувача має зв’язки з багатьма класами бо важливо зберігати дані про належність тренувань, планів та інших даних до конкретних користувачів. Клас плану користувача є композицією класів користувача та плану, це необхідно для забезпечення гнучкості системи та можливості описувати план тільки один раз а далі будувати зв’язки з різними користувачами. Тренування в свою чергу посилається на користувача та не залежить від плану. Це дає необхідну гнучкість системі так як користувач не зобов’язаний обирати план щоб завантажити тренування – тренуватися можна і без плану. Сутність плану в свою чергу має в собі колекцію сутностей по кожному дню, які в свою чергу містять елементи плану.

Дана структура забезпечує необхідні гнучкість та готовність до змін у майбутньому.

2.2.3 Побудова діаграми пакетів для представлення архітектури

Розробка діаграми пакетів є важливим етапом проектування, оскільки дозволяє зорієнтуватися в основних компонентах системи та з'ясувати залежності між ними. У діаграмі пакетів компоненти системи групуються за функціональністю або логічною спряженістю. Для створення діаграми пакетів для додатку необхідно визначити основні компоненти системи та групувати їх у логічні пакети. Метою діаграми пакетів є забезпечення загального уявлення про організацію системи та допомога розробникам зрозуміти залежності між пакетами.

На основі аналізу вимог можна визначити такі основні компоненти системи, як плани тренувань, тренування, користувачі та інформація про спортсмена. Можна згрупувати ці компоненти в логічні пакети згідно їх функціональності та взаємозв'язків. Наприклад, можна створити пакет з назвою «Плани», який міститиме класи які відносяться до планів тренувань. Інший пакет з назвою «Тренування» може містити класи що відносяться до тренувань. Пакет з назвою «Користувачі», містить класи користувачів та інформацію про спортсмена.

Крім основних компонентів, можна визначити підтримуючі компоненти, такі як доступ до бази даних та інтерфейс користувача. Ці компоненти можуть бути груповані в окремі пакети та пов'язані з основними пакетами за необхідності.

Діаграма пакетів забезпечує чітке візуальне представлення структури системи та допомагає забезпечити організованість та можливість підтримки системи. Організовуючи систему в логічні пакети, розробники можуть легко зрозуміти залежності між компонентами та внести зміни до системи, не впливаючи на не пов'язані компоненти.

Таким чином маємо діаграму пакетів яка описує систему та розділяє її на зони відповідальностей для зручності внесення майбутніх змін та модифікацій:

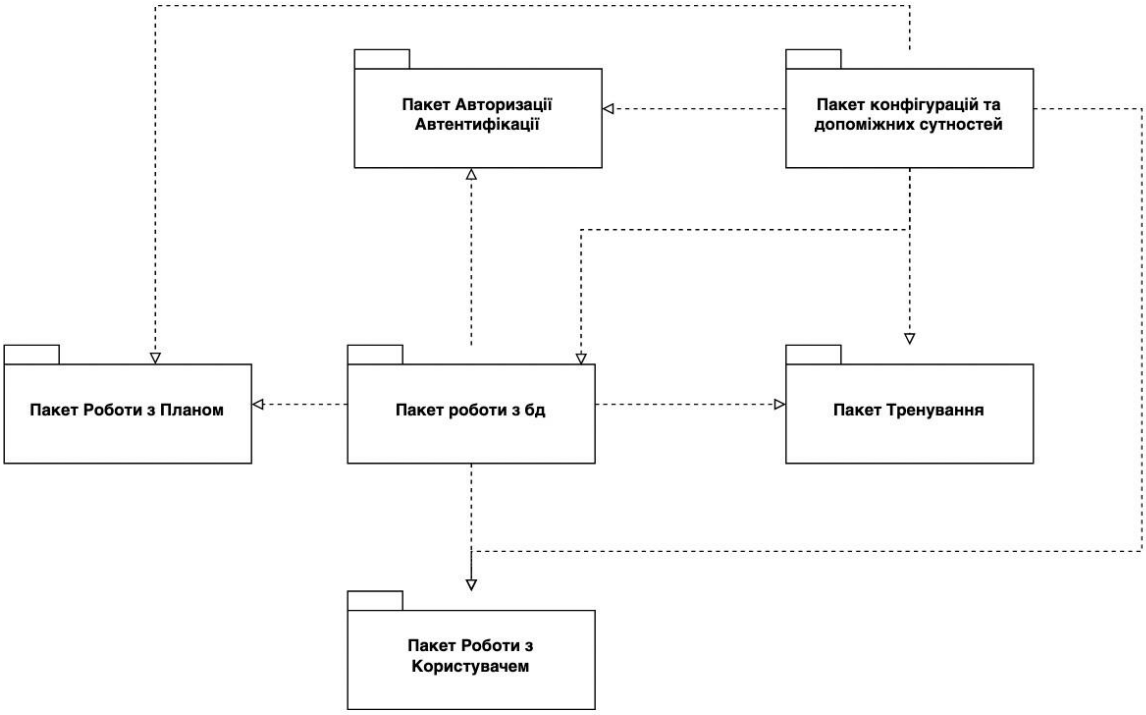


Рисунок 2.3 –Діаграма пакетів системи

Діаграма пакетів представлена на рис. 2.3 є відображенням підходу «Package by Feature» для побудови пакетів. Даний підхід – це метод організації компонентів програмного забезпечення шляхом групування їх за функціональними або бізнес-пов'язаними функціями. У цьому підході кожна функція або використання представлені окремим пакетом або модулем, який містить всі класи та компоненти, необхідні для реалізації цієї функції. Один з головних переваг Package by Feature – покращення модульності та інкапсуляції функціональності, що полегшує підтримку та модифікацію системи. Кожен пакет може бути розроблений та протестований незалежно, без впливу на інші пакети чи компоненти. За допомогою даного підходу розробники можуть швидко знайти та змінити код, пов'язаний з певною функцією, не шукаючи його у великій кодові базі. Це також сприяє кращому розумінню функціональності та бізнес-логіки системи, оскільки пакети названі за функціями, які вони реалізують.

Загалом, підхід «Package by Feature» є корисною технікою для організації та підтримки великих програмних систем, особливо тих з складними та взаємопов'язаними функціями.

2.3 Моделювання діаграми діяльності додатку

В даному розділі буде детально проаналізовано процеси, які відбуваються в додатку та запропоновано ефективну модель поведінки, яка буде відповідати вимогам користувачів та бізнесу.

Моделювання поведінки додатку за допомогою діаграми діяльності дозволяє визначити послідовність дій, які необхідно виконати для досягнення певної мети. Наприклад, метою даного додатку може бути створення плану тренувань для користувача. Для досягнення цієї мети потрібно виконати такі кроки, як авторизація користувача, вибір типу тренування, визначення тривалості та інтенсивності, та збереження плану. Кожен з цих кроків може бути представлений в діаграмі діяльності.

Діаграма діяльності дозволяє візуально представити послідовність дій та взаємозв'язки між ними. Вона може бути корисною для розробників, які хочуть розуміти логіку додатку, а також для тестувальників, які хочуть перевірити правильність роботи додатку.

Система що є темою даної роботи має тільки один тип користувача, для якого характерні наступні сценарії:

1. Вхід в систему нового та існуючого користувача.
2. Створення плану тренувань.
3. Вибір існуючого або створеного плану тренувань як активного.
4. Додавання тренувань в систему.
5. Введення даних про стан користувача.
6. Перегляд статистики тренувань.

Представлена діаграма (Додаток А) описує наведені вище сценарії та реалізує їх.

2.4 Проектування бази даних системи

2.4.1 Інфологічне проектування

Інфологічне проектування є ключовим етапом в розробці будь-якої програмної системи. Цей етап передбачає аналіз та моделювання бізнес-процесів та даних, які взаємодіють між собою в рамках системи.

В рамках даної роботи було проведено детальний аналіз бізнес-процесів та вимог до системи управління фінансами. За результатами аналізу, було розроблено інфологічну модель системи, що відображає всі сутності, взаємозв'язки та атрибути, що потрібні для успішної реалізації системи.

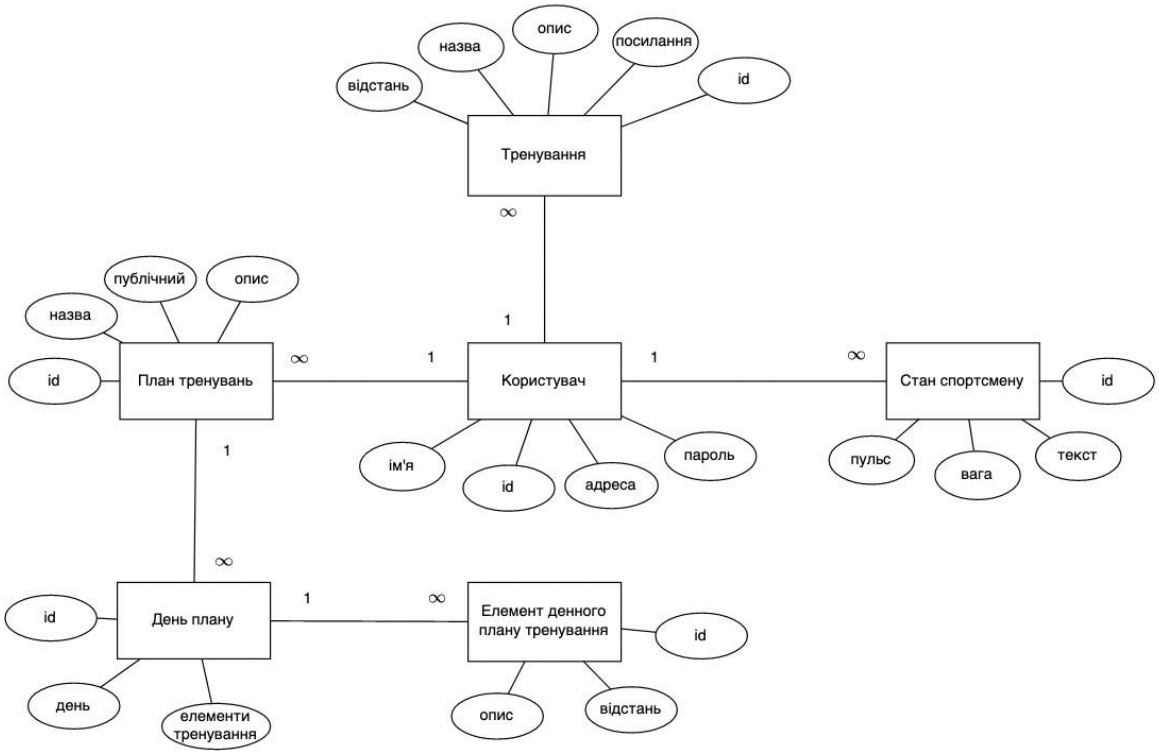


Рисунок 2.5 –Інфологічна модель бази даних

Дана модель описує структуру даних додатку на досить високому рівні, але достатньому для розуміння основних концепцій та ідей навколо проектування.

Сутність користувача лежить в основі структури так як дозволяє розмежовувати належність тренувань та планів різним клієнтам. Дана сутність містить в собі стандартний набір полів таких як пароль, ідентифікатор, ім’я користувача та електронна адреса.

Сутність тренування також є однією з базових так як представляє екземпляр тренування на рівні бази даних. Тренування може містить в собі інформацію про користувача, відстань, опис, назву тренування та посилання на тренування в одній з соціальних мереж для спортсменів. Зв'язок з сутністю користувача є один до багатьох, це пояснюється тим що один користувач може і буде мати безліч тренувань, тому необхідно закладати це знання в початкову структуру.

Сутність яка описує стан спортсмена покликана зберігати в собі статистичні дані, які допомагають відслідковувати стан спортсмена. Пульс спокою дає інформацію про стан судинної системи так як частота сердечних скорочень в спокої сигналізує про стан відновлення організму. Вага є також надважливим показником в тренувальному процесі так як саме від ваги залежить зусилля необхідне м’язам щоб пересувати тіло. Але мала вага також не є добре і свідчить про проблеми з харчуванням або здоров’ям, можливо це сигналізує про страту м’язів або хворобу, тому важливо відслідковувати ці дані регулярно. Дана сутність має зв'язок тільки з сутністю користувача. Це дає їй змогу не прив’язуватись до тренувань та плану. Таким чином спортсмен може користуватись платформою навіть без завантаження тренувань (під час травм) та фіксувати свій стан в додатку.

План тренувань є сутністю яка фіксує інформацію про план за яким тренується спортсмен. Даний план може бути складений самим спортсменом або іншим користувачем платформи. Дана сутність містить в собі інформацію про назву плану, видимість (публічний чи приватний доступ), опис, ідентифікатор та дочірні сутності. Дана сутність має зв'язок з сутністю користувача як один до багатьох. Це зумовлено тим що користувач потенційно може мати декілька планів. Аналогічно сутність має зв'язок з дочірньою сутністю яка описує конкретний день плану. Такий розділ зумовлений зручністю проектування та побудови запитів в майбутньому. Дана містить в собі інформацію про дату і екземпляри шаблонів тренувань які необхідно виконати. Дані шаблони тренувань описуються однойменною сутністю. Зв'язок аналогічний до

попередніх – однин до багатьох. Такий зв'язок пояснюється тим що план має в собі багато днів, а день може мати більше одного тренування.

2.4.2 Даталогічне проектування додатку тренувань

Даталогічне проектування є одним з найважливіших етапів в розробці програмного забезпечення, оскільки воно визначає, як дані будуть зберігатися та взаємодіяти між собою в системі. Цей етап включає в себе визначення потреб користувачів та їх взаємодії з додатком, відповідну моделювання та опис структури бази даних.

Дана система буде побудована на основі реляційної бази даних так як дані які будуть зберігатися в рамках додатку підпадають під реляційну модель краще ніж під документу. Саме тому даталогічна модель буде мати багато зв’язків та різних сутностей.

Перед фактичним представленням моделі необхідно описати сутності та поля які будуть належати до них, також слід визначитись зі зв’язками. Дана інформація вже була частково продемонстрована в рамках моделей вищого рівня. Тим не менше даталогічна модель є прямим відображенням реляційності моделі бази даних тому потребує кращої деталізацій.

Таблиця user (користувач):

– поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
– поле email (електронна адреса) являє собою бізнес-ідентифікатор користувача та необхідне для входу в систему;
– поле password (пароль) є строковим типом даних та буде зберігати в собі пароль в закодованому вигляді;
– поле user_auth_type зберігає в собі інформацію про те як користувач був зареєстрований: через OAuth чи стандартним способом з паролем;
– поле active_plan_id є зовнішнім ключем до таблиці user_plan та містить в собі ідентифікатор плану за яким тренується користувач в даний момент;

Таблиця plan (план тренувань):

- поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
- поле created_at (дата створення) є полем часового типу та містить інформацію про час створення сутності плану;
- поле days_count (кількість днів) є полем числового типу та містить інформацію про кількість днів на який розрахований план тренувань;
- поле description (опис) є полем текстового типу та містить довільну інформацію про план;
- поле name (ім’я) є полем текстового типу та містить інформацію про назву плану. Дана інформація необхідна для відображення на стороні клієнту;
- поле created_by_id є посиланням на таблицю користувачів та містить в собі ідентифікатор користувача хто створив план;

Таблиця user_plan (план користувача):

- поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
- поле user_id є зовнішнім ключем на таблицю користувачів та містить ідентифікатор користувача який обрав план;
- поле plan_id є зовнішнім ключем на таблицю планів. Разом з полем user_id вони утворюють відношення MANY-TO-MANY між таблицями планів та користувачів;
- поле start_at_epoch_day є полем числового типу та містить в собі порядковий номер дня епохи на який план починає свою дію;

Таблиця coin_transaction (транзакції монет):

- поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
- поле user_id є зовнішнім ключем на таблицю користувачів та містить ідентифікатор користувача якому належать монети;
- поле amount є полем числового типу та зберігає інформацію про кількість монет нарахованих користувачеві в рамках даної транзакції;

Таблиця day_plan (день плану):

- поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
- поле plan_id є зовнішнім ключем на таблицю планів та сигналізує про те до якого плану належить даний день;
- поле plan_day_number є полем числового типу та містить в собі порядковий номер даного дня;

Таблиця day_plan_item (елемент плану):

- поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
- поле day_plan_id є зовнішнім ключем на таблицю днів плану та сигналізує про те до якого дню належить даний елемент;
- поле distance (дистанція) є полем числового типу та містить в собі інформацію про відстань яку має подолати спортсмен під час тренування;
- поле name (ім’я) є полем текстового типу та містить в собі інформацію про назву тренування плану;
- поле commend (коментар) є полем текстового типу та містить в собі інформацію про те як необхідно виконувати дане тренування або довільні настанови;

Таблиця day_feel (відчуття):

- поле id (ідентифікатор) необхідне для задання унікальності запису та буде використовуватись як значення зовнішнього ключа на рівні інших таблиць;
- поле epoch_day (день епохи) є полем числового типу та містить в собі порядковий номер дня епохи на який припадає інформація про показники користувача;
- поле type (тип) є полем текстового типу та містить в собі значення типу запису. Це може бути запис ваги, пульсу або довільний коментар;
- поле value (значення) є полем текстового типу та містить в собі значення заданого типу показнику;
- поле user_id є зовнішнім ключем на таблицю користувачів та містить ідентифікатор користувача якому належить запис;

Таким чином маємо опис структури в якій наявні таблиці що описують модель даних додатку. Дані таблиці собою утворюють зв’язки за допомогою зовнішніх ключів. Зв’язки між таблицями зображено на відповідній діаграмі що представляє собою даталогічну модель (рис. 2.6).

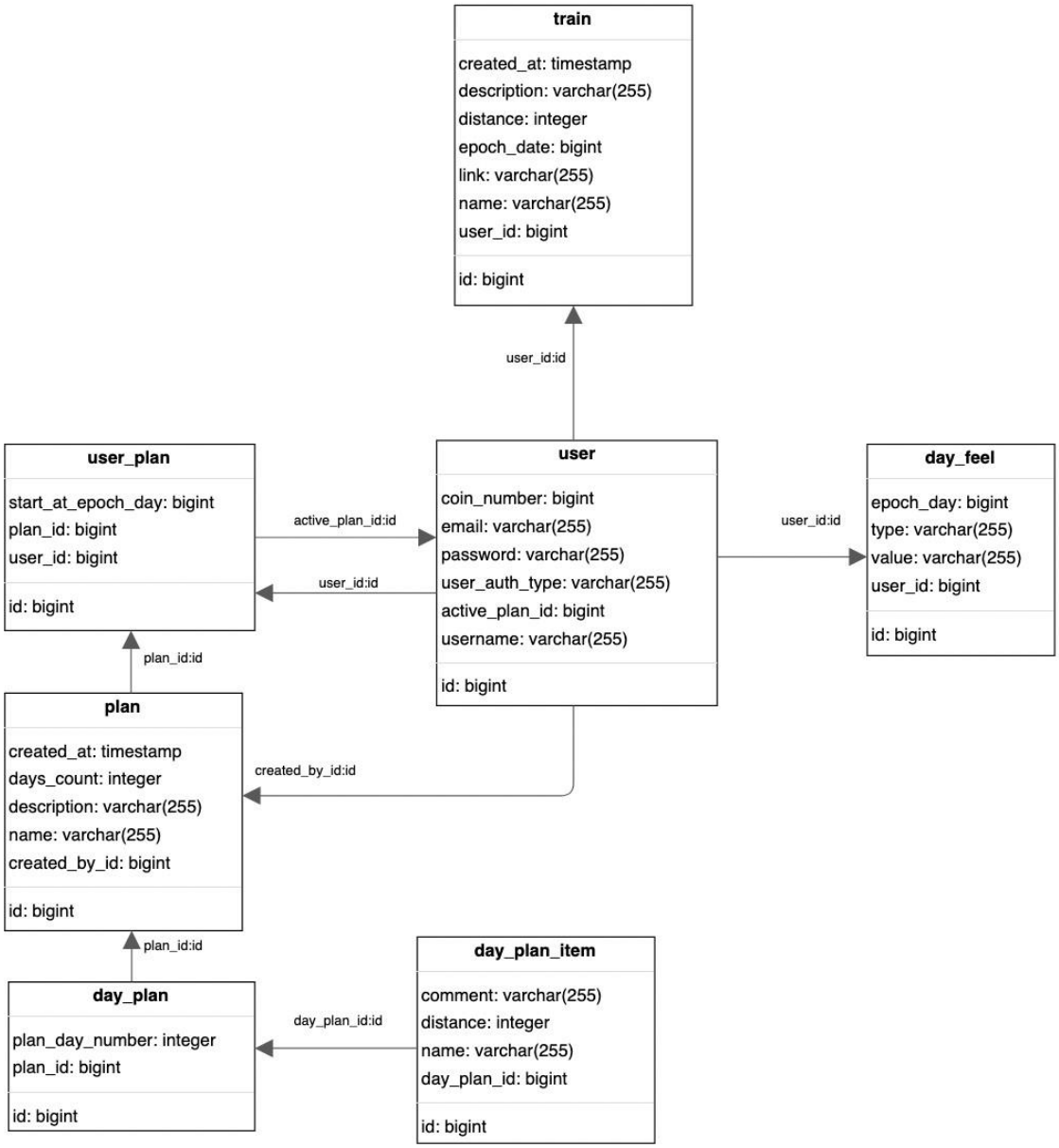


Рисунок 2.6 – Даталогічна модель бази даних

Висновки до розділу

В даному розділі було проаналізовано вимоги до програмного продукту та розглянуто процеси моделювання архітектури програмного забезпечення за

допомогою діаграм класів та пакетів, а також даталогічного проектування бази даних. Використання діаграм класів дозволяє створити візуальну модель компонентів системи та їх взаємодії, що полегшує подальшу розробку та розуміння системи. Діаграми пакетів можуть бути використані для групування компонентів за функціональністю та логічної організації, що сприяє модульності та підтримці системи.

Даталогічне проектування бази даних дозволяє створити відношення між об'єктами даних та описати їх залежності, що відображає логічну структуру даних та забезпечує їх цілісність та консистентність. Відповідне проектування бази даних є важливим етапом процесу розробки програмного забезпечення та дозволяє підтримувати ефективну роботу системи.

Загалом, використання діаграм класів та пакетів для моделювання архітектури програмного забезпечення та даталогічного проектування бази даних дозволяє забезпечити якість та ефективність розробки, що є важливими факторами для успішної реалізації проекту.

На основі архітектури що було сконструйована в рамках даного розділу має сенс перейти до реалізації програмного продукту та його тестування.

3 РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ

3.1 Вибір інструментарію для реалізації

Розробка додатку є комплексним процесом і найважливішим етапом завжди є реалізація. Для того щоб реалізувати програмний продукт необхідно мати набір певних інструментів що покликані допомогти розробнику в розробці. Від обраного інструментарію напряму залежить майбутнє проекту так як розробка не завершується після першого запуску системи, а навпаки – починається. Це пов’язано з тим фактом що, на практиці, більшість зусиль припадає не на первинну розробку системи, а на її підтримку. Тому критично важливо обирати інструментарій не тільки з точки зору власних бажань або знань, а і враховувати сучасні тенденції на ринку праці. Дане твердження підтверджується статистикою переходів працівників між різними компаніями з програмного забезпечення.

В наш час для менеджменту критично важливо побудувати процеси таким чином, щоб витрати на пошук нових розробників, потенційне розширення команди та інше – були мінімальними. Цього можна досягти багатьма способами, але найважливіший з них це правильний вибір інструментарію. Так як проект виконаний з використанням застарілих технологій не може зацікавити як досвідченого розробника, так і початківця так як зрозуміло що досвід роботи на даному проекті не буде достатньо релевантним.

Таким чином необхідно правильно обирати інструментарій та засоби для реалізації проекту. В рамках даного додатку є необхідність обрати інструментарій для розробки серверної частини, базу даних та клієнтської частини разом з дизайном.

3.1.1 Вибір інструментарію для реалізації серверної частини

Серверна частина додатку є місцем де зосереджено логіку поведінки, перевірки та механізми збереження, перестворення даних. Саме серверна частина поєднується з базою даних та надає дані клієнтській частині.

Для виконання серверної частини за основу було взято мову Java. Дана мова програмування є однією з найпопулярніших, це гарантує той факт, що розширювати команду та шукати нові кадри для розширення проекту буде відносно легко.

У якості фреймворку для мови Java було обрано Spring [5]. Даний фреймворк надає безліч різноманітних бібліотек для інтеграції з сервісами та інструментами різного типу, позбавляє проблем з конфліктами версій бібліотек і в цілому є стандартом індустрії для мови Java. В тому числі даний фреймворк реалізує принцим інверсії контролю, що робить процес програмування максимально гнучким та зручним.

Під час розробки моделі бази даних було з'ясовано, що модель даних є досить реляційною, тому цілком логічно використовувати реляційну базу даних. Серед реляційних баз даних існує безліч конкурентів. Для остаточного вибору слід брати до уваги унікальну функціональность кожної з баз даних та приймати рішення на основі факторів специфіки проекту, даних та побажань команди. В даному випадку вибір було зупинено на реляційній базі даних PostgreSQL [6]. Дана база даних відрізняється своєю системою розширень які досить легко встановлювати тим самим розширюючи її можливості. В рамках даного проекту не передбачається робота з розширеннями, тим не менше, ця можливість може стати у нагоді в майбутньому. Важливим чинником також слугує досвід в роботі з базою даних, це і є основний фактор чому вибір було зупинено на даній базі даних.

Для інтеграції серверного додатку з базою даних існує багато інструментів: ORM, JDBC та багато інших. Так як ORM надає можливості зручного представлення моделі даних у вигляді об'єктів мови програмування, а обраний фреймворк надає можливості почати використовувати даний інструментарій з мінімальними налаштуваннями – було обрано Hibernate ORM [7].

Так як дані мають структуру – виникає необхідність в інструментах міграції баз даних. В даному випадку вибір було зупинено на бібліотеці міграцій

Flyway [8] так як вона є досить популярною та підтримується як обраним фреймворком так і базою даних.

Таким чином набір інструментів для розробки серверної частини полягає в фреймворкові Spring для мови Java, Hibernate ORM та PostgreSQL.

3.1.2 Вибір інструментів для розробки клієнтської частини

Даних додаток є веб-орієнтованим, тому клієнтська частина потребує використання інструментів саме для веб-розробки. Клієнтська частина додатку представляє собою частину коду яка виконується в браузері на пристроях користувача.

Перед імплементацією клієнтської частини завжди розробляють інтерфейс та будують користувацький досвід взаємодії з додатком. На даному етапі розробки додатку часто виникає необхідність змінювати дизайн сторінок, переставляти елементи та вносити безліч коректив. Тому перед фактичною імплементацією виконують графічне відображення макету додатку та узгоджують його з зацікавленими сторонами. В рамках даної роботи з тих самих міркувань було застосовано інструмент для розробки дизайну Figma [9].

Базова мова програмування на рівні браузеру на момент написання даної роботи є JavaScript. У чистому вигляді використовувати дану мову досить складно через відсутність статичної типізації та готових реалізацій для типових завдань клієнтської частини. Цю нішу покликані заповнити фреймворки клієнтської частини. На момент написання даної роботи існує 3 освновних фреймворки для написання клієнтського коду: Angular [10], React.js [11] та Vue.js [12]. Ці фреймворки хоч і є схожими але все ж відрізняються один від одного. Для виконання даної роботи було обрано React.js через швидкість опанування та розробки. Також цей вибір зумовлено чудовою сумісністю з іншими популярними бібліотеками. Однією з таких бібліотек є Redux. Дана бібліотека дозволяє клієнтській зберігати стан додатку, таким чином оптимізується кількість запитів до серверу, розширюються можливості маніпуляцій даними та зменшується зв’язаність компонентів системи, що призводить до збільшення гнучкості.

Фреймворки і бібліотеки дозволяють створити певний рівень абстракції для розробки та спростити виконання достатньо складних систем надаючи простий у використанні інструментарій. Сучасні браузери та рівень з'єднання дозволяють передавати мережею великі пласти даних, але це не означає що розробник має безвідповідально ставитись до роботи та не використовувати наявні методики та інструментарій для зменшення кількісті даних які необхідно транспортувати. Зменшення об'єму даних призводить до того що користувачі з поганим з'єднанням будуть мати змогу використовувати сервіс з комфортом, а це означає збільшення кількості клієнтів. В тому числі статистика говорить по те, що час завантаження сторінки прямопропорційно впливає на кількість користувачів що чекають до кінця завантаження сторінки. Тому оптимізація даних що передаються є дуже важливим завданням і саме для цього використовуються різного року пакувальники проектів. В рамках даного проекту було використано бібліотеку Gulp [13] через простоту налаштування та можливість розширення.

3.1.3 Вибір технологій для розгортання

У сучасному світі дуже важливо постійно бути доступним та мати мінімальні затримки в реалізації потреб клієнтів. Завдяки хмарним обчисленням у кожного проекту є можливість розгорнути свій додаток у дата центрах по всьому світові. Це призводить до необхідності обирати між наявними провайдерами хмарних обчислень та інструментарієм який використовується для розгортання. Даний інструментарій хоч і не відноситься напряму до програмування, але є достатньо важливим. Важливість зумовлена тим, що саме від якості розгортання додатку та наявних ресурсів залежить праце спроможність та стабільність роботи додатку. Зрозуміло що ідеальний додаток, який здатен вирішувати проблеми користувача, але не є доступним – має таку саму користь як і відсутність цього додатку.

Для розгортання необхідно в першу чергу визначитись з провайдером хмарних обчислень. В даному випадку було обрано AWS [14]. Вибір зумовлений популярністю, користувацькою базою та досвідом використання.

Зрозуміло що для розгортання буде використовуватись віддалений сервер в парі з засобами контейнеризації. Такий підхід дозволяє ізолювати серверну частину від бази даних та серверу що виконує роль розповсюджувача статичних даних та одночасно з чим включити всі необхідні бібліотеки на рівні контейнеру.

Для розгортання клієнтської частини буде використано веб-сервер nginx [15], який вміє розповсюджувати статичні сторінки, виконувати функцію проксі та кешувати запити.

3.2 Структурна та функціональна схеми додатку

Перед початком фактичного кодування важливо розуміти які компоненти входять в систему та як вони взаємодіють між собою. Для цих цілей застосовують схеми різного типу. Фактична ціль всіх схем – спростити реалізацію та комунікацію між командами розробників, зробити так, щоб вони мали однакове уявлення про систему в цілому та її функції.

В рамках даного додатку наявні декілька основних компонентів системи: клієнтська частина та серверна частина які скаладють структурну схему (рис. 3.1.а). Дані сервіси між собою комунікують за допомогою протоколу HTTP. Фактично клієнтська частина відправляє запити до серверної та отримує відповіді. У більшості випадків клієнстька частина просто працює з даними які зберігаються в базі даних і доступ до яких має серверна частина, але це не завжди так – існують операції які мають на меті запустити виконання якогось процесу або просто сповіщення про подію.

Окремо варто відзначити, що для розробки веб-додатків рекомендовано використовувати різні репозиторії для клієнтської та серверної частин додатку, що дозволяє вести окрему розробку та підтримку кожної з них. Крім того, це сприяє зменшенню часу на розробку та полегшує відлагодження проблем у майбутньому.

Функціональна схема (рис. 3.1.б) описує компоненти програмного компоненту та їх взаємодію шляхом використання спеціалізованих позначень. Кожна з підсистем має свої дочірні компоненти, та описує потік даних в цих компонентах.

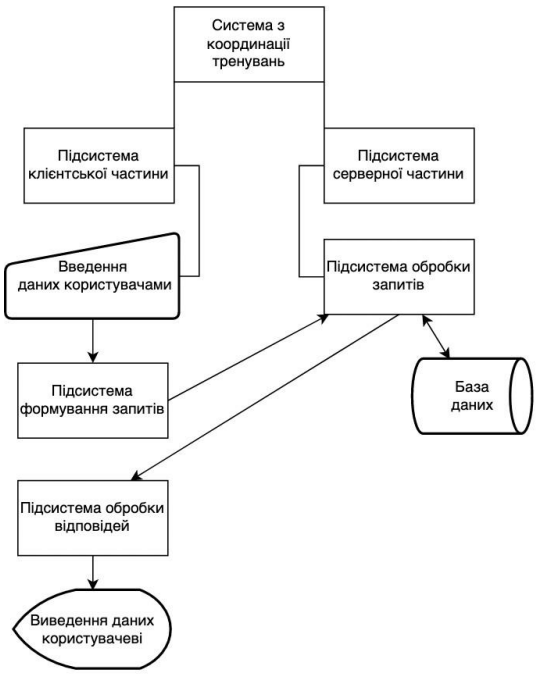


Рис. а) Рис.б)

Рисунок 3.1 а – Структурна схема. б – Функціональна схема

3.3 Фізична реалізація підсистеми бази даних

В попередніх розділах було описано процес вибору бази даних та було пояснено основні чинники які впливають на вибір. В рамках реалізації даного додатку було використано реляційну базу даних Postgresql. Дана база даних має реляційне представлення даних, яке в свою чергу добре відображається за допомогою об’єктної моделі яка в свою чергу являє собою бізнес шар додатку.

Для представлення сутностей бази даних у вигляді об’єктів було використано Hibernate ORM, яка має інтеграцію з Spring Framework та дозволяє швидко налаштувати необхідні конфігурації. Приклад опису сутності у вигляді об’єкту мови Java (рис 3.2) демонструє доцільність використання даної бібліотеки. В даному випадку назва таблиці відповідає назві класу, для генерації ідентифікатора використовується сутність послідовності на рівні бази даних, зв’язки (зовнішні ключі) задаються механізмом анотацій. Таким чином досить легко описати структуру бази даних, та Hibernate згенерує відповідний запит базуючись на описаній структурі.

```java
@Entity
@NoArgsConstructor
@Getter
@Setter
public class DayFeel {
    private static final String GENERATOR = "day_feel_generator";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = GENERATOR)
    @SequenceGenerator(name = GENERATOR, sequenceName = "day_feel_sequence")
    private Long id;
    @ManyToOne
    private User user;
    private Long epochDay;
    @Enumerated(EnumType.STRING)
    private FeelType type;
    private String value;

    Eragoo +1
    public DayFeel(User user, FeelInputDto dto) {
        this.user = user;
        this.epochDay = dto.getEpochDay();
        this.type = dto.getFeelType();
        this.value = dto.getValue();
    }
}
```

Рисунок 3.2 – Опис сутності відчуття користувача

Відповідно схема бази даних (рис 3.3) відображає структуру даних додатку в цілому.

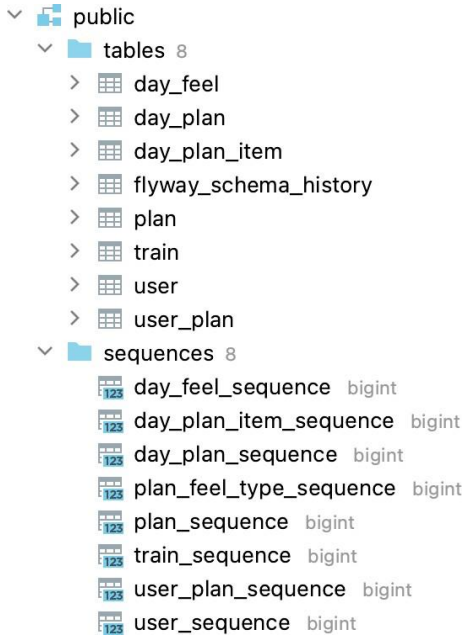


Рисунок 3.3 – Схема бази даних додатку

Всі таблиці описані відповідно до вимог Hibernate. Таблиця flyway_schema_history є службовою та необхідна для відслідковування процесу міграції бази даних – там знаходяться дані про виконані міграції.

Для запитів до бази даних використовується функціональність фреймворку Spring. Компонент даного фреймворку Spring Data JPA [16] має змогу на основі

опису сутності генерувати клас репозиторію, який надає доступ до операцій над даними. Слід зазначити що JPA є лише інтерфейсом, фактичною реалізацією є Hibernate, це є досить важливо при конфігурації та подальшій розробці системи.

3.4 Розробка серверної частини

Базис серверної частини полягає в методології розробки яку диктує Spring Framework. В даному випадку для веб розробки застосовується класичний патерн MVC (рис 3.4).

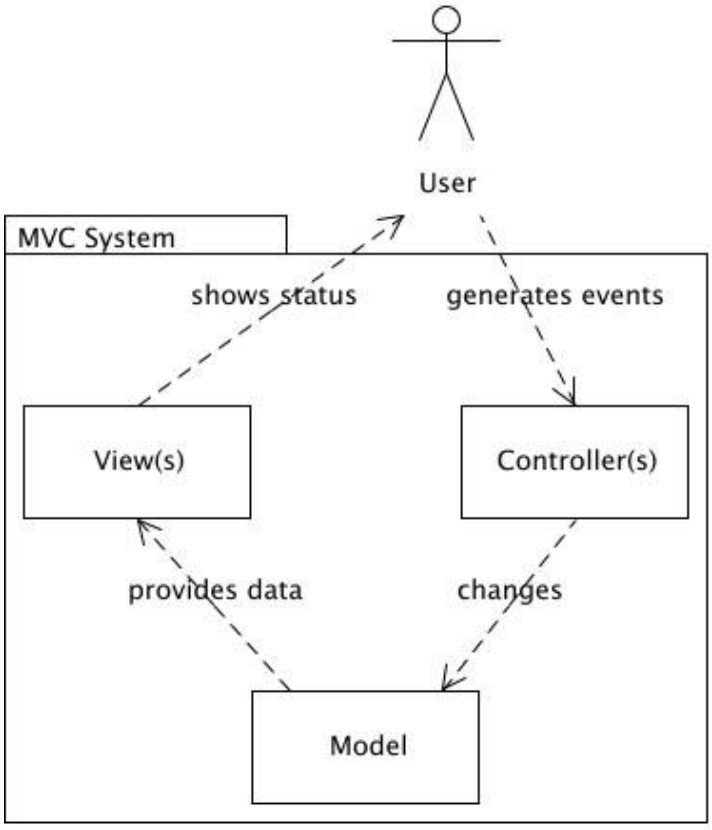


Рисунок 3.4 – Патерн MVC

Суть даного патерну полягає в розділенні відповідальностей. Фактично контролер відповідає за логіку обробки запитів, контролер взаємодіє з моделями. Моделі змінюються під впливом контролерів таким чином змінюючи дані додатку. Ці зміні відображаються в рамках компоненту View який описує саме логіку відображення моделей.

В рамках Spring підходу ми маємо додаткову сутність яку називають сервісом. Лістінг сервісу для тренувань наведено в додатку Б. На рівні сервісу відбувається взаємодія з базою даних, та саме сервіс оперує моделями і повертає

на рівень контролера спеціальні об’єкти які інкапсулюють відображення даних. Таким чином досягається розділення відповідальності між компонентами системи.

Для авторизації та автентифікації користувачів використовується компонент фреймворку Spring з назвою Security. Даний фреймворк використовує механізм фільтрів (рис. 3.5) які складаються в ланцюг відповідальності (схема даного патерну розміщена в додатку В).

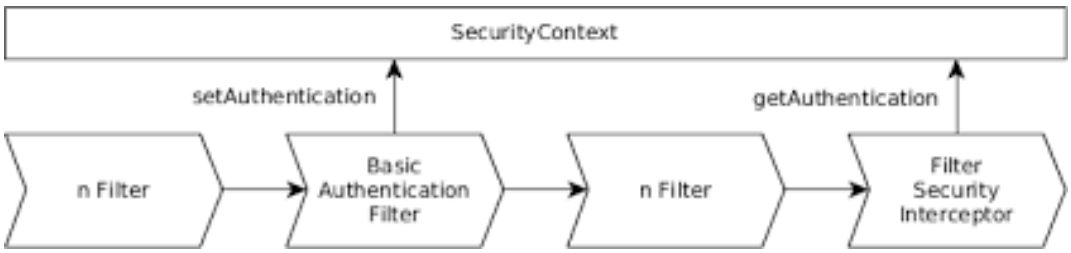


Рисунок 3.5 – Схема автентифікації Spring

Кожен фільтр в рамках ланцюга виконує певні функції. За допомогою фільтрів фреймворк має можливість парсити аргументи посилання та передавати їх на рівень контролера. В даному випадку цікавить саме механізм автентифікації. Для автентифікації клієнт на сторону серверу спеціальний токен в HTTP хедері Authorization. Даний токен є закодованою інформацією про користувача, дані закодовано на основі секрету який знаходиться тільки на стороні серверу, тому згенерувати новий токен без секрету неможливо. Варто зазначити, що токен має обмежений термін життя.

При кожному запиті серверна частина перевіряє наявність токену в хедерах запиту, даний функціонал реалізується на рівні фільтру (рис. 3.6). Якщо токен є і він правдивий, тобто секрет співпадає – даний фільтр формує об’єкт Authentication та встановлює його в SecutiryContext. Наступні фільтри перевіряють наявність об’єкту автентифікації, і якщо його немає – запит не доходить до рівня контролеру та фільтр повертає код 403, який на мові кодів HTTP означає «Unauthorized».

```java
@AllArgsConstructor
@Component
public class JwtAuthorizationFilter extends OncePerRequestFilter {
    public static final String AUTH_TOKEN_PREFIX = "Bearer ";
    public static final String AUTH_HEADER_STRING = "Authorization";

    private TokenProvider tokenProvider;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        String header = request.getHeader(AUTH_HEADER_STRING);

        if (header != null && header.startsWith(AUTH_TOKEN_PREFIX)) {
            String token = header.substring(AUTH_TOKEN_PREFIX.length());
            tokenProvider.parseUser(token)
                    .map(user -> new UsernamePasswordAuthenticationToken(user, null, user.getAuthorities()))
                    .ifPresent(auth -> SecurityContextHolder.getContext().setAuthentication(auth));
        }
        filterChain.doFilter(request, response);
    }
}
```

Рисунок 3.6 – Фільтр авторизації

Система передбачає елемент move-to-earn, який покликаний збільшити інтерес до платформи, та мотивувати спортсменів. Нарахування монет відбувається на основі завантажених тренувань. При завантаженні тренування формується подія, яку оброблює відповідний сервіс да та формулою (див. додаток Г) оброблює інформацію тренування та формує об’єкт транзакції. В залежності від відстані яку подолав спорстмен відбувається нарахування монет. За вістані довші за 10 кілометрів нараховується додатковий бонус. Далі за необхідності показати поточний баланс користувача – транзакції збираються та сума всіх транзакцій є кінцевим значенням балансу.

Описана підсистема серверу надає інтерфейс для взаємодії який буде використовувати клієнтська сторона. Для наочності на зручності даний інтерфейс було додатково описано за допомогою інструменту опису API – Swagger. Даний інструмент чудово інтегрується з використаним фреймворком, та автоматично генерує опис інтерфейсу взаємодії з можливістю тестування.

Таким чином маємо:

– реалізовану об’єктну модель, на основі якої генеруються таблиці бази даних;
– виконаний рівень контролерів, які задають інтерфейс;
– завершений рівень сервісів, які взаємодіють з базою даних та викликаються контролерами;
– описаний рівень репозиторіїв, який генерується засобами ORM;

– реалізований механізм авторизації та автентифікації на основі фільтрів Spring та шаблону ланцюга відповідальності;
– реалізований механізм нарахування нагород;
– описану структуру API серверної частини за допомогою Swagger;

3.5 Реалізація клієнтської частини додатку

Для розробки клієнтської частини додатку було використано фреймворк React.js. Даний фреймворк вводить концепцію віртуального DOM. Це механізм, який використовує React.js для ефективного оновлення сторінок. Коли відбувається зміна контенту на веб-сторінці, React.js не оновлює весь HTML-код відразу, а замість цього, створює віртуальний DOM – копію сторінки, яка зберігається в пам'яті браузера. React змінює дані у віртуальному DOM і порівнює його з реальним DOM. Після того, як React знаходить різницю між віртуальним та реальним DOM, він змінює тільки ті елементи, які потрібно оновити. Даний підхід дозволяє ефективно оновлювати сторінки без необхідності ререндерингу всієї сторінки, що в свою чергу дуже сильно заощаджує ресурси.

В React.js застосовується компонентний підхід до розробки. Фактично розробник має можливість створювати компоненти і потім комбінуючи їх – робити унікальні сторінки. Існує безліч готових бібліотек компонентів для стандартних операцій. У випадку додатку для координації тренувань було використано бібліотеку яка надає можливість працювати з календарем. Фактично вона додає в контекст додатку новий компонент календарю, з яким можна взаємодіяти. В рамках роботи даний компонент було розширено, додано відображення плану тренувань, завантаження тренувань та статистику.

В методології розробки на React є поняття схожу до поняття репозиторії на стороні серверу. Фактично це шар доступу до даних де дані беруться з сторони серверу. Даний шар архітектури який має функції доступу до даних в рамках React.js називають сервісом. Сервіс для автентифікації (рис. 3.7) є типовим прикладом сервісу, роль якого виконувати запити до серверної сторони та відповідним чином повертати результат у відображенні заданої моделі даних.

```
import { instance } from "./instance";

export default class AuthService {
  static async getAuth(payload) {
    return instance.post("auth/basic/token", payload);
  }

  static async getCurrentUser(payload) {
    return instance.get("/users/current", {
      headers: {
      "Authorization": payload
      }
    });
  }
  static async register(payload) {
    return instance.post("users", payload);
  }
  static async restorePassword(payload) {
    return instance.post("auth/basic/password-reset", payload);
  }
}
```

Рисунок 3.7 – Сервіс автентифікації

Відображення даних описується на рівні компонентів (файлів з розширенням .jsx). Компонент відображення інформації про користувача (рис. 3.8) приймає на вхід дані про конкретного користувача та на вихід повертає готовий компонент який може бути доданий на HTML сторінку.

```
import React from 'react'
import { AiOutlineUser } from "react-icons/ai";

const UserInfo = ({username, avatar}) => {
if(username) return (
    <div className="userinfo">
      <div className="userinfo__avatar">
      {avatar ?
      <img src={avatar} alt="Avatar" />
      : <AiOutlineUser/>}
      </div>
      <span className="userinfo__username">{username}</span>
    </div>
  )
else return " ";
}

export default UserInfo
```

Рисунок 3.8 – Компонент користувача

Структура клієнтської частини полягає в реалізації створення плану тренувань, відображення тренувань та статистики – це є головні функції клієнтської частини. Кожна з функцій є компонентом який описується

ізольовано та може бути перевикористаний в різних частинах додатку. Таким чином маємо гнучкий додаток де складові частини є консистентними та описаними лиш одного разу з достатнім рівнем налаштувань. Компонент що задає відображення наведено в додатку Д.

Інтерфейс користувача полягає в необхідності реалізувати функцію створення плану (рис. 3.9.а). Дана функція реалізована на основі компоненту календаря. Елементи плану мають бути додані індивідуально для кожного дня, на день може бути декілька тренувань. До елементу плану можна додати коментарі з вказівками де тренер описує індивідуальні особливості виконання конкретного завдання (рис. 3.9.б).

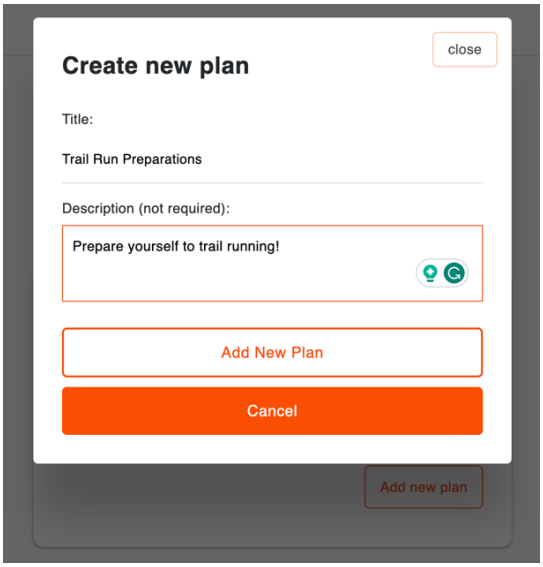


Рис. а)

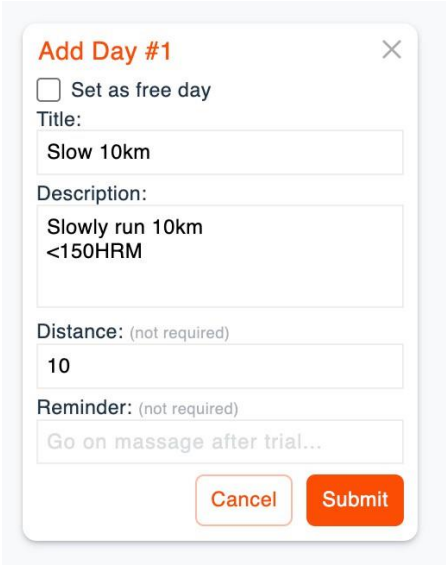


Рис. б)

Рисунок 3.9 а – Створення плану. б – Додавання елементу плану

Крім відображення плану необхідно відображати тренування що виконав користувач, це також відбувається в форматі календарю так як цей спосіб є досить наочним та допомагає відразу на головній сторінці додатку бачити що необхідно зробити користувачеві в рамках означеного плану. Разом з тренуваннями доцільно показувати інформацію про тренування які необхідно виконати в конкретний день відповідно до плану (рис. 3.10). Таким чином користувач буде мати можливість переглядати актуальний стан тренувального процесу.

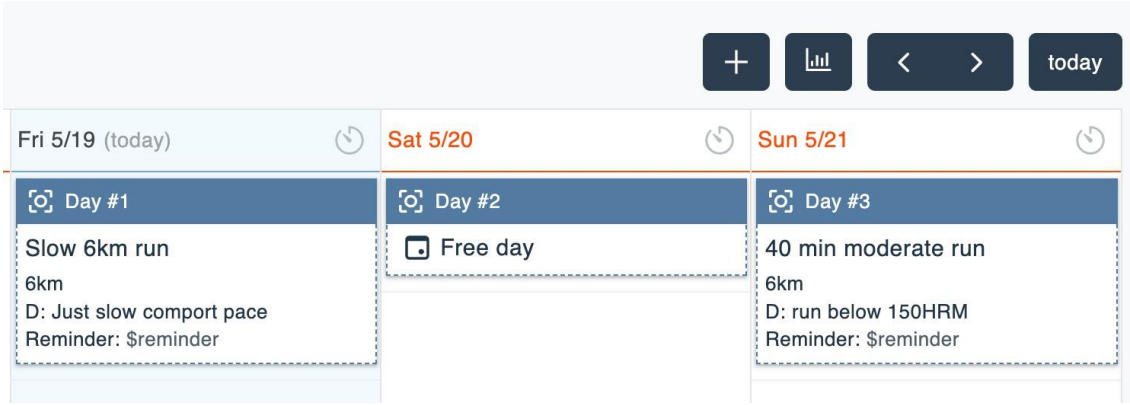


Рисунок 3.10 – Відображення поточних тренувань за планом

Для того щоб слідкувати за прогресом на дистанції необхідно мати інформацію про статистику користувача. Доцільно відображати статистику дистанції за тижнями (рис 3.11). Також є доцільним відображати статистику зміни базових показників спортсмену які від може передати системи. Дані показники включають в себе пульс спокою, вагу та довільний коментар. Так як вага та пульс спокою передаються не більше одного разу за день і є типізованими числами – доцільно показувати графік зміни цих показників

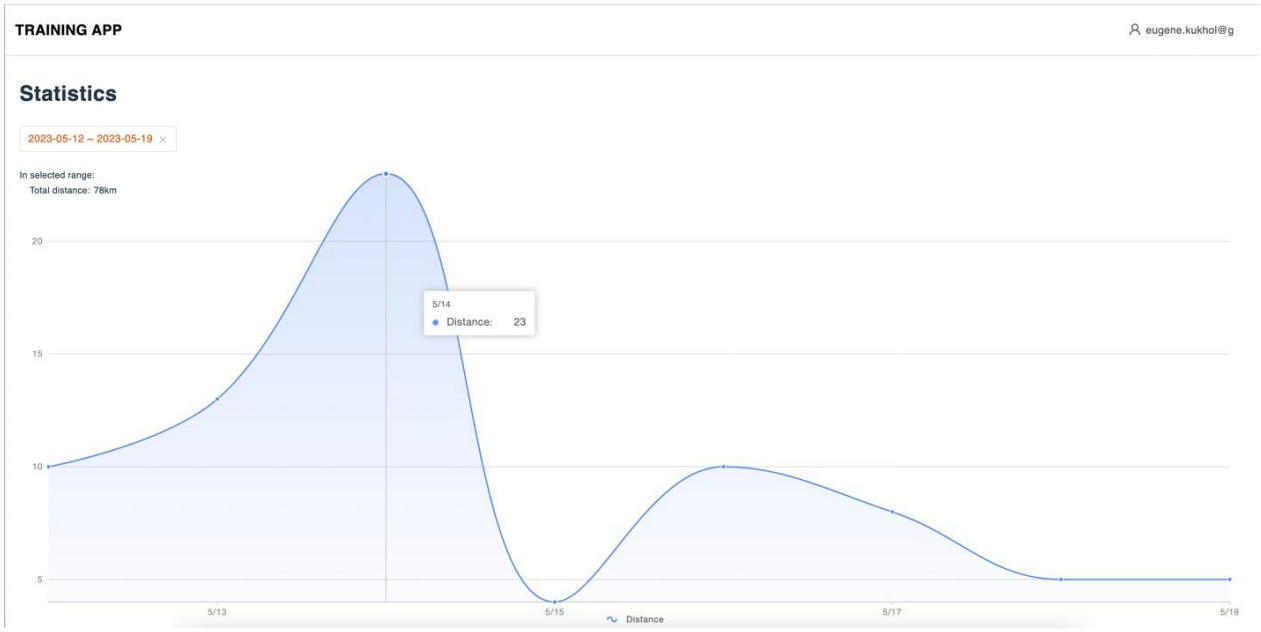


Рисунок 3.11 – Графік денної дистанції за тиждень

Ці дані є дійсно важливими для спортсмену так як вага пропорційно впливає на зусилля які докладає користувач під час тренувань, а пульс спокою сигналізує про рівень стресу організму.

3.6 Тестування додатку

Тестування додатку було розділено на декілька категорій: тестування мануальне за принципом чорного ящику, та автоматичне тестування компонентів серверної частини.

Метод чорного ящику в тестуванні – це методологія тестування програмного забезпечення, яка базується на тестуванні функціональності програми без знання її внутрішньої реалізації. Принцип полягає в тому, що розглядається програма як чорний ящик, в який можна вводити дані і отримувати результати, не знаючи, як саме програма обробляє ці дані. Цей метод зазвичай використовують для тестування функцій, таких як взаємодія з інтерфейсом користувача, обмін даними з сервером, обробка введених даних тощо. При тестуванні методом чорного ящика не звертають увагу на внутрішні деталі реалізації функцій, а зосереджуються на тому, щоб переконатися, що програма поводиться вірно за різних умов введення даних. Один з головних принципів методу чорного ящика полягає в тому, що необхідно тестувати програму на всі можливі вхідні дані, включаючи некоректні дані. Це допомагає знайти можливі помилки та допущені недоліки в програмі.

Таблиця 3.1 – Тестування методом чорного ящику

Тестові сценарії	Актуальний результат	Очікуваний результат
Створення нового плану	План створено та доступно до вибору	План створено та доступно до вибору
Перегляд доступних планів тренувань	Тренування відображаються в рамках календаря	Тренування відображаються в рамках календаря
Додавання тренування	Тренування додається в обраний день	Тренування додається в обраний день
Перегляд статистики	Статистика змінюється відповідно до тренувань	Статистика змінюється відповідно до тренувань

Додавання показників	Показники додаються для кожного дня не більше 1 разу	Показники додаються для кожного дня не більше 1 разу
Перегляд графіку ваги	Графік ваги відповідає переданим показникам	Графік ваги відповідає переданим показникам
Перегляд графіку зміни пульсу спокою	Графік пульсу відповідає переданим показникам	Графік пульсу відповідає переданим показникам
Вибір активного плану	Після вибору план відображається на календарі тренувань	Після вибору план відображається на календарі тренувань
Зміна паролю	Після запиту на зміну паролю користувач отримує повідомлення з кодом підтвердження який він може ввести для створення нового паролю	Після запиту на зміну паролю користувач отримує повідомлення з кодом підтвердження який він може ввести для створення нового паролю

Результати тестування методом чорного ящику (таб. 3.1) свідчать про коректність роботи основних сценаріїв системи.

Автоматичне тестування покликане автоматизувати процес верифікації коректної роботи додатку. Так як Java є строго типізованою мовою програмування, тому багато помилок відловлюються на момент компіляції. Також варто зазначити що немає ніякого сенсу в тестуванні функціональності фреймворку, бо виконати це так само якісно як це зроблено розробниками даного фреймворку – неможливо. Тому тестувати API або рівень взаємодії з базою даних немає так як ці функції бере на себе фреймворк. Завдання протестувати ту частину логіки додатку яка відповідає конкретно за бізнес процеси, та є досить комплексною. Таким компонентом в рамках даної системи є генератор нагород на основі подоланої дистанції. Для того що протестувати даний компонент

можна скористатися фреймворком для тестування Junit [17]. У результаті маємо лістинг (рис. 3.14) який складається з 4 автоматичних тестів, які виконуються при кожній компіляції автоматично. Дані тест-кейси тестують граничні значення дистанції та перевіряють код генерації монет на відсутність логічних помилок. Що є цілком доцільним на відмінну від тестування функціональності фреймворку.

```
@Test
public void testCalculateCoinsEarned_returnsEmptyOptionalIfDistanceIsLessThanMin() {
    Optional<Integer> coinsEarned = CoinsEarnedCalculator.calculateCoinsEarned( distance: 500);
    assertFalse(coinsEarned.isPresent());
}

@Test
public void testCalculateCoinsEarned_returnsBaseCoinsIfDistanceIsBetweenMinAndMax() {
    Optional<Integer> coinsEarned = CoinsEarnedCalculator.calculateCoinsEarned( distance: 5000);
    assertTrue(coinsEarned.isPresent());
    assertEquals(Integer.valueOf( i: 50), coinsEarned.get());
}

@Test
public void testCalculateCoinsEarned_returnsMaxCoinsIfBaseCoinsIsGreaterThanMax() {
    Optional<Integer> coinsEarned = CoinsEarnedCalculator.calculateCoinsEarned( distance: 15000);
    assertTrue(coinsEarned.isPresent());
    assertEquals(Integer.valueOf( i: 100), coinsEarned.get());
}

@Test
public void testCalculateCoinsEarned_returnsBaseCoinsPlusLongRunCoinsIfDistanceIsGreaterThanLongRunDistance() {
    Optional<Integer> coinsEarned = CoinsEarnedCalculator.calculateCoinsEarned( distance: 12000);
    assertTrue(coinsEarned.isPresent());
    assertEquals(Integer.valueOf( i: 60), coinsEarned.get());
}
```

Рисунок 3.14 – Автоматичні тести

Виходячи з результатів автоматичного тестування можна зробити висновок що компонент генерації монет працює коректно.

Висновки до розділу

При практичній реалізації додатку було уточнено набір бібліотек та детально пояснено мотивацію вибору та можливості обраних інструментів. Було побудовано діаграми які описують систему на високому рівні для того щоб в подальшому декомпозувати підсистеми на фактичні задачі реалізації. Було описано реалізації серверної частини та кожного шару архітектури. Було описано реалізацію клієнтської частини та обґрунтовано актуальність обраного підходу. Було протестовано систему на відсутність помилок засобами автоматичного тестування та методом чорного ящику.

ВИСНОВКИ

Під час проектування та розробки проекту кваліфікаційної роботи було розроблено додаток для координації спортивних занять з елементами move-to-earn. Актуальність даного додатку полягає у відсутності рішень з подібним функціоналом на сучасному ринку. Даний додаток має безліч переваг перед потенційними конкурентами до яких належить можливість створення планів тренувань, використання існуючих планів, механізм нагород за досягнення цілей.

Для реалізації даного веб-застосунку було пройдено наступні етапи:

1. Проаналізовано ринок схожих застосунків та найближчих конкурентів.
2. Для кожного схожого додатку на ринку було знайдено переваги та недоліки.
3. За допомогою аналізу переваг та недоліків схожий додатків було сформовано вимоги до програмного продукту базуючись на ринку ПЗ.
4. На основі розроблених вимог було спроектовано систему що є веб-додатком.
5. Відповідно до розробленої архітектури було обрано необхідні інструменти та бібліотеки для реалізації та обґрунтовано даний вибір.
6. Використовуючи розроблену архітектуру та вимоги було розроблено серверну частинку додатку та клієнтську частину.
7. Було реалізовано автоматичні тести та перевірено систему методом чорного ящика.

В ході реалізації було використано наступні інструменти для розробки:

– spring Framework, Hibernate ORM, Flyway, Swagger для реалізації серверної частини;
– react.js, Figma, Redux, Gulp, Redux Saga для реалізації клієнтської частини;
– docker, Nginx, Traefik, AWS EC2, ECR, SES для розгортання в хмарі;

Додаток є самостійним продуктом який можна використовувати за основним призначенням. Було реалізовано весь основний функціонал і методи безпеки.

Подальший розвиток додатку передбачає реалізацію наступного функціоналу:

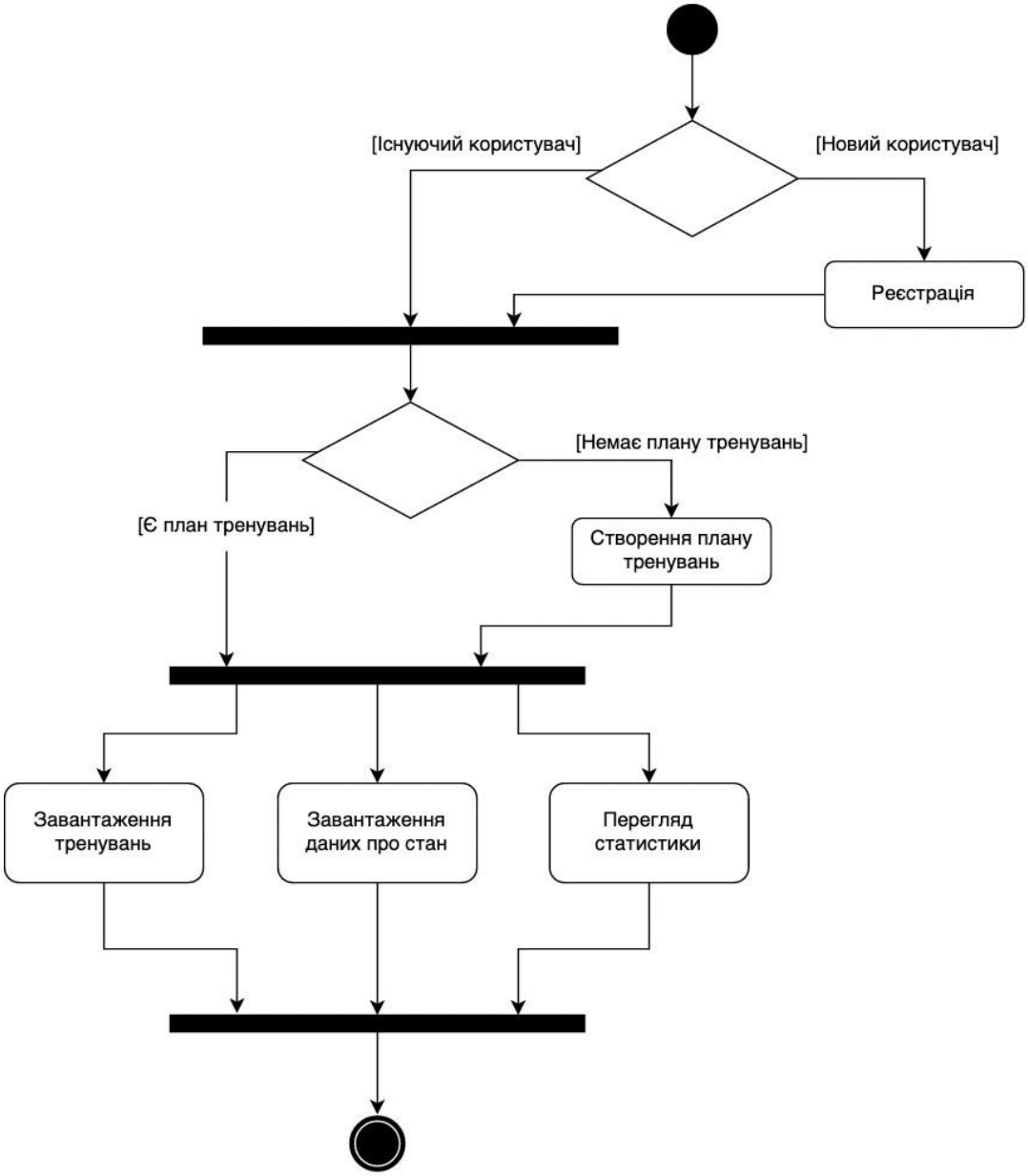
- інтеграція зі Strava для автоматичного завантаження тренувань на платформу;
- розширення типів тренувань які підтримує додаток;
- реалізація публічного профілю користувача та можливості об’єднуватись в групи для соціальної складової;
- інтеграція з програмним забезпеченням розумних гаджетів для отримання даних про пульс спокою та вагу;
- введення механізму платної підписки на основі інтеграції зі Stripe або Fondy;

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільний додаток «Runkeeper» [Електронний документ]: https://apps.apple.com/ua/app/runkeeper-%D0%BF%D1%80%D0%BE%D0%B1%D0%B5%D0%B6%D0%BA%D0%B8-%D1%81-gps/id300235330?l=ua.
2. Мобільний додаток «Runner's Log» [Електронний документ]: https://apps.apple.com/us/app/runners-log/id500565287.
3. Мобільний додаток «Running Diary» [Електронний документ]: https://apps.apple.com/us/app/running-diary/id1125720449.
4. Мобільний додаток «Strava» [Електронний документ]: https://apps.apple.com/ua/app/strava-%D0%B1%D0%B5%D0%B3%D0%B8%D0%B2%D0%B5%D0%BB%D0%BE%D1%81%D0%BF%D0%BE%D1%80%D1%82-gps/id426826309?l=ua.
5. Spring Framework home page [Електронний документ]: https://spring.io/.
6. Домашня сторінка бази даних «PostgreSQL» [Електронний документ]: https://www.postgresql.org/.
7. Домашня сторінка проекту «Hibernate ORM» [Електронний документ]: https://hibernate.org/orm/.
8. Домашня сторінка проекту міграцій «Flyway» [Електронний документ]: https://flywaydb.org/.
9. Домашня сторінка проекту «Figma» [Електронний документ]: https://www.figma.com.
10. Домашня сторінка фреймворку «Angular» [Електронний документ]: https://angular.io/.
11. Домашня сторінка фреймворку «React.js» [Електронний документ]: https://uk.reactjs.org/.

12. Домашня сторінка фреймворку «Vue.js» [Електронний документ]: https://ua.vuejs.org/.
13. Домашня сторінка фреймворку «Gulp.js» [Електронний документ]: https://gulpjs.com/.
14. Домашня сторінка хмари «AWS» [Електронний документ]: https://aws.amazon.com/?nc1=h_ls.
15. Домашня сторінка веб-серверу «Nginx» [Електронний документ]: https://nginx.org/en/.
16. Документація фреймворку «Spring Data JPA» [Електронний документ]: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.
17. Документація фреймворку для тестування «JUnit» [Електронний документ]: https://junit.org/junit5/.

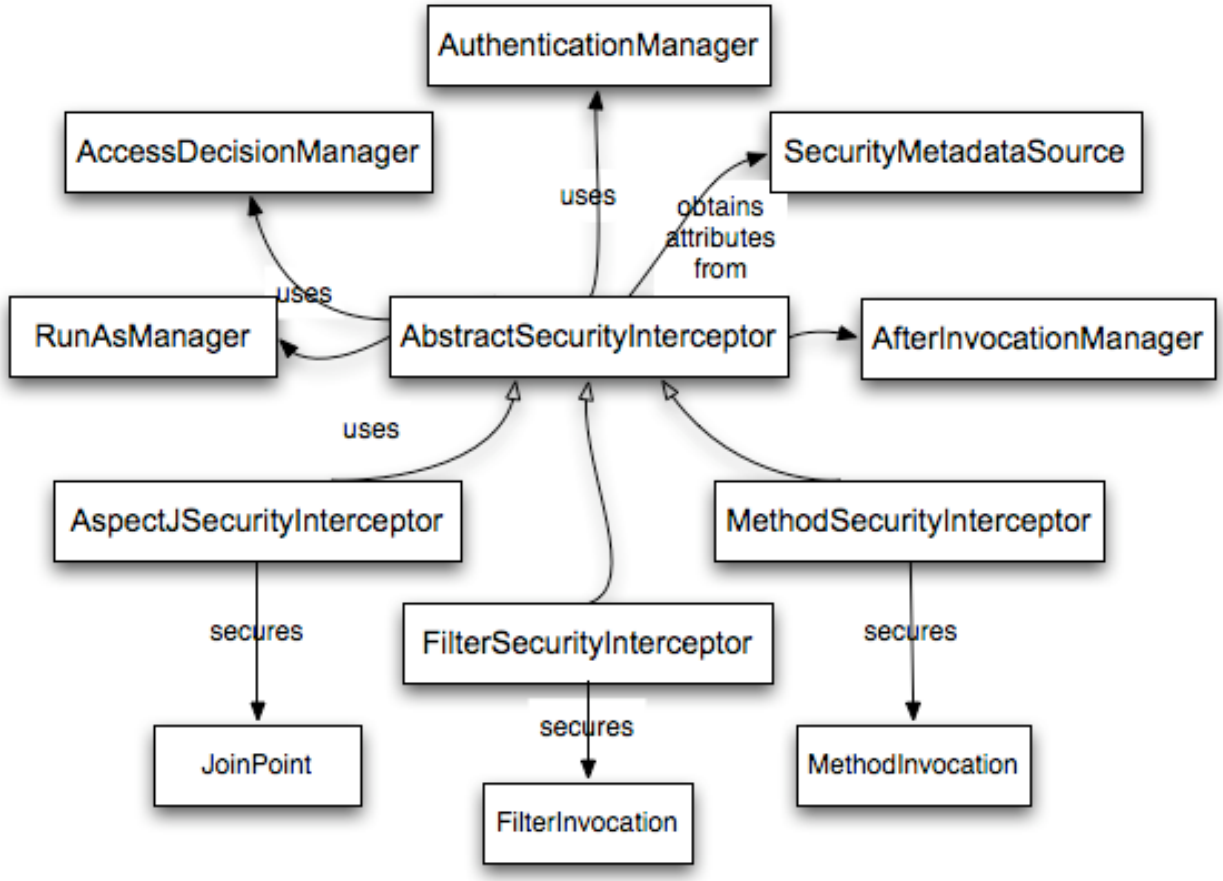
ДОДАТОК А


[Існуючий користувач]
[Новий користувач]
Реєстрація
[Немає плану тренувань]
[Є план тренувань]
Створення плану тренувань
Завантаження тренувань
Завантаження даних про стан
Перегляд статистики

ДОДАТОК Б ЛІСТІНГ СЕРВІСУ ТРАНЗАКЦІЙ

```
@AllArgsConstructor
@Slf4j
@Service
public class TransactionService {
    private TransactionRepository transactionRepository;
    private UserRepository userRepository;

    Eragoo
    @EventListener(TrainingCreatedEvent.class)
    @Transactional
    public void onTrainingCreated(TrainingCreatedEvent event) {
        User user = userRepository.findById(event.getUserId())
                .orElseThrow(() -> new RuntimeException("User not found"));


        CoinsEarnedCalculator.calculateCoinsEarned(event.getDistance())
                .ifPresent(coins -> createTransaction(user, coins));
    }

    Eragoo
    private void createTransaction(User user, Integer coins) {
        Transaction transaction = new Transaction();
        transaction.setAmount(new BigDecimal(coins));
        transaction.setOwner(user);
        transactionRepository.save(transaction);
        log.info("Transaction created: {}", transaction);
    }

    Eragoo
    @Transactional(readOnly = true)
    public BigDecimal getBalance(Long userId) {
        return transactionRepository.findAllByOwnerId(userId).stream()
                .reduce(BigDecimal.ZERO, (a, b) -> a.add(b.getAmount()), BigDecimal::add);
    }
}
```

ДОДАТОК В АРХІТЕКТУРА SPRING SECURITY



ДОДАТОК Г ЛІСТІНГ КАЛЬКУЛЯТОРУ МОНЕТ

```java
public class CoinsEarnedCalculator {
    private static final int MIN_DISTANCE_FOR_COINS = 1000;
    private static final double MIN_DISTANCE_FOR_LONG_RUN = 10000.0;
    private static final double COINS_PER_METER = 0.001;
    private static final int MAX_COINS_PER_RUN = 100;
    private static final int COINS_FOR_LONG_RUN = 10;

    /**
     * Calculates the amount of coins a user earns based on the distance they run.
     * The user earns 1 coin for every 100 meters they run, up to a maximum of 100 coins per run.
     * In addition, if the user runs more than 5 kilometers (10000 meters), they earn an additional 10 coins.
     * @param distance The distance the user has run, in meters
     * @return The amount of coins the user earns for this run
     */
    public static Optional<Integer> calculateCoinsEarned(double distance) {
        // Check if the user has run far enough to earn any coins at all
        if (distance < MIN_DISTANCE_FOR_COINS) {
            return Optional.empty();
        }

        // Calculate the base amount of coins the user earns based on distance
        int baseCoins = (int) Math.floor(distance / 100.0) * (int) Math.round(COINS_PER_METER * 100.0);

        // Cap the amount of coins earned at the maximum for this run
        int coinsEarned = Math.min(baseCoins, MAX_COINS_PER_RUN);

        // Check if the user has run far enough to earn the additional long run coins
        if (distance >= MIN_DISTANCE_FOR_LONG_RUN) {
            coinsEarned += COINS_FOR_LONG_RUN;
        }

        // Return the final amount of coins earned
        return Optional.of(coinsEarned);
    }

    /**
     * Formats a decimal number to a string with a fixed number of decimal places.
     * This is used to format the coins earned value as a string for display purposes.
     * @param value The decimal value to format
     * @param decimalPlaces The number of decimal places to display
     * @return The formatted string representation of the decimal value
     */
    private static String formatDecimal(double value, int decimalPlaces) {
        BigDecimal bd = new BigDecimal(value).setScale(decimalPlaces, RoundingMode.HALF_UP);
        return bd.toString();
    }
}
```

ДОДАТОК Д ЛІСТІНГ СТОРІНКИ ПЛАНІВ

```
import React from "react";
import { BsCalendar3, BsAward, BsChatRightText } from 'react-icons/bs';

const PlanPage = () => {
 return (
  <div className="plan_page">
   <div className="plan_content">
    <h1 className="plan_title">Getting Started</h1>
    <div className="plan_blocks">
     <div className="plan_block">
      <h2 className="plan_block__title">Do It Yourself</h2>
      <p className="plan_block__desc">Track, analyze and plan on the web.</p>
      <div className="plan_block__icon">
        <BsAward/>
      </div>
      <button className="plan_btn btn">Start Now</button>
     </div>
     <div className="plan_block">
      <h2 className="plan_block__title">Get On A Plan</h2>
      <p className="plan_block__desc">Select an expertly designed training plan.</p>
      <div className="plan_block__icon">
        <BsCalendar3/>
      </div>
      <button className="plan_btn btn">Start Now</button>
     </div>
     <div className="plan_block">
      <h2 className="plan_block__title">Work With A Coach</h2>
      <p className="plan_block__desc">Get matched with a certified coach.</p>
      <div className="plan_block__icon">
        <BsChatRightText/>
      </div>
      <button className="plan_btn btn">Start Now</button>
     </div>
    </div>
   </div>
  </div>
 );
};

export default PlanPage;
```