МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Катерина НЕСТЕРЕНКО
“____”_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
“МАГІСТР”

Тема: «Інтерактивна мапа з маршрутами для катання на роликах»

Виконавець: Прокопенко Максим Сергійович

Керівник: к.т.н, доцент Борковська Любов Олексіївна

Нормоконтролер: асистент Кравченко Ольга Сергіївна

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Прокопенка Максима Сергійовича

1. Тема дипломної роботи: «Інтерактивна мапа з маршрутами для катання на роликах»
затверджена наказом ректора від 29.09.2023 р. № 1994/ст

2. Термін виконання проекту: з 02.10.2023 р. до 24.12.2023 р.

3. Вихідні данні до проекту: програмний продукт розробити у вигляді веб-додатку використовуючи Java Enterprise Edition.

4. Зміст пояснювальної записки:
   1. Аналіз існуючих рішень на ринку веб-додатків для трекінгу маршрутів для катання.
   2. Визначення вимог до бек-енд підсистеми та інтерфейсу користувача.
   3. Реалізація застосунку інтерактивної мапи з маршрутами для катання на роликах.

5. Перелік обов'язкових слайдів презентації:
   1. Огляд швидкості цифровізації та пояснення необхідності розробки
   2. Важливість трекінгу часу при розробці
   3. Огляд архітектурних підходів та їх переваги і недоліки
   4. Огляд структурних схем та діаграм
   5. Тестування системи як важливий пункт валідації перед релізом повної версії

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Складання та затвердження графіку роботи дипломного проектування Написання 1 розділу, представлення керівнику	02.10.2023-08.10.2023	
2.	Попередній друк 1 розділу та допоміжних сторінок (чорновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел, 1-й нормо-контроль.	08.10.2023-20.10.2023	
3.	Написання 2 розділу, представлення керівнику	21.10.2023-30.10.2023	
4.	Написання 3 розділу, представлення керівнику	01.11.2023-14.11.2023	
5.	Написання висновку, представлення керівнику	15.11.2023-30.11.2023	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	30.11.2023-03.12.2023	
7.	Проходження нормо-контролю, перевірка на антиплагіат, перепліт пояснювальної записки.	04.12.2023-10.12.2023	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	10.12.2023-15.12.2023	
9.	Отримання відгуку керівника, рецензії.	11.12.2023-17.12.2023	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, CD-R з електронними копіями ПЗ, презентації, відгук керівника, рецензія) в папці	18.12.2023-24.12.2023	

7. Дата видачі завдання 29.09.2023

Керівник: к.т.н. доцент Любов БОРКОВСЬКА

Завдання прийняв до виконання: Максим ПРОКОПЕНКО

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Інтерактивна мапа з маршрутами для катання на роликах»: 93 с., 26 рис., 2 табл., 26 інформаційних джерел.

ВЕБ-СИСТЕМА, ОБРОБКА ДАНИХ, МАПА, REST API, РОЛИКОВІ КОВЗАНИ, ВЕБ-СТОРІНКА

Об’єкт розробки – інтерактивна мапа.

Мета роботи – розробити інтерактивний додаток, що дозволить спростити створення та використання маршрутів для катання.

ABSTRACT

Explanatory note to the thesis "Interactive map with routes for roller skating": 93 pages, 26 pictures, 2 tables, 26 information sources.

WEB SYSTEM, DATA PROCESSING, MAP, REST API, ROLLER SKATES, WEB PAGE

Property development – interactive map.

Purpose – to develop an interactive application that will simplify the creation and use of skating routes.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ООП – об'єктно-орієнтоване програмування;

ОС – операційна система;

ПЗ – програмне забезпечення;

РОС – підтвердження концепту;

MVC – Model View Controller;

НОЕ – експеремент проведений всередені (команди, компанії);

ІК – інтерфейс користувача;

СРТ – Compton;

API – набір визначень і протоколів для створення та інтеграції прикладного програмного забезпечення;

BE – back end (бєк-енд);

FE – front end (фроент-енд);

POI – точка інтересу (на мапі);

JOI – інтерфейс орієнтований на Java;

PIMP – програмний інтерфейс мапи позицій;

CI/CD – безперервна інтеграція та розгортання.

ВСТУП

У гонитві за «цифровізацією» Україна активно використовує трансформаційну силу Інтернет-технологій, що розвиваються, і широкого використання персональних пристроїв. Створення онлайн-присутності, будь то через веб-додатки, веб-сайти чи мобільні програми, стало необхідністю для служб і організацій будь-якого розміру та походження. Цей зсув означає відхід від традиційних бюрократичних процесів у бік більш ефективних і сучасних рішень. Завдяки пристрасті до катання на роликах і зростаючому попиту на доступну інформацію було розроблено додаток під назвою «інтерактивна мапа для на роликах».

Сучасна веб-система повинна відповідати суворим критеріям, включаючи швидкість, надійність, толерантність до навантаження і, що важливо, масштабованість, забезпечуючи стабільність серед зростаючої бази користувачів і нових функцій. Головною метою цього дипломного проекту є успішна розробка веб-системи, яка бездоганно взаємодіє з кінцевим користувачем інтерактивної мапи для катання на роликах. Підхід до проектування охоплює архітектурні та прикладні методи, призначені для розробки підсистеми обробки даних у системі карти.

Окреслені завдання проектування включають аналіз предметної області та методологій, пов’язаних із розробкою серверної підсистеми, з подальшою концептуалізацією та подальшим впровадженням внутрішньої підсистеми обробки даних для інтерактивної карти катання на роликах. Практична значущість цього дипломного проекту полягає в успішній розробці серверної підсистеми в системі карт, призначеної для забезпечення інтерактивного та ефективного досвіду для ентузіастів катання на роликах.

Задачі проектування:

1. Проаналізувати предметну область та методології розробки веб-систем.

2. Спроектувати веб-систему інтерактивної мапи для катання на роликах.

3. Реалізувати веб-систему інтерактивної мапи для катання на роликах.

Практична цінність дипломного проекту полягає у тому, що була розроблена інтерактивна мапад ля катання на роликах, що полегшить життя роллерам.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДОЛОГІЇ РОЗРОБКИ ВЕБ-ДОДАТКІВ

1.1. Аналіз предметної області поставленого завдання та визначення його актуальності

Предметною областю є розробка інтерактивної мапи для катання на роликах. Любителі катання на роликових ковзанах часто прагнуть захоплюючих відчуттів ковзання новими та захоплюючими маршрутами. Незалежно від того, чи це для відпочинку, фітнесу чи просто любові до природи, пошук ідеального маршруту для катання на роликах може покращити враження.

На щастя, сучасні технології проклали шлях для інтерактивних карт, розроблених спеціально для катання на роликах, пропонуючи безліч переваг, які покращують це популярне заняття. Інтерактивні карти: найкращий друг любителя роликових ковзанів.

Кожен спортсмен може уявити собі, що має доступ до вичерпної карти, яка не лише демонструє маршрути для катання на роликових ковзанах, але й надає важливу інформацію, щоб зробити пригоду безперешкодною та приємною. Ці інтерактивні карти мають такі функції, як налаштування маршруту на основі рівня навичок, детальні описи місцевості, відстеження відстані, профілі висоти та маркери для визначних місць.

Незалежно від того, чи мова йде про новачка, чи про досвідченого спортсмена, що шукає для себе виклик, даний застосунок дозволить обрати ідеальний маршрут.

Дотик соціальних мереж

Можна ще більше покращити життя спортсменів, запровадивши функції соціальних медіа на цих інтерактивних картах. Будь-кому приємно мати спільноту ентузіастів катання на роликових ковзанах, які діляться улюбленими маршрутами, враженнями та порадами. Інтеграція функцій

соціальних мереж дозволяє любителям роликів взаємодіяти один з одним, створюючи відчуття спільності та товариства.

Переваги інтеграції соціальних мереж Розбудова спільноти:

Об’єднуючи любителів роликових ковзанів через спільну платформу, інтеграція соціальних медіа розвиває почуття спільноти. Ентузіасти можуть ділитися своїми пригодами, відкриттями та навіть організовувати групові прогулянки. Це сприяє формуванню згуртованої спільноти, яка захоплюється катанням на роликах.

Оновлення в режимі реального часу:

Функціональні можливості соціальних мереж дозволяють оновлювати маршрути, погодні умови та будь-які потенційні перешкоди в режимі реального часу. Любителі, які їздять на роликових ковзанах, можуть бути в курсі подій і відповідним чином планувати свої прогулянки, забезпечуючи плавну та безпечну роботу.

Мотивація та підбадьорення:

За допомогою лайків, коментарів і обміну досвідом любителі роликів можуть мотивувати та заохочувати один одного. Позитивна взаємодія надихає людей досягати цілей у фітнесі та впевнено досліджувати нові маршрути. Пропаганда здорового способу життя: активна участь у спільноті катання на роликових ковзанах сприяє здоровому способу життя. Обмін досягненнями, прогресом і досвідом спонукає інших прийняти катання на роликових ковзанах як заняття фітнесом, сприяючи загальному добробуту.

Поєднання катання на роликових ковзанах і технологій відкрило захоплюючі можливості, а інтеграція функцій соціальних медіа в інтерактивні карти розширює ці можливості.

Далі представлений список переваг, які може дати подібна система з інтеграцією соц-мереж:

1. Відкриття та різноманітність маршруту

Інтерактивні карти, підкріплені інтеграцією в соціальні мережі, представляють величезну кількість маршрутів для катання на роликах. Обмін

маршрутами та досвідом спонукає любителів роликових ковзанів досліджувати нові місцевості та різноманітні ландшафти. Незалежно від того, чи то мальовнича прибережна стежка чи складна міська траса, спільні маршрути надихають любителів роликових ковзанів розширювати свої горизонти та підкорювати свіжі траси.

2. Підвищення рівня безпеки

Безпека має першорядне значення для будь-якого активного відпочинку. Завдяки функціям соціальних мереж роликові ковзани можуть ділитися оновленнями в реальному часі про умови маршруту, потенційні небезпеки або несподівані блокпости. Такий обмін інформацією підвищує безпеку, дозволяючи іншим відповідно планувати свої сеанси катання на роликах і бути в курсі будь-яких потенційних проблем, з якими вони можуть зіткнутися на обраному маршруті.

3. Експертні думки та поради

Спільнота катання на роликових ковзанах складається з людей з різними рівнями досвіду. Інтеграція соціальних медіа дозволяє досвідченим катанням на роликах ділитися своїми порадами, прийомами та прийомами з новачками. Початківці можуть отримати пораду, керівництво та рекомендації щодо відповідних маршрутів та обладнання, сприяючи культурі навчання серед ролерів.

4. Групова координація та заходи

Планування групової пригоди на роликових ковзанах стає легким завдяки функціям соціальних мереж, інтегрованим в інтерактивні карти. Ролери можуть координувати зустрічі, створювати події та запрошувати інших ентузіастів приєднатися. Це не тільки зміцнює зв’язок між громадою, але й заохочує групову діяльність, сприяючи відчуттю єдності та веселощів у сфері катання на роликах.

5. Відзначення досягнень і віх

Кожна подорож на роликових ковзанах — це досягнення саме по собі, а інтеграція соціальних мереж дозволяє любителям роликових ковзанів

відзначати свої важливі досягнення. Незалежно від того, чи це досягнення певної відстані, освоєння нового трюку чи проходження складного маршруту, спільнота може вболівати та мотивувати одне одного, перетворюючи кожне досягнення на спільну перемогу.

6. Надихаюче піклування про навколишнє середовище

Ділячись захоплюючими фотографіями та враженнями від катання на роликових ковзанах, ентузіасти також можуть надихнути екологічну свідомість. Зображення краси природи під час подорожі заохочує глибше цінувати довкілля та може виховати почуття відповідальності за його збереження та захист.

Тож, інтеграція функціональних можливостей соціальних медіа в інтерактивні карти, призначені для катання на роликових ковзанах, перетворює цю, на перший погляд, рекреаційну діяльність у сферу колективного захоплення та взаємодії. Це розвиває активну спільноту, сприяє безпеці, заохочує до пригод і, зрештою, збагачує досвід катання на роликах. Коли люди катаються на роликових ковзанах, вони не просто переміщуються на колесах, а рухаються у процвітаючій віртуальній спільноті, посилюючи свою пристрасть до цього захоплюючого виду спорту.

Любителі роликових ковзанів отримують значну користь від інтерактивних карт, легко досліджуючи нові маршрути, адаптовані до їхніх уподобань. Незалежно від того, досвідчений скейтер чи новачок, повна карта гарантує можливість переміщатися різними місцевостями та відкривати приховані перлини. Безпека є головним у будь-якій фізичній активності, і катання на роликових ковзанах не є винятком. Інтерактивні карти дозволяють користувачам визначати та вибирати маршрути відповідно до їхнього рівня навичок, висвітлюючи потенційні небезпеки та дозволяючи скейтерам розумно планувати свої маршрути.

Створення почуття спільності є життєво важливим у будь-якому виді спорту, і катання на роликових ковзанах не є відмінністю (Рис. 1.1) [1]. Інтерактивні карти забезпечують платформу для скейтерів, щоб ділитися

своїми улюбленими маршрутами, рекомендувати мальовничі місця та спілкуватися з іншими ентузіастами, створюючи мережу підтримки та додаючи соціального елементу до досвіду. Для тих, хто використовує катання на роликах як фітнес-заняття, інтерактивні карти є цінним інструментом для відстеження прогресу. Користувачі можуть записувати маршрути, відстежувати відстані та контролювати швидкість, додаючи ігровий елемент до катання на роликах і мотивуючи людей випробувати себе. Інтерактивні карти також полегшують планування подій і координацію в спільноті ролерів.

Рис. 1.1 Люди обєднані ролер-спортом

Слугуючи центральним центром для обміну подробицями подій і місць зустрічі, ці карти забезпечують більш плавну координацію та безперебійну навігацію визначеними маршрутами під час групових катань або змагань. У світі катання на роликах, який постійно розвивається, роль інтерактивних карт важко переоцінити. Ці цифрові інструменти значною мірою сприяють

задоволенню та безпеці ентузіастів катання на роликових ковзанах, починаючи від покращення дослідження та сприяючи появі спільності.

Оскільки технології продовжують розвиватися, можливості для інтерактивних карт у спільноті ролерів безмежні, що обіцяє ще більш захоплююче та взаємопов'язане майбутнє для цього динамічного виду спорту. Зав'яжіть ковзани, досліджуйте світ і дозвольте інтерактивній карті стати вашим провідником у захоплюючій подорожі катанням на роликах.

У динамічній сфері катання на роликових ковзанах інновації та конкуренція є головними рушійними силами, які рухають поле до постійного зростання та розвитку. Взаємодія між цими двома елементами не тільки покращує досвід для ентузіастів, але й сприяє створенню середовища прогресу та просування. Інновації в катанні на роликах не обмежуються дизайном самих ковзанів; воно поширюється на технологію, яка підтримує діяльність.

Інтерактивні карти, наприклад, являють собою технологічну інновацію, яка змінила те, як катаються на роликових ковзанах і відчувають навколишнє середовище.

Інтеграція доповненої реальності, відстеження маршруту в режимі реального часу та інші функції не тільки покращують користувацький досвід, але й відкривають двері для подальшого дослідження творчих можливостей у спорті. Крім того, інновації викликають креативність у розробці нових маневрів, прийомів і стилів. Оскільки скейтери розширюють межі того, що вважається можливим на роликових ковзанах, уся спільнота отримує вигоду від багатого гобелена різноманітних технік і підходів.

Цей постійний пошук інновацій не тільки робить спорт свіжим і захоплюючим, але й залучає нових учасників, сприяючи зростанню спільноти ролерів. Конкуренція, з іншого боку, служить каталізатором для вдосконалення та досконалості. Коли фігуристи беруть участь у товариських або змагальних змаганнях, вони мотивовані вдосконалювати свої навички, прагнучи досягти нових висот.

Прагнення до перемоги сприяє розвитку культури безперервного вдосконалення, де учасники отримують натхнення вдосконалювати свої техніки, досліджувати інноваційні стратегії та, зрештою, сприяти розвитку роликових ковзанів у цілому. Крім індивідуального рівня, конкуренція забезпечує платформу для індустрії роликових ковзанів, щоб продемонструвати свої досягнення.

Виробники прагнуть виробляти передове обладнання, яке може дати спортсменам конкурентну перевагу, (Рис. 1.2) стимулюючи цикл інновацій та вдосконалень. Це приносить користь не лише професійним спортсменам, але й охоплює ширшу спільноту, оскільки нові технології та дизайни стають доступнішими. Глобальна сцена змагань з роликових ковзанів також діє як об’єднуюча сила, об’єднуючи ентузіастів з різних куточків світу.

Такий обмін ідеями та стилями посилює загальну різноманітність спорту, оскільки фігуристи черпають натхнення один від одного, створюючи багатий гобелен технік і підходів.

Рис. 1.2 Спортсмени як-от спідскейтери часто використовують спеціальні маршрути, які складно знайти на новому місці

Підсумовуючи аналіз предметної області представлений вище, можна впевнено стверджувати, що синергія між інноваціями та конкуренцією має вирішальне значення для загального розвитку роликових ковзанів. Інновації спонукають до еволюції обладнання, технологій і техніки, тоді як конкуренція діє як тигель для досягнення досконалості, спонукаючи спортсменів до постійного вдосконалення. Ця динамічна взаємодія не тільки покращує досвід для окремих ентузіастів, але й гарантує, що катання на роликових ковзанах залишається яскравим спортом, що розвивається, залучаючи нових учасників і сприяючи почуттю спільності в глобальному масштабі. Оскільки галузь продовжує охоплювати інновації та конкуренцію, майбутнє роликових ковзанів обіцяє бути відзначеним хвилюванням, різноманітністю та постійним прогресом.

Взагалі, роликові ковзани відійшли на другий план популярності після двотисячних років, замість цього значного розповсюдження набули скейти та самокати. Проте, завдяки подібним інноваціям та створенню спілки однодумців, можна досягнути «єффекту сніжного кома», коли сфера стане розвиватись по енерції, завдяки вдало створеному фундаменту для такого росту. Розробка подібних додатків спрямована на відродження інтереса до цього спорту, так як подібна трансформація не можлива без фактору цікавості, що може представити такий додаток. Новачки у хоббі матимуть значно більше впевненості у перших кроках, так і розумітимуть у яку сторону розвиватись.

1.2. Дослідження відомих практик інженерії вимог щодо розробки програмного забезпечення.

Оцінка вимог передбачає ретельну оцінку програмного продукту з метою узгодження його з потребами кінцевого користувача чи замовника [19]. Враховуючи те, що кінцевий користувач представляє різноманітну групу, яка охоплює різні глобальні місця, веб-система вимагає ретельного

проектування та повинна пройти суворий процес затвердження з клієнтом для вирішення та виправлення будь-яких недоліків програми.

Ініціювання процесу розробки вимог починається з побудови моделі для визначення вимог. Важливо негайно визначити основні функціональні та нефункціональні вимоги, характерні для цієї програми.

Функціональні вимоги пояснюють список дій, виконуватимуться обраним додатком. Ключові дії, що виконуватиметеся у інтерактивній мапі – це:

- Перегляд існуючих мап для обрання маршруту
- Перегляд коментарів для основних точок маршруту
- Створення власного акаунту
- Створення свого маршруту
- Оцінка маршруту за пятибальною шкалою

Нефункціональні вимоги допомагають визначити критерії для оцінки якості програмної системи/програмного засобу. Сюди входять апаратні та програмні вимоги до програми (визначення платформ, необхідних для роботи системи), вимоги до інтерфейсу та операційні вимоги (вимоги користувача).

Веб-додаток має єдину апаратну вимогу, а саме:

- будь-який пристрій із стабільним доступом до мережі Інтернет.

Основною програмною вимогою для передбачуваної веб-системи є розробка продукту з використанням мови програмування Java EE, що включає Spring Boot для внутрішньої розробки.

Розробка інтерфейсу використовуватиме мову програмування JavaScript (React.js) і систему дизайну AntDesign. Завдяки своїм унікальним характеристикам ця веб-система займає особливу позицію серед програмних продуктів такого роду, що підкреслює першорядну важливість її актуальності в цифровому середовищі. Що стосується вимог до інтерфейсу, адаптивна веб-система буде містити елементи 2D-графіки, включаючи зображення, таблиці та текст. Інтерфейс і статичні графічні об’єкти утворюють основні

графічні компоненти з вибраною анімацією, включеною за допомогою React.js.

Взаємодія між користувачами та програмним засобом призначена як для мобільних пристроїв, так і для персональних комп'ютерів. Основна вимога для успішного графічного інтерфейсу полягає в тому, щоб він був інтуїтивно зрозумілим для нових користувачів, пропонуючи безперебійні можливості для будь-якої взаємодії, передбаченої розробником. Експлуатаційні вимоги, що визначають принципи взаємодії з середовищами або системами, є вирішальними. Це показники робочого часу, захист даних і досягнення якості, дотримуючись рекомендацій використовуваного стандарту. З точки зору користувача, програма повинна забезпечувати доступність, надійність, зручність, простоту, масштабованість і цілісність даних.

Примітно, що веб-система повинна бути доступною для користувачів з різними пристроями, залишатися функціональною під час помилок, бути зручною для користувача, легко масштабуватися без компромісу продуктивності та підтримувати цілісність даних під час обробки інформації. У світлі цих вимог механіку програми та основні процеси можна приблизно спланувати, узгоджуючи з передбачуваним соціальним розвитком та спрощенням стосунків між людьми. Життєвий цикл інструменту необмежений, з можливістю оновлення до нових версій. Поки зберігається потреба в таких веб-додатках, актуальність інструменту зберігатиметься протягом багатьох років. Робоче середовище програми охоплює будь-який пристрій користувача зі стабільним відкритим доступом до мережі Інтернет.

У рамках цих суворих вимог можна окреслити механіку програми, пропонуючи розуміння основних процесів і бізнес-логіки, які керуватимуть її функціональністю. Очікувані користувачі цієї програми різноманітні, охоплюючи громадян, які користуються послугами різноманітних веб-програм. Основна функція програми полягає в сприянні соціальному

розвитку, спрощенні взаємодії між окремими людьми та полегшенні життя спортсменів.

Дуже важливо підкреслити глобальне охоплення та потенційний вплив програми. Здатність обслуговувати користувачів у всьому світі, надаючи культурно збагачені та актуальні дані, позиціонує веб-систему як каталізатор глобального обміну ідеями та інформацією. Завдяки повному об’єднанню 2D-графіки, інтуїтивно зрозумілих інтерфейсів і адаптивного дизайну програма стає універсальним інструментом, доступним для користувачів із різним досвідом, досвідом і рівнями технологічної підготовки. Постійний розвиток веб-системи залежить від її здатності залишатися актуальною та реагувати на нові технологічні тенденції.

У міру того, як цифровий ландшафт змінюється, адаптивність додатків і прихильність до задоволення потреб користувачів будуть мати першорядне значення. Забезпечення відповідності оновлень і новіших версій очікуванням користувачів зміцнить статус програми як динамічної та тривалої платформи у сфері веб-програм. По суті, ця веб-система постає не просто як технологічний витвір, а як культурний фасилітатор, який об’єднує людей, долає кордони та робить внесок у ширший ландшафт цифрових інновацій.

У контексті передбачуваної веб-системи, важливість відсутності простою набуває ще більш виразної ролі. Спортсмени, які покладаються на дані в реальному часі та навігаційні функції, які надає програма, залежать від її постійної доступності для безперебійної роботи. Будь-який час простою створює не тільки перешкоди для їхньої рекреаційної діяльності, але й потенційні проблеми з безпекою, особливо під час навігації незнайомими місцевостями.

Для любителів роликових ковзанів, які використовують додаток, інформація в реальному часі про маршрути, потенційні небезпеки та навігаційні вказівки є найважливішою. Надійна та стабільно доступна система стає надійним компаньйоном, покращуючи загальний досвід катання на роликах і сприяючи безпеці користувачів. Відсутність простоїв гарантує,

що любителі роликових ковзанів можуть впевнено покладатися на актуальну інформацію в додатку, дозволяючи їм досліджувати нові маршрути та місцевості, не турбуючись про раптові збої. Крім того, спільнота катання на роликах часто активна в різних часових поясах і місцях.

Здатність веб-системи підтримувати безперервність роботи стає об'єднуючим фактором, полегшуючи зв'язок, обмін маршрутами та координацію подій серед ролерів у всьому світі. Таким чином, відсутність простоїв не тільки покращує індивідуальний досвід катання на роликових ковзанах, але й сприяє створенню зв'язаної та підтримувальної глобальної спільноти ролерів. По суті, надійність і відсутність простоїв веб-системи є не просто технічними міркуваннями, а важливими компонентами для безпеки, задоволення та спільної участі любителів роликових ковзанів. Забезпечуючи безперервний доступ до важливої інформації, програма стає незамінним інструментом, який дає змогу кататися на роликових ковзанах досліджувати, підключатися та керувати своїми подорожами з впевненістю та надійністю.

Зрештою, дотримання встановлених вимог забезпечує стабільну та ефективну роботу веб-серверів. У свою чергу, ця програма готова надавати мільйонам людей у всьому світі дані, необхідні для розвитку культури, посилюючи її значну роль у цифровому ландшафті.

1.3. Дослідження практичних особливостей архітектурного проектування веб-додатків

Внутрішню структуру програми розроблено з централізованим підходом, який зосереджує всю програмну логіку на сервері. У свою чергу, користувачі отримують доступ до інтерфейсу користувача через невелику програму, фундаментальний компонент, який зберігається з моменту створення мережі HTTP [9].

По суті, архітектуру веб-додатку можна порівняти з ретельно сконструйованим «скелетом» або макетом, що окреслює складну взаємодію між різними компонентами, такими як логіка додатків, системи проміжного

програмного забезпечення, інтерфейси користувача та бази даних. Ця зв'язана структура взаємодії полегшує одночасну роботу кількох програм, сприяючи синергетичному середовищу (рис. 1.3).

Коли користувач відкриває веб-сторінку, сервер негайно реагує, відправляючи певні дані в браузер користувача відповідно до запиту користувача. Точніше, веб-клієнт, також відомий як агент користувача, має здатність вимагати веб-ресурси або загальновідомі веб-документи, такі як HTML, JSON або PDF-файли з веб-сервера. Цей простий процес розгортається плавно, результатом якого є представлення необхідної інформації. Після цього початкового обміну даними починається динамічна взаємодія між користувачем і веб-сайтом. По суті, архітектура веб-додатків організовує гармонійний обмін даними та інструкціями між пристроєм користувача та сервером.

Ця співпраця, якій сприяє реакція сервера на запити користувачів, є основою залучення користувачів і взаємодії з веб-програмою. Постійний характер цієї моделі забезпечує швидкий пошук і відображення інформації, сприяючи ефективній і зручній взаємодії для людей, які орієнтуються в цифровому просторі.

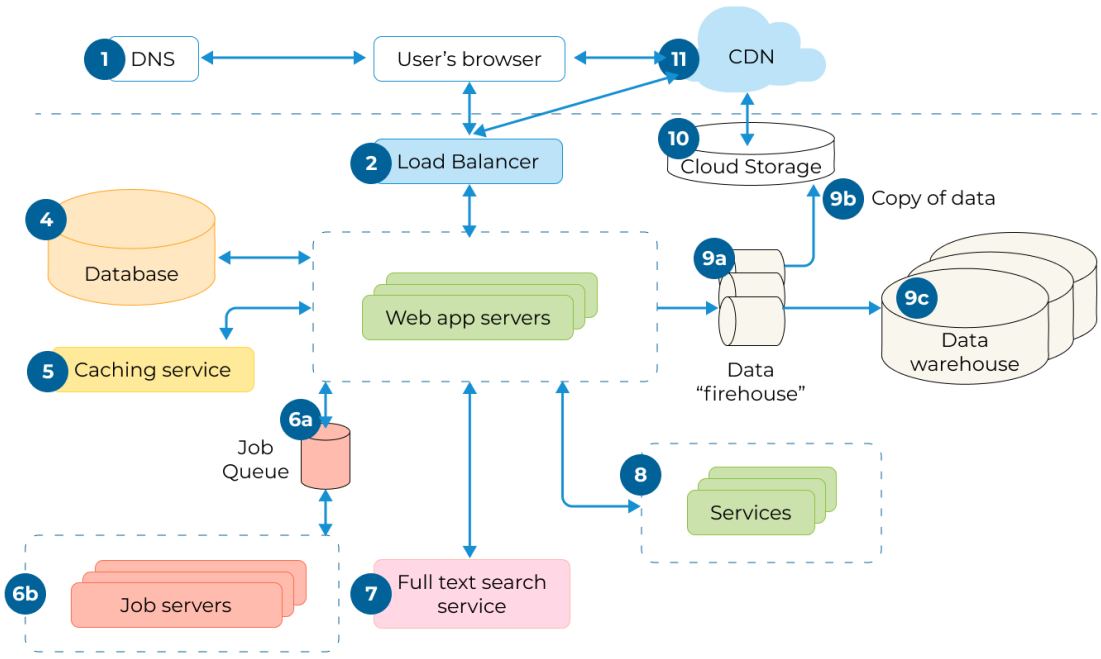

Рис. 1.3 Архітектура сучасного веб-додатку

Продовжуючи дослідження архітектури веб-додатків, дуже важливо заглибитися в складну динаміку, яка визначає відносини між користувачами та сервером. Коли користувачі відкривають веб-сторінку, швидкість реагування сервера стає стрижнею в забезпеченні бездоганної роботи. Обмін даними між сервером і браузером користувача — це не просто односпрямований потік; це закладає основу для постійного діалогу, який характеризує сучасну мережеву взаємодію. Агент користувача, або веб-клієнт, діє як посередник, який організовує цю взаємодію. Він ініціює запити на певні веб-ресурси або документи з сервера, запускаючи низку дій, які завершуються представленням запитуваної інформації.

Цей процес є основоположним для взаємодії з користувачем, оскільки ефективність обміну даними впливає на сприйняту швидкість і реакцію веб-програми. Крім того, адаптивність і розширюваність архітектури веб-додатків відіграють ключову роль у задоволенні різноманітних потреб користувачів. З появою різноманітних веб-технологій, від сценаріїв динамічного вмісту до асинхронного зв’язку, сучасні веб-програми можуть забезпечити багатий і динамічний досвід користувача. Включення фреймворків, таких як React.js, як згадувалося в попередньому обговоренні, є прикладом постійних зусиль для покращення складності та інтерактивності інтерфейсів користувача. Окрім взаємодії клієнт-сервер, сучасна архітектура веб-додатків часто включає системи проміжного програмного забезпечення, які полегшують зв’язок між різними компонентами.

Ці системи проміжного програмного забезпечення діють як мости, забезпечуючи безпроблемний обмін даними між різними елементами програми, такими як інтерфейс користувача та база даних. Цей взаємозв’язок має вирішальне значення для цілісної функціональності програми та гарантує, що оновлення або зміни в одному компоненті не порушують роботу всієї системи.

Архітектура сучасних веб-додатків найчастіше розділяється на два основоположних підходи – так званий «моноліт» та мікро сервісна архітектура (рис. 1.4). Обидва підходи мають свої значні переваги та недоліки та мають свої місця використання.

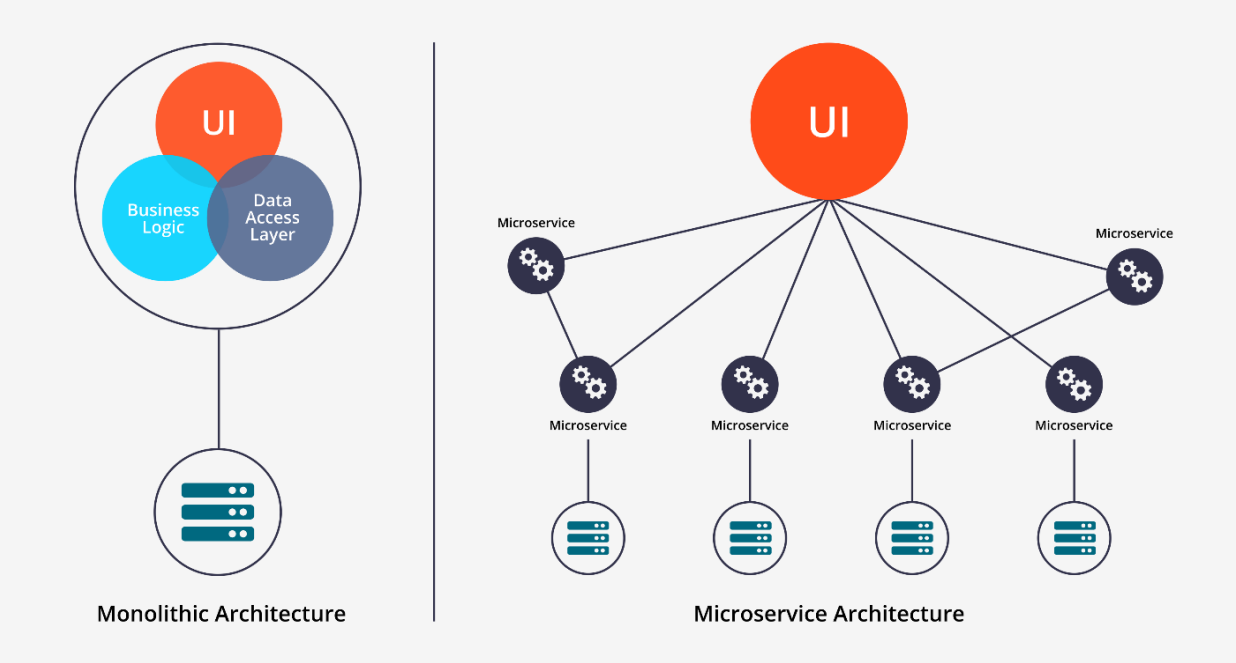

Рис. 1.4 Абстрактна візуалізація двох типів архітектури

1.3.1. Монолітна архітектура

По-перше, проаналізовано концепцію монолітної архітектури. Цей архітектурний підхід виник у більш ранню епоху, зумовлений різними факторами, включаючи відносно скромний масштаб додатків, розроблених у той час. Монолітна архітектура, породжена обмеженнями попереднього ландшафту розвитку, демонструє відмінні характеристики, які значно впливають на її практичність та ефективність.

Визначальною рисою монолітної програми є її унітарна структура, де всі елементи, логіка та компоненти об’єднуються в тісно взаємопов’язану сутність. Ця архітектурна єдність, хоча й спрощує певні аспекти розробки, створює проблеми, пов’язані з розгортанням, часом запуску та масштабованістю. Запуск монолітної програми вимагає спеціального середовища, вимога, що випливає з взаємозалежності її різних компонентів.

Велика тривалість запуску, яка часто займає кілька годин, відображає необхідність ініціалізації та синхронізації цих взаємопов’язаних елементів.

Ця характеристика не тільки впливає на взаємодію з користувачем, але також впливає на можливість запуску програми на окремих робочих станціях розробника. Останнє вимагає потужних робочих станцій, додаючи попит на інфраструктуру, який можна вважати непомірно високим, особливо для розробників, які працюють з обмеженими ресурсами. Крім того, тонкощі монолітної архітектури створюють проблеми на етапі розробки. Спільні зусилля стають складними, оскільки вся програма знаходиться в одному місці, що ускладнює роботу різних команд розробників незалежно один від одного над окремими компонентами. Відсутність модульності може перешкоджати гнучкості та уповільнювати процес розробки, особливо в динамічних середовищах, що швидко розвиваються.

Незважаючи на труднощі, монолітна архітектура має свої переваги, особливо в сценаріях, де пріоритетом є простота та зрозумілість. Це залишається життєздатним вибором для невеликих програм або проектів, де накладні витрати на керування розподіленими системами можуть переважити переваги. Однак у міру того, як технологія розвивається, а масштаб і складність додатків зростають, альтернативні архітектурні підходи, такі як мікросервіси, набувають популярності, пропонуючи рішення для проблем, пов’язаних з монолітними архітектурами.

1.3.2. Переваги монолітної архітектури

Хоча монолітна архітектура створює труднощі, вона також має невід’ємні переваги, які роблять її придатним вибором для певних сценаріїв. Однією з помітних переваг є його простота та легкість розробки. Згуртованість монолітної програми оптимізує процес розробки, полегшуючи розробникам концептуалізацію, проектування та реалізацію всієї системи в єдиній структурі. Ця простота може бути перевагою, особливо для невеликих проектів або програм, де складність розподіленої системи може бути

непотрібною. Крім того, централізований характер монолітної архітектури спрощує процеси налагодження та тестування. Оскільки всі компоненти знаходяться в одному місці, виявлення та вирішення проблем стає більш простим.

Розробники можуть легко відстежувати потік логіки та виявляти потенційні помилки, скорочуючи час і зусилля, необхідні для налагодження. Ця простота допомагає підтримувати цілісність програми, особливо на ранніх стадіях розробки, коли швидка ітерація та вдосконалення є вирішальними. Ще однією помітною перевагою є легкість розгортання. Оскільки всі компоненти об’єднані разом, розгортання монолітної програми передбачає керування одним блоком, а не координацію кількох незалежних модулів.

Ця простота в розгортанні може призвести до швидших циклів випуску, особливо в сценаріях, де необхідні часті оновлення. Це дозволяє розробникам ефективніше впроваджувати зміни або запроваджувати нові функції без складнощів, пов’язаних із координацією кількох служб. Крім того, ресурсоефективність монолітної архітектури очевидна, особливо в середовищах з обмеженими ресурсами.

Оскільки вся програма працює в одному середовищі виконання, менше накладних витрат, пов’язаних із керуванням зв’язком і координацією між розподіленими службами. Така ефективність використання ресурсів може бути вигідною для невеликих команд або проектів з обмеженими бюджетами та інфраструктурою.

Окрім властивої їй простоти та раціоналізованих процесів розробки, монолітна архітектура пропонує додаткові переваги, які сприяють її придатності в певних контекстах. Однією з помітних переваг є легкість вертикального масштабування.

Коли виникає потреба у підвищенні продуктивності, вертикальне масштабування за рахунок збільшення ресурсів існуючого сервера стає простим рішенням. Така простота масштабування може бути корисною для

додатків, де очікуються передбачувані моделі зростання та додаткові потреби в ресурсах. Крім того, уніфікована кодова база монолітної програми сприяє згуртованості та одноманітності стандартів кодування. Розробники, які працюють над різними аспектами програми, мають спільне розуміння кодової бази, що полегшує співпрацю та мінімізує криву навчання для нових членів команди. Ця уніфікованість сприяє ремонтопридатності та забезпечує постійну якість коду в усій програмі.

Монолітна архітектура також спрощує процес керування версіями та оновлення. Оскільки вся програма об’єднана як єдине ціле, розгортанням оновлень або поверненням до попередніх версій можна керувати більш узгоджено. Це спрощує процес контролю версій, полегшуючи для розробників підтримку та випуск оновлень програмного забезпечення без складнощів, пов’язаних із керуванням декількома компонентами, які незалежно розгортаються.

Надійна екосистема та доступність інструментів, що підтримують монолітну архітектуру, роблять її привабливою. Багатий набір інструментів розробки, фреймворків і бібліотек добре підходить для створення та підтримки монолітних програм. Ця обширна екосистема підтримки спрощує процес розробки, пропонуючи розробникам велику кількість ресурсів для підвищення продуктивності та задоволення різноманітних вимог до програм.

Підсумовуючи, переваги монолітної архітектури поширюються на її простоту вертикального масштабування, уніфікованість стандартів кодування, оптимізоване керування версіями та оновлення, а також надійну екосистему інструментів. Хоча це може бути рішенням не для всіх додатків, особливо для тих, які мають складні вимоги, що швидко розвиваються, його переваги роблять його прагматичним вибором для конкретних сценаріїв, демонструючи, що придатність вибору архітектури залежить від узгодження архітектури з конкретними потребами та характеристиками проект під рукою.

1.3.3. Недоліки монолітної архітектури

Хоча монолітна архітектура пропонує гарну кількість своїх переваг, важливо визнати її обмеження та недоліки, що робить її потенційно менш придатною для певних сценаріїв. Помітним недоліком є відсутність модульності та масштабованості. У міру розширення програми монолітна структура може діяти як вузьке місце, перешкоджаючи незалежному масштабуванню компонентів. Відсутність модульності ускладнює впровадження нових технологій або оновлення окремих функцій без впливу на всю систему, що може призвести до збільшення зусиль і часу на розробку.

Інший суттєвий недолік полягає в зростанні складності з часом. У міру того, як додаток росте, кодова база стає складною і її складно підтримувати. Надто складні монолітні архітектури сприйнятливі до «спагетті-коду», де залежності тісно переплетені, що ускладнює ізоляцію та вирішення проблем без небажаних наслідків. Ця складність може перешкоджати продуктивності розробників, перешкоджати співпраці та підвищувати ймовірність виявлення помилок. Крім того, процес розгортання монолітних програм може бути громіздким, особливо у великомасштабних проектах. Розгортання всієї програми як одного блоку може призвести до простою під час оновлень, що вплине на взаємодію з користувачем. Це відрізняється від більш сучасних архітектур, таких як мікросервіси, де окремі компоненти можна оновлювати незалежно, що забезпечує більш плавні та гнучкі стратегії розгортання.

Крім того, монолітні архітектури можуть зіткнутися з проблемами, пов’язаними з різноманітністю технологій. У технологічному ландшафті, що постійно розвивається, монолітному підходу може бути важко адаптуватися до нових інфраструктур або мов. Оновлення технологічного стеку монолітної програми може бути величезним, включаючи модифікації всієї кодової бази та потенційно призводячи до проблем із сумісністю та подовжених термінів розробки.

Також, тісно пов’язані компоненти в монолітній архітектурі можуть створювати перешкоди для командної співпраці. Коли кілька команд

розробників зосереджуються на різних аспектах програми, можуть виникати конфлікти через залежність від спільних ресурсів. Ця взаємозалежність може обмежити автономію команди та створити додаткові витрати на спілкування, особливо у великомасштабних проектах.

Ще одним недоліком монолітної архітектури є проблема використання ресурсів. Оскільки всі компоненти спільно використовують ту саму інфраструктуру та ресурси, може виникнути неефективність, що призведе до неоптимального використання обчислювальної потужності та сховища. Відсутність ізоляції ресурсів може призвести до ситуацій, коли певні частини програми можуть відчувати дефіцит ресурсів, тоді як інші мають надлишкову ємність. Ця неефективність стає більш помітною в міру масштабування програми, потенційно впливаючи на загальну продуктивність і економічну ефективність.

Питання технологічного блокування викликає занепокоєння для монолітних архітектур. Оскільки вся програма тісно інтегрована та залежить від певного стеку технологій, перехід на нові технології стає складним і високим завданням. Це може обмежити гнучкість команд розробників, обмежуючи їх здатність досліджувати інноваційні рішення або адаптуватися до мінливих галузевих стандартів.

Міркування безпеки також мають значення для монолітних архітектур. У разі вразливості безпеки або зламу в одному компоненті вся система опиниться під загрозою. Централізований характер монолітних програм ускладнює реалізацію детальних заходів безпеки, потенційно піддаючи всю програму вразливостям. Це викликає серйозні занепокоєння, особливо в середовищах, де захист даних і конфіденційність є найважливішими. Крім того, життєвий цикл розробки в монолітних архітектурах може зіткнутися з проблемами, пов’язаними з масштабуванням команд розробників. У міру того, як додаток росте, координація зусиль команди, що розширюється, стає складною.

Відсутність чітких меж між компонентами може призвести до конфліктів під час одночасної розробки, впливаючи на контроль версій і збільшуючи ймовірність проблем інтеграції.

В результаті, хоча монолітна архітектура має такі переваги, як простота та легкість розробки, важливо ретельно зважити її недоліки, охоплюючи такі питання, як модульність, проблеми масштабованості, складність, громіздке розгортання та проблеми адаптації. Вибір між монолітною та альтернативною архітектурою має відповідати конкретним потребам, масштабу та очікуваній еволюції проекту, забезпечуючи стійкий та ефективний підхід до розвитку.

1.3.5. Переваги мікро сервісної архітектури

На відміну від монолітної архітектури, архітектура мікросервісів пропонує кілька переваг, які вирішують багато проблем, пов’язаних з аналогом. Однією з помітних переваг є покращена масштабованість. Мікросервіси розбивають програму на менші сервіси, які можна розгортати незалежно, що дозволяє кожному компоненту незалежно масштабуватися відповідно до конкретних потреб. Така деталізація масштабованості забезпечує оптимальне використання ресурсів, дозволяючи організаціям ефективно розподіляти ресурси та ефективніше реагувати на мінливі вимоги.

Архітектура мікросервісів сприяє покращеній ізоляції помилок. Оскільки кожен мікросервіс працює незалежно, збій в одному компоненті не обов’язково призводить до збою в системі. Ця ізоляція гарантує, що помилки містяться в ураженому мікросервісі, запобігаючи каскадним збоям і підвищуючи загальну стійкість системи. Крім того, мікросервіси сприяють технологічній різноманітності. Кожен мікросервіс можна розробляти та розгортати за допомогою різних технологій, що дозволяє організаціям вибирати найбільш підходящі інструменти для конкретних завдань.

Ця гнучкість дає змогу командам розробників застосовувати новітні технології, адаптуватися до галузевих тенденцій і оптимізувати кожен

мікросервіс для його цільового призначення без шкоди для цілісності всієї системи. Значною перевагою є легкість безперервної інтеграції та безперервного розгортання (CI/CD) із мікросервісами. Кожен мікросервіс можна розробляти, тестувати та розгортати незалежно, що дозволяє частіше оновлювати та випускати. Ця гнучкість у процесі розробки дозволяє організаціям надавати нові функції або швидко вирішувати проблеми, підвищуючи загальну швидкість розробки та час виходу на ринок.

Мікросервіси підтримують розподілену модель розробки, що дозволяє кільком командам розробників одночасно працювати над різними мікросервісами. Цей підхід до паралельної розробки підвищує автономію команди, прискорює терміни розробки та сприяє ефективній співпраці. Команди можуть зосередитися на конкретних бізнес-функціональних можливостях, не будучи обмеженими тонкощами монолітної кодової бази.

Ще одна значна перевага архітектури мікросервісів полягає в її здатності сприяти покращеному використанню ресурсів. На відміну від монолітних додатків, де всі компоненти мають спільну інфраструктуру, мікросервіси дозволяють більш ефективно використовувати обчислювальні ресурси. Кожен мікросервіс можна розгорнути в інфраструктурі, яка найкраще відповідає його вимогам, оптимізуючи розподіл ресурсів і потенційно знижуючи операційні витрати. Мікросервіси також підтримують незалежне керування даними. Кожен мікросервіс може мати власну базу даних або рішення для зберігання даних, що дозволяє командам вибирати бази даних, які відповідають їхнім конкретним потребам. Така децентралізація управління даними підвищує автономність даних, дозволяючи командам приймати незалежні рішення щодо зберігання, пошуку та обробки даних.

Мікросервіси дозволяють краще ізолювати помилки з точки зору проблем з продуктивністю. Якщо один мікросервіс відчуває вузьке місце або сповільнення продуктивності, це не обов’язково вплине на всю систему. Ця ізоляція гарантує, що проблеми з продуктивністю в одному мікросервісі не

поширюватимуться каскадно на всю програму, зберігаючи послідовну та надійну взаємодію з користувачем. Крім того, архітектура мікросервісів підтримує покращене керування версіями та зворотну сумісність.

Оскільки мікросервіси працюють незалежно, команди можуть оновлювати та розгортати нові версії окремих мікросервісів, не впливаючи на всю систему. Ця гнучкість керування версіями спрощує керування оновленнями, забезпечуючи більш плавний перехід для користувачів і мінімізуючи збої.

Мікросервіси також добре відповідають принципам DevOps і гнучкої розробки. Модульний і незалежний характер мікросервісів дозволяє командам розробки та операцій працювати разом і частіше вносити зміни. Ця гнучкість має вирішальне значення в швидкоплинних середовищах розробки, де важлива швидка адаптація до мінливих вимог. Крім того, мікросервіси можуть підвищити стійкість програми до збоїв і зовнішніх залежностей. Завдяки впровадженню шаблонів стійкості, таких як автоматичні вимикачі та резервні механізми, команди можуть гарантувати, що навіть у разі збою мікросервісу чи зовнішнього сервісу вся система залишається працездатною та швидко реагує.

Архітектура мікросервісів пропонує переваги у використанні ресурсів, незалежному управлінні даними, ізоляції помилок у зв’язку з проблемами продуктивності, керуванні версіями та зворотній сумісності, узгодженні з DevOps і гнучкими практиками, а також підвищеній стійкості.

У той час як впровадження мікросервісів вимагає ретельного розгляду проблем, таких як збільшення операційної складності та витрати на зв’язок, ці переваги сприяють зростанню популярності мікросервісів як сучасного та ефективного підходу до створення масштабованих і стійких додатків.

1.3.6. Недоліки мікро сервісної архітектури

Незважаючи на численні переваги, архітектура мікросервісів створює певні труднощі та недоліки, які організації повинні ретельно розглянути: Підвищена операційна складність: Управління розподіленою системою з кількома мікросервісами може бути складним з точки зору операцій. Для розгортання, моніторингу та підтримки численних незалежних служб можуть знадобитися складні інструменти та досвід.

Складність оркестровки та координації мікросервісів створює труднощі для забезпечення безперебійної роботи всієї програми. Комунікаційні витрати: Мікросервіси покладаються на зв’язок між службами, часто через мережу. Незважаючи на те, що це забезпечує незалежність і гнучкість, це також створює додаткові витрати на спілкування. Координація дій між мікросервісами може призвести до затримки, особливо в сценаріях, коли сервіси розподілені в різних географічних місцях. Узгодженість даних і транзакції: Підтримання узгодженості між розподіленими базами даних може бути складним завданням. Мікросервіси часто мають власні бази даних, і забезпечення узгодженості даних у транзакціях, які включають кілька сервісів, може бути складним.

Реалізація розподілених транзакцій пов’язана з власним набором проблем і може вплинути на продуктивність системи. Виявлення сервісу та виклики інфраструктури: Оскільки кількість мікросервісів зростає, керування виявленням сервісів стає вирішальним. Для зв’язку необхідно знати місцезнаходження та статус кожної служби. Впровадження надійного механізму виявлення служб та керування інфраструктурою, такою як балансування навантаження та мережеві конфігурації, ускладнює загальну архітектуру.

Питання безпеки: Безпека стає складнішою в середовищі мікросервісів. Координація автентифікації та авторизації між службами, захист каналів зв’язку та керування контролем доступу на детальному рівні вимагають ретельного розгляду.

Уразливість в одному мікросервісі потенційно може поставити під загрозу безпеку всієї системи. Початкові витрати на розробку: Перехід від монолітної архітектури до мікросервісів може потребувати значних зусиль на початковому етапі розробки. Розбиття монолітної програми на мікросервіси, встановлення протоколів зв’язку та впровадження належних механізмів тестування та моніторингу потребують ретельного планування та виконання. Послідовне ведення журналів і моніторинг:

Досягнення комплексного журналювання та моніторингу мікросервісів може бути складним завданням. Агрегування журналів і показників із розподілених служб, визначення джерела проблем і підтримка узгоджених практик моніторингу в різних технологіях можуть потребувати додаткових інструментів і зусиль. Розгляд вартості та ресурсів: Хоча мікросервіси можуть оптимізувати використання ресурсів, вони також можуть створити додаткові витрати. Управління декількома послугами може спричинити збільшення операційних витрат, а потреба в спеціалізованих інструментах та інфраструктурі може вплинути на загальні витрати. Підсумовуючи, хоча архітектура мікросервісів пропонує численні переваги, включаючи масштабованість і гнучкість, вирішення пов’язаних із цим проблем є вирішальним для успішного впровадження. Організації повинні зважити переваги та складності та ретельно спланувати свій підхід, щоб гарантувати, що переваги мікросервісів переважують потенційні недоліки в конкретному контексті.

1.3.7. Висновок що до архітектури

Вибір оптимальної архітектури для інтерактивної карти катання на роликових ковзанах передбачає детальний розгляд конкретних вимог, цілей і обмежень проекту. Як монолітна архітектура, так і архітектура мікросервісів мають певні переваги та виклики. У сценаріях, де простота, легкість розробки та ефективність використання ресурсів є найважливішими, монолітна архітектура може виявитися життєздатним вибором.

Згуртованість монолітних програм спрощує процеси розробки, налагодження та розгортання. Він пропонує уніфіковану кодову базу та може бути більш простим в управлінні для невеликих проектів або коли швидкі цикли розробки є пріоритетом. З іншого боку, для програм, які вимагають масштабованості, гнучкості та стійкості, особливо в контексті інтерактивної карти з динамічними функціями, архітектура мікросервісів може бути більш підходящим варіантом. Здатність самостійно масштабувати окремі компоненти, сприяти розподіленій розробці та адаптуватися до технологій, що розвиваються, добре відповідає вимогам багатофункціональної інтерактивної карти, що розвивається.

Враховуючи природу інтерактивної мапи, яка працює в режимі реального часу, сильні сторони архітектури мікросервісів щодо ізоляції помилок, масштабованості та безперервної інтеграції можуть переважити можливі ускладнення. Можливість незалежно оновлювати певні функції, адаптувати стеки технологій, що розвиваються, і забезпечувати відмовостійкість узгоджується з динамічною та розвиваючою природою такої карти. Зрештою, вибір між монолітною архітектурою та архітектурою мікросервісів має ґрунтуватися на глибокому розумінні масштабу проекту, цілей розвитку та очікуваного розвитку.

У той час як монолітні архітектури пропонують простоту та ефективність використання ресурсів, архітектури мікросервісів забезпечують гнучкість і масштабованість, необхідні для інтерактивних додатків, що розвиваються. Встановлення правильного балансу передбачає ретельний розгляд компромісів і узгодження вибраної архітектури з унікальними вимогами проекту карти для роликових ковзанів. Переваги та недоліки обох варіантів архітектури представлено у вигляді таблиці 1 для більш візуально наглядного порівняння.

Таблиця 1.1.

Порівняльна характеристика монолітної та мікро сервісної архітектури

Характеристика	Монолітна архітектура	Мікро сервісна архітектура
Швидкість розробки	Швидша у початку, сповільнюється при зростанні складності. Так як вся кодова база знаходится у одному місці, новим розробникам що доєднуються у команду може знадобитись велика кількість часу на вивчення усіх особливостей додатку.	Повільна у початку, швидша впродовж життєвого циклу. Для початку роботи системи необхідно зробити дуже багато налаштувань. Проте, надалі швидкість розробки може сильно зростати. Нові розробники можуть плавно входити у систему, опановуючи сервіс за сервісом, без необхідності повністю опанувати відразу.
Поріг входження	Середній. Навіть команда з відносто невеликим досвідом зможе провести проект від ідеї до працюючого додатку.	Високий. Потребує великої кількості розробників, що мають багато досвіду.
Горизонтальне масштабування	Так як дуже багато логіки та даних може зберігатись безпосередньо у памяті програми, досягнення горизонтального масштабування має свої виклики.	Гарні можливості розширення досягаються завдяки системам оркестрації та автобалансування. Додаткові копії севісів можуть бути запущені при досягненні певного порогу навантаження, чи знищені при низькому трафіку.
Вертикальне масштабування	Просте до реалізації завдяки розподіленим серверами.	Досить складне у реалізації з причини великої розподіленості.
Складність системи	Інфраструктура системи є помірно складною. Немає потреби у оркестрації, організації окремого рішення для моніторингу та безпеки.	Інфраструктура системи потребує наявності окремої команди для її організації та підтримки. Система взагалі має велику кількість рухливих частин, що, з одного боку, надає велику гнучкість. Потребує дуже впевнених знань.

Закінчення таблиці 1.1.

Стабільність	Стабільність залежить від якості та протестованості коду	На стабільність може впливати сумісність модулів
Вартість розробки	Розробка не потребує додаткових кадрів, що робить її дешевшою	Необхідність у команді для підтримки інфраструктури та великій кількості додаткових сервісів може значно збільшити вартість розробки та подальшої підтримки продукту
Можливості тестування	Є можливість протестувати повний пайплайн обробки даних на окремиій машині	Для тестування кожного сценарію скоріш за все знадобиться виклик ланцюжка сервісів, що значно збільшує час для тестування.
Гнучкість	Гнучкість системи базується на гнучкості самого рішення у вигляді коду	Додаткова гнучкість досягається за рахунок розподілення обовязків. Кожен елемент з ланцюжку можна реалізувати інакше, допоки інтерфес сервісу задовольняє встановлений раніше контракт.
Моніторинг	Моніторинг організовуються досить просто, так як для всього додатку можна використати одну систему логування.	Якісний моніторинг потребує додаткових витрат та рішень. Щоб організувати моніторинг упродовж всього ланцюжка, необхідно використовувати складні системи, як от OpenTelemetry.
Безпека	Так як сама система не має великої кількості внутрішньої комунікації та має одну точку входу, організувати безпеку легше.	Так як сервіси мають постійно спілкуватись між собою та з зовнішнім світом, треба дуже комплексно відноситись до організації безпеки. Система має багато точок входу.

У розробці даного проекту, по при вимушені проблеми із часом та відстуність команди, вирішено обрати монолітну архітектуру проектування. Хоча проект і має потенціал та можливоссті до розширення, скоріше всього, критична маса завеликої складності не буде досягнута на даному етапі, зважаючи на природу предметної області. З міркувань потреб часу та сил на розробку та враховуючи досвід великих проектів, що спершу розробляли монолітне рішення та переробляли його на мікросервісне після досягнення певного розміру компанії, можна піти таким самим шляхом. Тож, монолітна архітектура, після зауваження усіх недоліків та переваг, є найкращим вибором у даній ситуації [1].

1.4. Огляд технологій та засобів, що використовуватимуться для створення програмного засобу

Вибір актуальних та підходящих технологій та інструментів для розробки і подальшої підтримки програмного засобу несе ключову роль у розробці ПЗ. Так як зміна обраної технології або підходу може бути дуже складним та коштовним процессом, обгрунтований вибір інструментів є важливим завданням на початку розробки.

По-перше, обрані технології суттєво впливають на ефективність проекту. Вибір відповідних інструментів може оптимізувати робочі процеси розробки, покращити співпрацю між членами команди та оптимізувати використання ресурсів. І навпаки, невідповідний стек технологій може створити непотрібні складності, перешкоджаючи швидкості та ефективності розробки.

По-друге, масштабованість і перспективність проекту тісно пов’язані з початковими технологічними рішеннями. Добре зважений вибір забезпечує плавну інтеграцію нових функцій, адаптацію до змінних вимог і інкорпорацію нових галузевих стандартів. Навпаки, поспішне або необізнане рішення може призвести до проблем масштабованості, вимагаючи значного

рефакторингу або навіть необхідності повного перегляду технологічного стеку.

Від грамотного вибору технологій залежить надійність і стабільність кінцевого продукту. Надійні та добре підтримувані інструменти сприяють створенню стабільної програми, зводячи до мінімуму ймовірність багів, помилок і системних збоїв. І навпаки, невідповідний або застарілий стек технологій може створити вразливі місця та поставити під загрозу загальну надійність проекту. Рішення щодо технологій також має серйозні наслідки для підтримки та довгострокової життєздатності проекту.

Ретельно підібраний стек сприяє простоті обслуговування, спрощує оновлення, виправляє помилки та включає нові функції. Навпаки, довільний вибір може призвести до проблем в управлінні та розвитку проекту з часом. Крім того, не слід недооцінювати вартість і ресурси технологічного рішення. Вибрані технології впливають не тільки на витрати на розробку, але й на витрати на поточне обслуговування. Невідповідні технології можуть призвести до вищих операційних витрат, збільшення технічної заборгованості та більшої ймовірності ресурсомісткого усунення несправностей. Підсумовуючи, прийняття свідомого рішення щодо технологій та інструментів перед початком проекту є стратегічним імперативом. Наслідки цього рішення впливають на весь життєвий цикл проекту, впливаючи на ефективність, масштабованість, надійність, ремонтопридатність і загальний успіх проекту. Витрачення часу на ретельну оцінку та вибір найбільш підходящих технологій є інвестицією в довговічність проекту, адаптивність і остаточне досягнення його цілей.

Для розробки додатку обрано мову програмування Java 21, з декількох важливих причин. Вибір Java 21 як мови програмування для проекту є розумним вибором, особливо з огляду на кваліфікацію розробників, широку підтримку фреймворків і статус Java як широко поширеного стандарту в корпоративних середовищах.

Перш за все, рівень знань розробників у Java відіграє ключову роль у процесі прийняття рішення. Якщо команда розробників добре володіє Java, використання наявних знань у цій мові може значно підвищити ефективність розробки.

Так як сам розробник цього диплому найкраще володіє данною мовою програмування, це стало значним фактором щодо вибору мови.

Багата екосистема фреймворків Java є ще однією вагомою причиною для вибору її як мови для проекту. За допомогою таких фреймворків, як Spring, Hibernate та інших, Java забезпечує повну підтримку різних аспектів розробки додатків, включаючи веб-розробку, доступ до даних і функціональні можливості корпоративного рівня. Ці фреймворки спрощують процеси розробки, просувають найкращі практики та пропонують велику кількість готових модулів, прискорюючи часові рамки проекту та забезпечуючи дотримання галузевих стандартів. Крім того, визнана позиція Java як стандарту в корпоративних середовищах додає довіри та надійності до вибору мови. Багато великих підприємств покладаються на Java для створення критично важливих програм завдяки її надійності, функціям безпеки та масштабованості.

Вибір Java відповідає галузевим нормам і сприяє бездоганній інтеграції з існуючими корпоративними системами та технологіями. Прихильність Java до зворотної сумісності заслуговує на увагу. Мова надає перевагу збереженню сумісності з попередніми версіями, гарантуючи, що програми, розроблені за допомогою старих версій Java, можуть працювати на новіших версіях. Ця прихильність до зворотної сумісності збільшує довговічність програм, забезпечуючи стабільну основу для довгострокових проектів і мінімізуючи потенційні збої, спричинені частими оновленнями мови. Крім того, переносимість Java на різні платформи є значною перевагою.

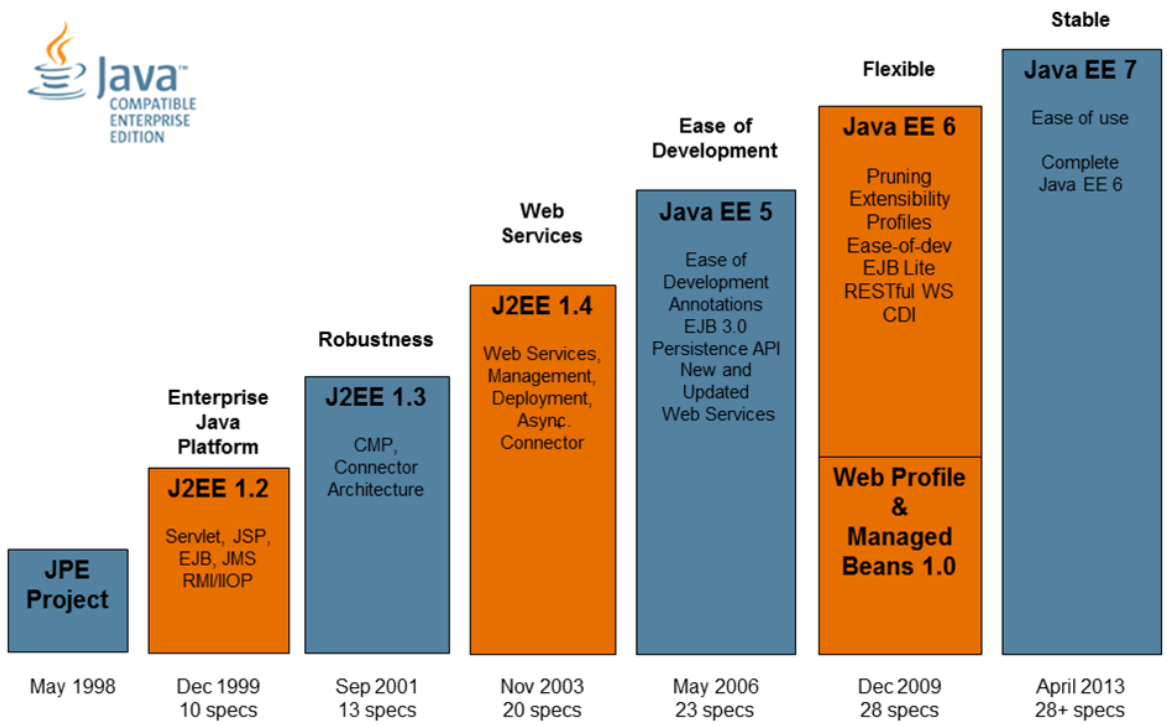

Рис. 1.5 Візуалізація основних стадій розвитку Java EE

Програми, написані на Java, можуть працювати на широкому діапазоні пристроїв і операційних систем, пропонуючи гнучкість у розгортанні та розширюючи охоплення проекту в різних середовищах без істотних змін. Підсумовуючи, вибір Java 21 для проекту є розумним рішенням, яке ґрунтується на кваліфікації розробників, надійній екосистемі фреймворку та усталеному статусі Java як корпоративного стандарту. Використання наявного досвіду команди розробників, використання потужності фреймворків Java і узгодження з галузевими практиками створюють міцну основу для успішної розробки та розгортання проекту.

Наступний крок – вибір базового фреймворка для розробки. Spring Boot виділяється як переконливий вибір для розробки інтерактивної програми для катання на роликових ковзанах завдяки кільком ключовим функціям і перевагам, які відповідають конкретним вимогам цього типу програми: Швидка розробка та спрощена конфігурація: Spring Boot наголошує на традиційності, а не на конфігурації, що дозволяє розробникам швидко розпочати роботу з мінімальним налаштуванням. Його спрощена

конфігурація та параметри за замовчуванням дозволяють розробникам зосередитися на створенні функцій програми, а не витрачати багато часу на шаблонний код.

1) Підтримка архітектури мікросервісів: Якщо прийнято рішення прийняти архітектуру мікросервісів для додатка карти катання на роликах, Spring Boot добре підходить для цього підходу. Його модульна конструкція та підтримка створення незалежних служб, які можна розгортати, узгоджуються з парадигмою мікросервісів, полегшуючи створення масштабованих і слабко пов’язаних компонентів. Вбудований веб-сервер: Spring Boot містить вбудований веб-сервер (зазвичай Apache Tomcat, Jetty або Undertow), що спрощує розгортання веб-додатків. Цей вбудований сервер добре підходить для надання користувачам інтерактивної карти, усуваючи потребу у конфігурації зовнішнього сервера та зменшуючи складність розгортання.

2) Інтеграція з Spring Ecosystem: Spring Boot легко інтегрується з ширшою екосистемою Spring, надаючи доступ до великої кількості бібліотек і модулів. Ця екосистема включає Spring Security для обробки автентифікації та авторизації, Spring Data для спрощеного доступу до даних і Spring Cloud для реалізації хмарних шаблонів, якщо програма розвивається до розподіленого середовища.

3) Початкова конфігурація: Spring Boot пропонує колекцію «стартерів», які спрощують включення загальних залежностей і конфігурацій для конкретних випадків використання. Для картографічної програми відповідні стартери, такі як Spring Boot Starter Web для веб-розробки та Spring Boot Starter Data JPA для доступу до даних, можуть прискорити процес розробки.

4) Моніторинг та управління: Spring Boot Actuator надає вбудовані функції для моніторингу та керування програмами. Це включає в себе кінцеві точки для перевірки працездатності, показників і інформації про програми. Ці функції є цінними для забезпечення працездатності та продуктивності додатка карти катання на роликах, особливо у виробничому середовищі.

5) Підтримка спільноти та документація: Spring Boot отримує переваги від надійної та активної спільноти, що означає обширну документацію, навчальні посібники та підтримку, керовану спільнотою. Ця велика кількість ресурсів може бути безцінною для розробників, які працюють над додатком карти для катання на роликових ковзанах, надаючи рішення для поширених проблем і просуваючи найкращі практики. На основі Java та агностик мови: Будучи базою на Java, Spring Boot використовує широке поширення мови програмування Java та розгалужену екосистему бібліотек. Крім того, Spring Boot може включати компоненти, розроблені іншими мовами, що робить його універсальним для програм із різноманітними вимогами.

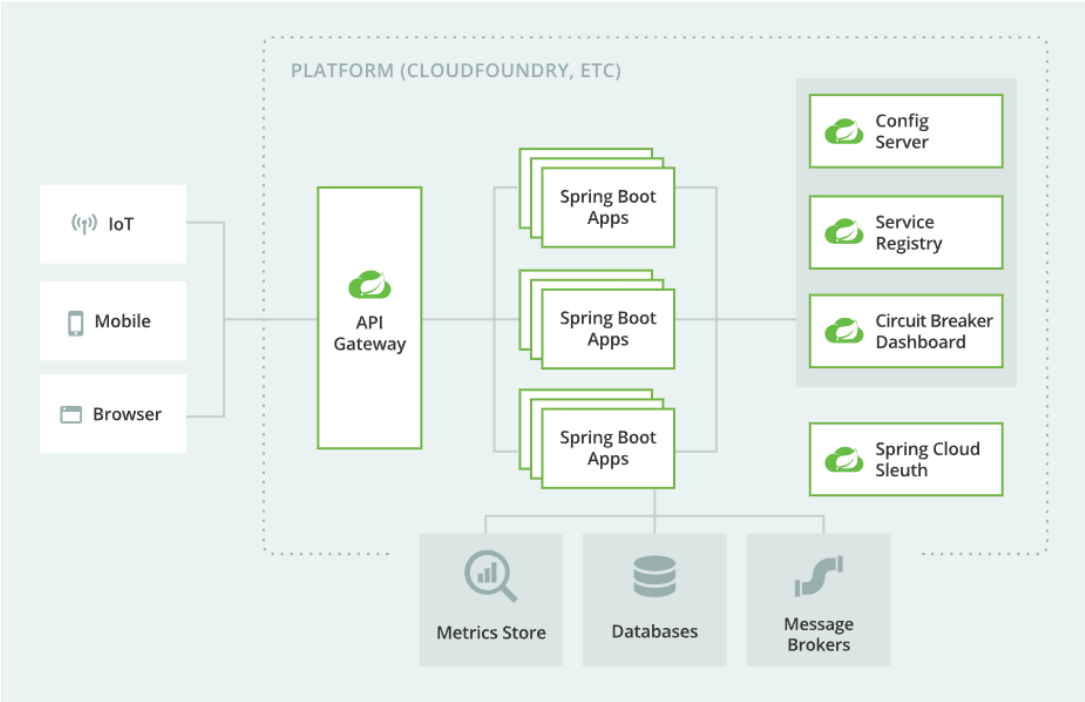

Рис. 1.6 Діаграма типової архітектури додатку з використанням Spring Boot

1.5. Аналіз сучасних технологій верифікації та тестування програмних систем

Тестування ПЗ – спосіб перевірки працездатності та якості програмного забезпечення, або процес перевірки відповідності висунутих вимог до програмного засобу, та його наявних функціональних можливостей.

Наявність надійного багаторівневого тестового покриття має вирішальне значення для забезпечення надійності, стабільності та загальної якості програмного забезпечення. Цей комплексний підхід до тестування охоплює різні рівні стека програмного забезпечення, включаючи модульні, інтеграційні та наскрізні тести. Ось ключові причини, чому підтримка якісного багаторівневого тестового покриття є надзвичайно важливою:

1) Раннє виявлення помилок: Багаторівневе тестування допомагає виявляти помилки та проблеми на різних етапах процесу розробки. Модильні тести націлені на конкретні функції або методи, виявляючи проблеми на рівні коду. Інтеграційні тести гарантують безперебійну роботу різних компонентів, а наскрізні тести імітують взаємодію користувача, виявляючи потенційні проблеми у всій системі. Виявлення та виправлення помилок на ранніх етапах життєвого циклу розробки мінімізує витрати та зусилля, необхідні для усунення помилок на наступному етапі.

2) Забезпечення якості коду: Модульні тести сприяють підтримці високої якості коду, перевіряючи правильність окремих одиниць коду. Це гарантує, що кожен компонент працює належним чином і відповідає встановленим вимогам. Комплексне модульне тестування також заохочує розробників писати модульний, добре структурований і підтримуваний код.

3) Підвищення зручності коду: Добре продуманий набір тестів служить живою документацією для кодової бази. Це допомагає розробникам зрозуміти очікувану поведінку різних компонентів, полегшуючи підтримку та зміну коду з часом. Це особливо важливо у великих, складних проектах, де розробники можуть змінювати або підтримувати код, який вони спочатку не писали.

4) Сприяння безперервній інтеграції та розгортанню (CI/CD): Багаторівневе тестове покриття є важливим для успішних конвеєрів CI/CD. Автоматизовані тести на різних рівнях гарантують, що зміни, внесені до кодової бази, не порушують існуючу функціональність. Це дозволяє швидше

та впевненіше розгортати нові функції чи оновлення, оскільки процес тестування забезпечує захист від регресії.

5) Покращення стійкості системи: Тести інтеграції перевіряють взаємодію між різними модулями або службами в програмі. Завдяки ретельному тестуванню цих з'єднань розробники можуть виявити й вирішити потенційні проблеми з інтеграцією, забезпечивши загальну стійкість і стабільність системи.

6) Покращення взаємодії з користувачем: Наскрізні тести імітують сценарії реального користувача, надаючи цілісне уявлення про функціональність програми з точки зору користувача. Комплексне наскрізне тестування допомагає переконатися, що всі взаємодії з користувачем, робочі процеси та користувацькі інтерфейси працюють безперебійно, сприяючи позитивній та надійній взаємодії з користувачем.

7) Підвищення впевненості в змінах: Комплексний набір тестів, що включає різні рівні тестування, вселяє впевненість у розробників і зацікавлених сторін під час внесення змін до кодової бази. Знання про наявність автоматичних тестів для перевірки поведінки програми дає змогу більш рішуче приймати рішення під час циклів розробки та випуску.

8) Відповідність бізнес-вимогам: Багаторівневе тестування гарантує відповідність програмного забезпечення вимогам бізнесу на різних рівнях. Модильні тести підтверджують, що окремі функції відповідають специфікаціям, тоді як інтеграційні та наскрізні тести перевіряють, що програма в цілому відповідає бізнес-цілям вищого рівня.

Взагалі, тести – основний інструмнет який дозволяє впевнетись у тому що бізнес вимоги покриті правильно та функціонування додатку задовольняє їх. З досвіду розробки стало зрозуміло, що відсутність гарного покриття тестами може значно погіршити якість коду та значно затримати розробку продукту.

Також, створення та документування тест-кейсів дозволяє впевнитись у тому, що після останнього оновлення кодової бази усі попередні логічні

елементи продовжують працювати так само якісно. З досвіду стало зрозуміло, що для вдалого ведення тестової документації дуже важливо мати зручні інструменти. Прикладом є інструмент Test Rail.

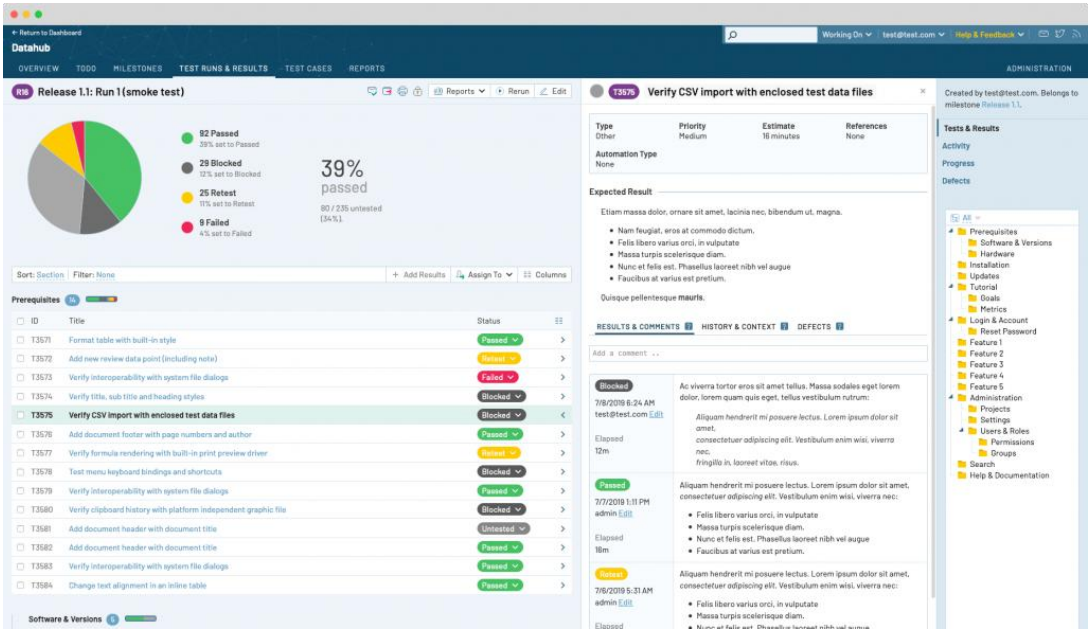

Рис. 1.7 Інтерфейс програми TestRail

TestRail виділяється як чудовий вибір для керування тестами та тестовими прогонами завдяки кільком ключовим функціям і перевагам, які спрощують процес тестування.

По-перше, TestRail забезпечує зручний та інтуїтивно зрозумілий інтерфейс, що полегшує тестерам і командам ефективне керування своїми тестами. Добре продуманий макет платформи гарантує, що навігація, створення та оновлення тестових випадків є простим процесом, підвищуючи загальну зручність використання. Надійні можливості TestRail з керування тестовими випадками полегшують організацію та структурування тестових випадків, дозволяючи командам створювати комплексний набір тестів. Цей структурований підхід підтримує краще планування, виконання та документування тестів, що сприяє підвищенню ефективності тестування.

Крім того, TestRail пропонує потужні функції співпраці, що дозволяє командам безперебійно працювати в різних проектах і циклах тестування. Тестувальники можуть співпрацювати над розробкою тестових випадків,

ділитися думками та надавати відгуки, сприяючи створенню середовища для спільного тестування. Інтеграційні можливості TestRail сприяють його універсальності. Він легко інтегрується з різними інструментами тестування, системами відстеження проблем та іншими системами управління проектами. Ця інтеграція забезпечує плавний робочий процес, дозволяючи командам підключати TestRail до наявних інструментів і використовувати його можливості без збоїв. Функції звітності та аналітики TestRail надають цінну інформацію про хід і результати тестування.

Команди можуть створювати детальні звіти про виконання тестів, статус і охоплення, надаючи зацікавленим сторонам інформацію, необхідну для прийняття обґрунтованих рішень щодо якості проекту. Крім того, TestRail підтримує ефективне планування та виконання тестів шляхом створення тестових прогонів. Тестування дозволяє командам організовувати тестові випадки на основі конкретних критеріїв, таких як типи тестів, функції чи випуски. Ця гнучкість підвищує адаптивність стратегій тестування, спрощуючи керування та виконання циклів тестування.

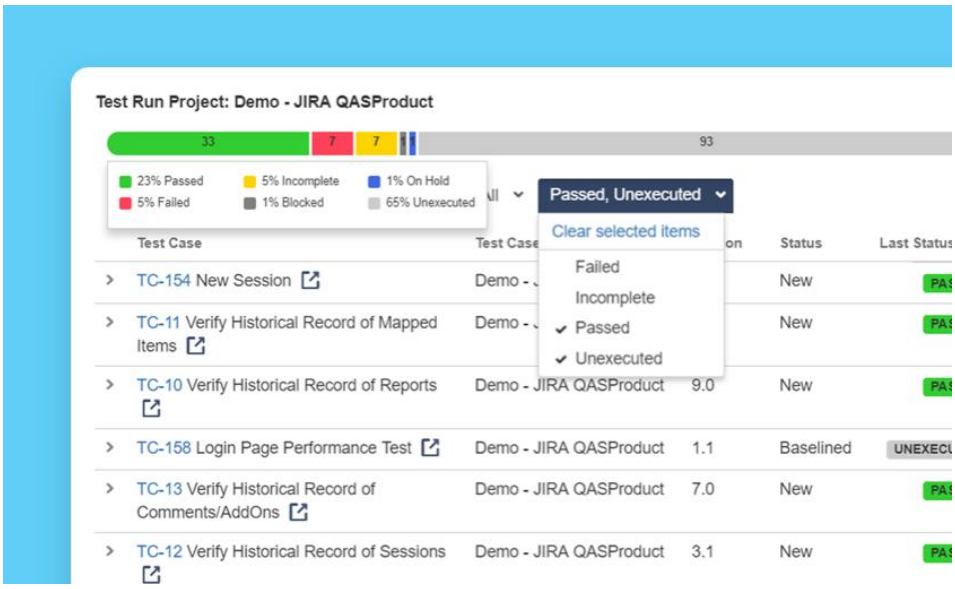

Рис. 1.8 Інтеграція з Jira

Функції відстеження дозволяють командам пов’язувати тестові випадки з конкретними вимогами чи історіями користувачів, забезпечуючи

всебічне охоплення тестами відповідно до цілей проекту. Ця відстежуваність допомагає підтвердити, що програмне забезпечення відповідає запланованим специфікаціям і вимогам. Хмарна доступність TestRail сприяє віддаленій співпраці та гнучкості в управлінні тестами з будь-якого місця. Це особливо важливо для розподілених команд або тих, хто працює в різних місцях.

1.6. Значення Scrum підходу для керування програмними проектами у IT-підприємствах сьогодення

Scrum — це структура, в рамках якої люди можуть вирішувати складні адаптаційні проблеми, продуктивно та творчо надаючи продукти найвищої можливої цінності [13].

Це легка структура, яка допомагає людям, командам і організаціям створювати цінність за допомогою адаптивних рішень для складних проблем. Спільні творці Scrum Кен Швабер і Джефф Сазерленд написали The Scrum Guide, щоб пояснити Scrum чітко і коротко. Цей посібник містить визначення Scrum. Це визначення складається з відповідальності Scrum, подій, артефактів і правил, які їх пов’язують (рис. 1.4).

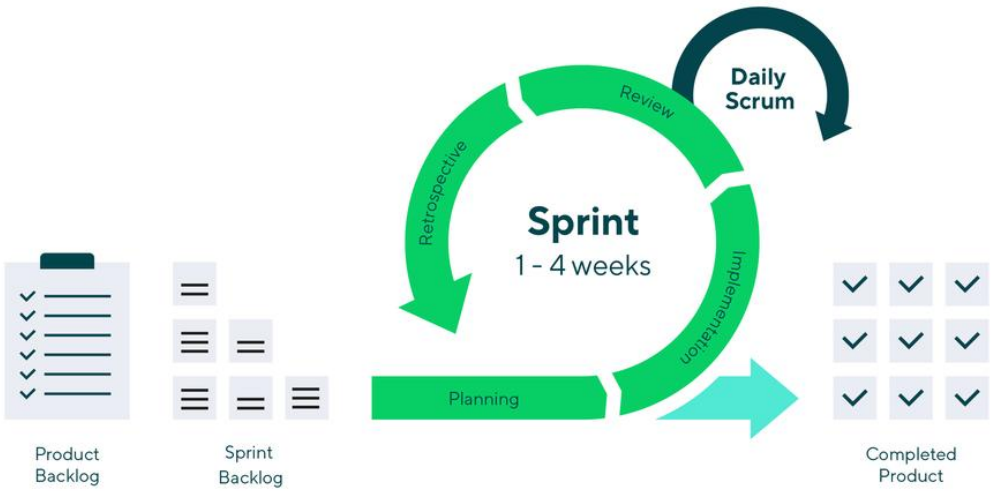

Рис. 1.9 Діаграма Scrum-циклу

Scrum постає як високоефективний підхід до сучасної розробки, пропонуючи динамічну та адаптовану структуру, яка добре відповідає вимогам сучасної практики розробки програмного забезпечення. Однією з помітних сильних сторін Scrum є акцент на ітераційній та поетапній розробці. Цей підхід дозволяє командам забезпечувати відчутну цінність за короткі регулярні проміжки часу, сприяючи безперервному зворотному зв'язку із зацікавленими сторонами. Ітераційний характер Scrum враховує зміни та вдосконалення, дозволяючи командам швидко реагувати на зміну вимог і динаміку ринку. Структура Scrum сприяє співпраці та прозорості між членами команди. Щоденні зустрічі забезпечують платформу для відкритого спілкування, сприяючи обміну інформацією про прогрес, виклики та залежності. Це середовище для співпраці покращує згуртованість команди та гарантує, що всі працюють на одній сторінці, сприяючи спільному розумінню цілей проекту.

Рольова структура в Scrum із визначеними ролями, такими як Власник продукту, Scrum Master і Команда розробників, покращує підзвітність і розподіл відповідальності (рис.1.10.).

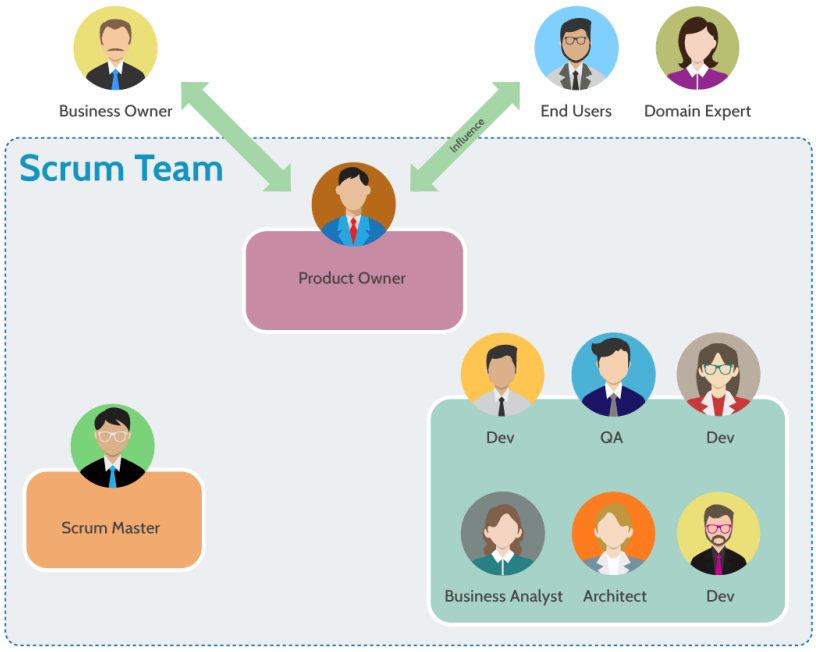

Рис. 1.10 Ролі у команді за методологією

Кожна роль сприяє загальному успіху проекту, забезпечуючи чітке визначення завдань, моніторинг прогресу та швидке усунення перешкод. Такий чіткий розподіл обов'язків підвищує ефективність і прискорює процеси прийняття рішень.

Орієнтація Scrum на надання мінімально життєздатного продукту (MVP) на ранніх стадіях циклу розробки узгоджується з принципами швидкого та економічного розвитку. Цей підхід дозволяє командам перевіряти припущення, збирати відгуки користувачів і вносити обґрунтовані коригування, гарантуючи, що кінцевий продукт ефективно відповідає потребам користувачів.

Структура Scrum є адаптивною та приймає зміни. Регулярні цикли спринту, як правило, два-чотири тижні, дають можливість для постійних роздумів і вдосконалення. Команди можуть адаптувати свої стратегії, вдосконалювати процеси та оптимізувати робочі процеси на основі інформації, отриманої під час кожного спринту.

Ця здатність до адаптації особливо цінна в динамічних і швидкоплинних середовищах розробки. Акцент Scrum на емпіричному контролі процесів із регулярними циклами перевірки та адаптації сприяє підходу до прийняття рішень, керованого даними. Використання метрик і механізмів зворотного зв’язку дає змогу командам об’єктивно оцінювати свою ефективність, визначати сфери, які потрібно вдосконалити, і вносити корективи на основі даних для підвищення загальної ефективності.

Церемонії Scrum, такі як планування спринту, огляд спринту та ретроспектива спринту, створюють структуровані можливості для роздумів, зворотного зв’язку та постійного вдосконалення. Ці церемонії сприяють розвитку культури навчання та інновацій у команді, заохочуючи виявлення та впровадження кращих практик.

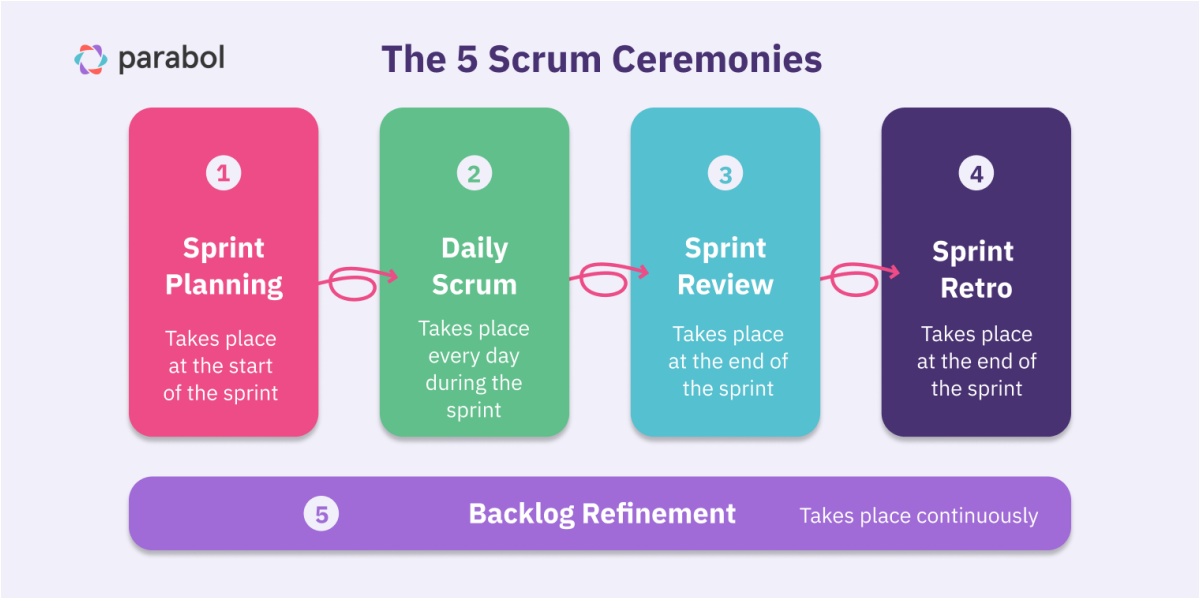

Рис. 1.11 Основні Scrum церемонії

Підсумовуючи, Scrum виділяється як підхід, який добре підходить для сучасного розвитку, наголошуючи на ітераційному прогресі, заохочуючи співпрацю, забезпечуючи підзвітність, сприяючи адаптації, підтримуючи практики економії та створюючи культуру постійного вдосконалення. Його гнучка та прозора структура позиціонує Scrum як цінну методологію для навігації в складних умовах сучасного ландшафту розробки програмного забезпечення.Scrum простий. Це протилежність великої колекції переплетених обов'язкових компонентів. Scrum - це не методологія. Scrum реалізує науковий метод емпіризму. Scrum замінює запрограмований алгоритмічний підхід евристичним, з повагою до людей і самоорганізацією для боротьби з непередбачуваністю та вирішенням складних проблем. На малюнку нижче зображено Scrum в дії, як описано Кеном Швабером і Джеффом Сазерлендом у їхній книзі «Програмне забезпечення за 30 днів», що веде нас від планування до доставки програмного забезпечення.

1.7. Визначення необхідності розробки програмного продукту, потенційні місця для його впровадження та особливості реалізації

Впровадження ПЗ – це фінальний етап в процесі розробки програмного забезпечення. Його метою є виконання усіх необхідних дій що необхідні для підготування веб-системи до використання. Після цього фінальний користувач матиме до неї доступ.

Кожний додаток використовує власний стек технологій та інструментів, тож, увесь процес може досить сильно відрізнятися. Безпосередньо процес впровадження може відрізнятися за кількістю кроків, необхідних для кожної окремої системи. Час, який виділяється саме на впровадження засобу, також може значно відрізнятися в залежності від складності структури продукту. Єдине, що є суцільно спільним для усіх програмних засобів – це саме те, що за для переходу на етап впровадження додаток має бути повністю готовим і виконувати поставлені перед ним задачі.

Для впровадження в експлуатацію додатку інтерактивної мапи необхідно в першу чергу перевірити в працездатності програмного продукту. Наступним кроком є розміщення додатку на веб-сервері, з метою безперервної роботи і можливістю одночасного доступу багатьох користувачів. Далі, необхідно налаштувати події, що будуть запускати серверну частину додатку, що буде відповідати безпосередньо за актуальність даних про маршрути. Після цього необхідно повторно протестувати роботу програмного засобу, та впевнитися у відсутності значних змін в роботі. І тільки після успішного регресивного та системного тестування користувачам надається доступ до роботи з програмною системою.

На перших етапах роботи системи потрібно забезпечити якісний супровід програмного засобу з метою негайного попередження виникнення неочікуваних проблем. Сам додаток має бути практичним у використанні як серед ролерів, так і серед представників інших суміжних видів спорту. Через

те, що продукт буде реалізовано у вигляді веб-системи, кожному зацікавленому користувачу надаватиметься можливість у пару рухів і без зайвих зусиль швидко отримати актуальну потрібну інформацію про маршрути.

1.8. Висновки до розділу

Під час написання розділу було проаналізовано дану предметну область поставленого завдання та визначено його актуальність як достатньо високу, дивлячись на швидкість цифровізації суспільства. Дослідивши актуальні та ефективні практики інженерії вимог до розробки веб-систем і практичні особливості саме архітектурного проектування веб-додатків зроблено висновоки, що кращим у даній ситуації архітектурним підходом до проектування системи буде саме монолітна архітектура, з причини наявності зрозумілої цілі що до функціоналу додатку та сильно обмежених часових та людських ресурсів на розробку проекту.

Було проаналізовано сучасні технології верифікації та системного тестування програмних систем. Тестування – один з найважливіших компонентів роботи над будь-якою системою, бо воно дає змогу впевнитися у правильності виконання вимог, поставлених перед розробкою додатка, а досить часто й дають змогу переосмислити вже встановлені вимоги. Проведено визначення методології Scrum для керування проектами у сучасному IT-підприємстві.

Scrum як методологія займає зараз домінуючу позицію серед ринку розробки програмних систем на сьогоднішній день. Така модель часто підходить до різноманітних сценаріїв розробки, де команди розробників будь-якого рівня стикаються кожен робочий день. Хоча при розробці в даній ситуації не можливо назвати свій підхід канонічним Scrum, бо він потребує наявності повної команди, де кожен член має свою зону відповідальності. Коли одна людина працює відразу над усім проектом, починаючи з коду й закінчуючи розробкою бізнес-плану та написанням документації, ясна річ,

якість усіх пунктів буде дещо нижча, а тем роботи значно повільнішим. Проте, навколишні реалії нажаль диктують саме цей сценарій, без видимої можливості на апеляцію. Тож, робота велась відповідно до обставин.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ВЕБ-СИСТЕМИ ІНТЕРАКТИВНОЇ МАПИ ДЛЯ КАТАННЯ НА РОЛИКАХ

2.1. Визначення вимог до програмного продукту

Перед розробкою програмного продукту спершу необхідно виявити перелік та визначити вимоги до додатку. Встановлення чітких і вичерпних вимог до проекту програмного забезпечення має першочергове значення, оскільки воно служить основоположним планом, який спрямовує весь процес розробки. Чітко визначені вимоги забезпечують спільне розуміння між зацікавленими сторонами, включаючи розробників, дизайнерів і клієнтів, щодо цілей, функцій і обмежень програмного забезпечення.

Вони служать дорожньою картою, допомагаючи пом’якшити непорозуміння, узгодити очікування та забезпечити відповідність кінцевого продукту наміченим цілям. Чіткі вимоги сприяють ефективному плануванню проекту, розподілу ресурсів і управлінню ризиками, дозволяючи командам надавати рішення, яке відповідає потребам користувачів, відповідає галузевим стандартам і постачається в установлені терміни та бюджет.

По суті, встановлення вимог є вирішальним кроком у сприянні успіху проекту, мінімізації невизначеності та закладанні основи для впорядкованого та цілеспрямованого процесу розробки.

2.1.1. Огляд аналогічних систем

Інтерактивні карти, призначені для катання на роликових ковзанах, стали свідками значного прогресу за останні роки завдяки технологічним інноваціям і зростаючому інтересу до розваг на природі. Ці карти створені для покращення досвіду катання на роликах, надаючи основні функції, допомагаючи у виборі маршруту та підвищуючи загальну безпеку. Тут буде аналіз деяких з відомих інтерактивних карт, доступних сьогодні, і їхніх ключових характеристик.

1. Strava

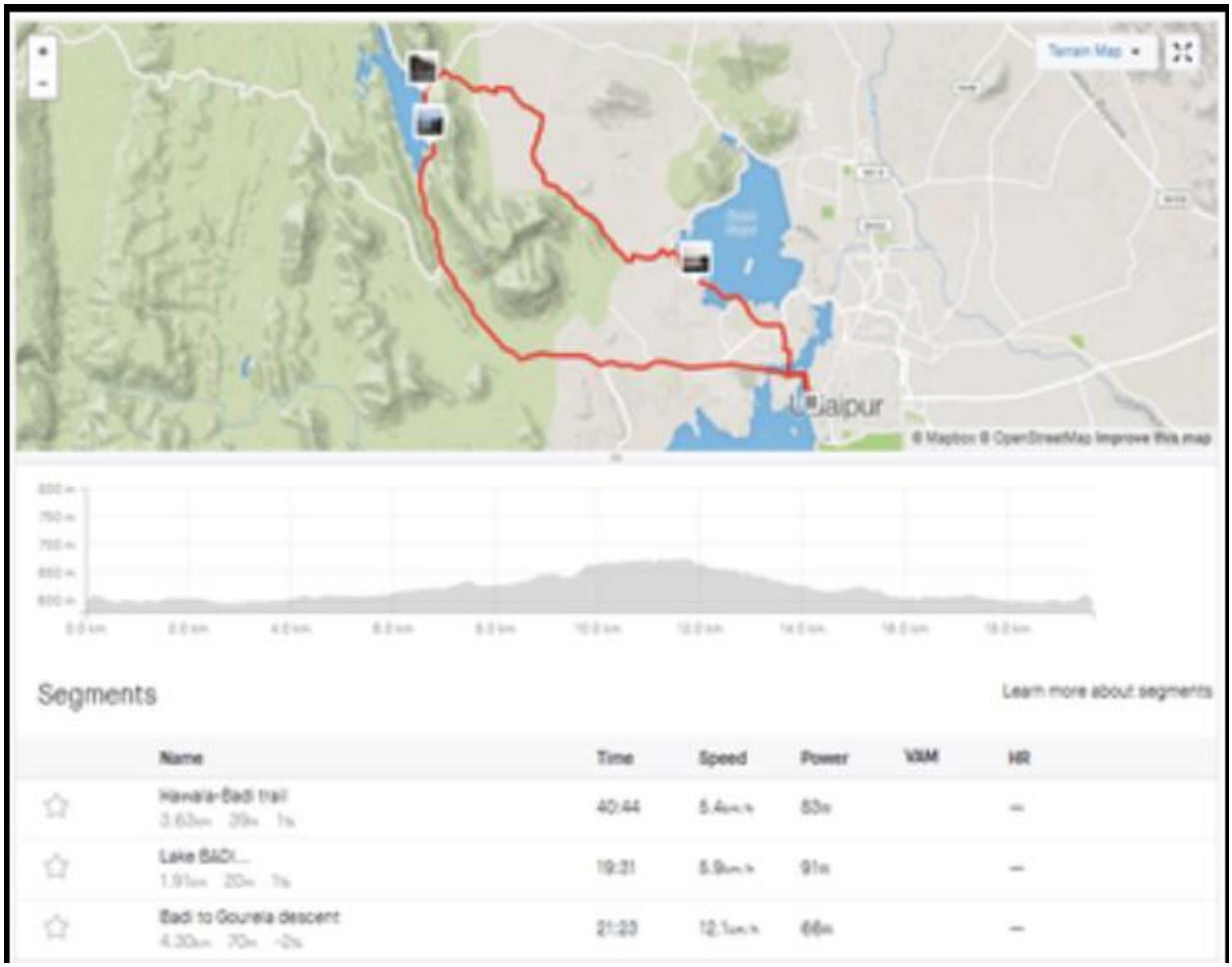

Рис. 2.1 Інтерфейс додатку Strava

Основні функції: відстеження маршруту, розрахунок відстані, профіль висоти, спільний доступ до спільноти, виклики сегментів. Strava є широко визнаною платформою у фітнес-спільноті. Він пропонує любителям роликових ковзанів можливість записувати та аналізувати свої маршрути, відстежувати показники продуктивності та ділитися своїм досвідом із зацікавленою спільнотою. Ролери можуть створювати та досліджувати популярні маршрути, ставити цілі та брати участь у дружніх змаганнях через сегментовані завдання.

2. MapMyRide

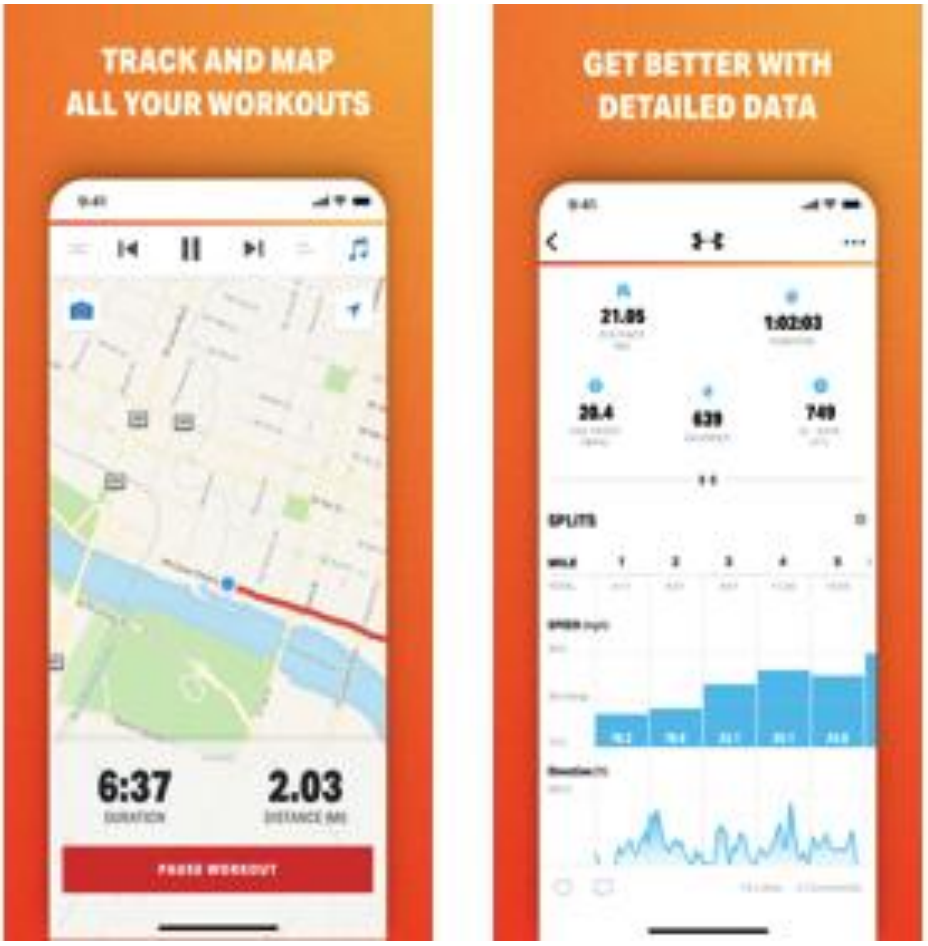

Рис. 2.2 Інтерфейс додатку MapMyRide

Основні характеристики: планування маршруту, GPS-відстеження, журнал тренувань, залучення спільноти, відстеження харчування. MapMyRide, частина пакету Under Armour Connected Fitness, надає інтуїтивно зрозумілий інтерфейс для планування та відстеження маршрутів катання на роликах. Користувачі можуть планувати свої поїздки, ставити цілі та відстежувати прогрес. Додаток також забезпечує соціальну взаємодію в межах спільноти, де любителі роликових ковзанів можуть ділитися маршрутами та досягненнями.

Додаток поставляється у якості мобільного застосування із досить зручним інтерфейсом, проте не має версії для компютера, що може ускладнити такі сценарії як побудова довгого маршруту та аналіз нюансів. Це

можна виконати і на мобільному пристрої, але версія для ПК додає набагато більше можливостей.

3. KOMOOT

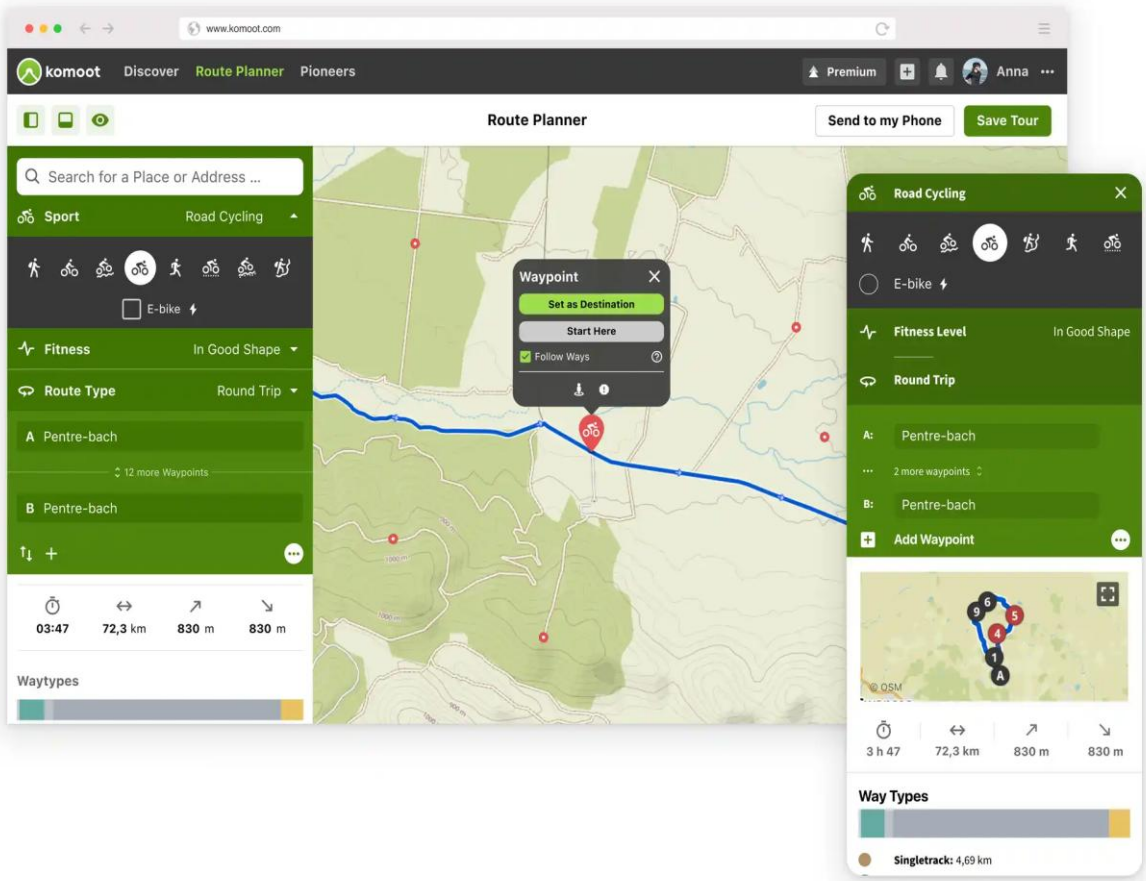

Рис. 2.3 Інтерфейс додатку KOMOOT

Основні характеристики: планування маршруту та навігація, планування пригод, голосова навігація від повороту до повороту, офлайн-карти. Komoot відомий своїм наголосом на плануванні пригод, що робить його чудовим вибором для любителів катання на роликах, які прагнуть досліджувати. Він пропонує детальне планування маршруту, допомогу в навігації та навіть пропонує персоналізовані маршрути на основі вподобань. Опція офлайн-карт забезпечує доступ до маршрутів навіть у віддалених районах.

Цей додаток має як версію для ПК, так і мобільний додаток, що робить його дуже універсальним. Таким чином можна побудувати маршрут чи оцінити існуючий працюючи з достатньо великим єкраном на ПК, щоб потім відстежувати своє переміщення та прогрес за допомогою додатку.

Мінусом відповідно до вимог цього проекту є відсутність соціальної складової: можна легко будувати маршрути та використовувати їх, проте немає можливості перевірити рейтинг та дізнатись думку спільноти.

4. All trails

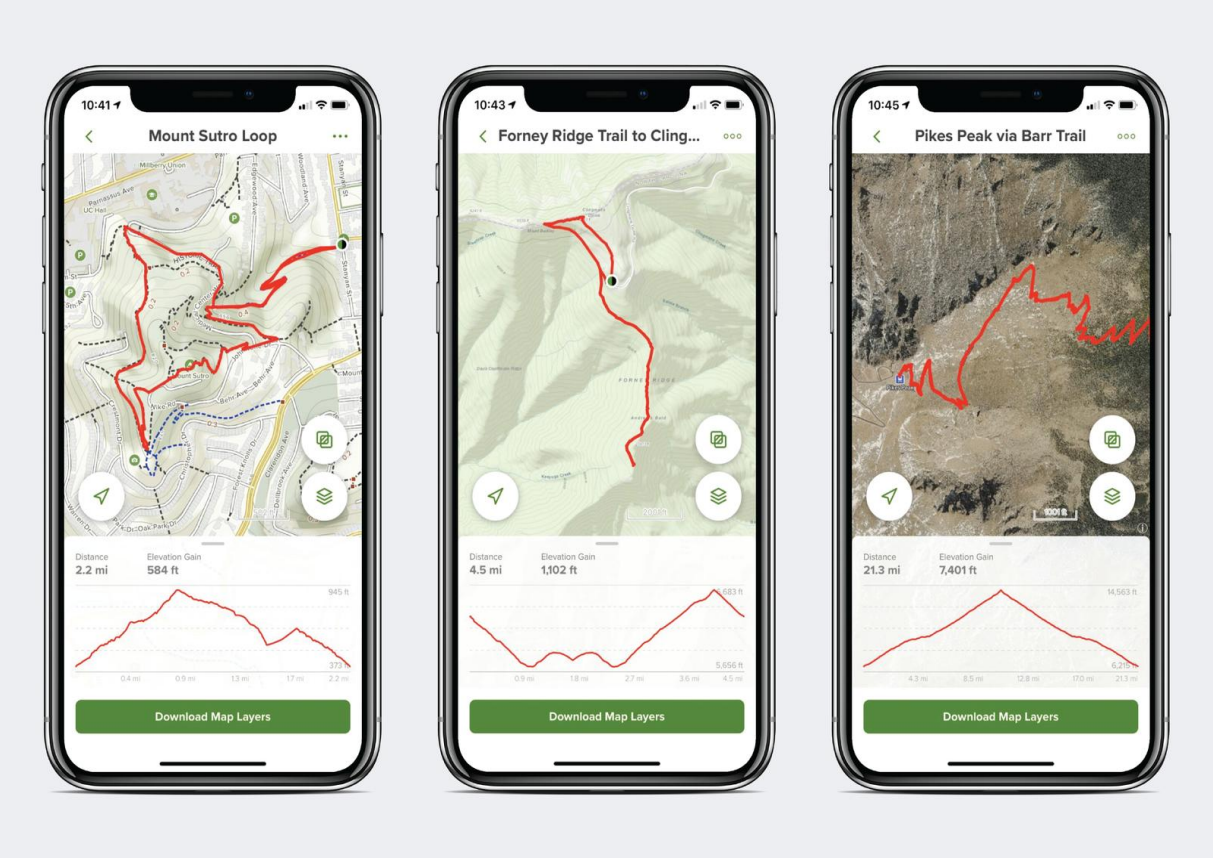

Рис. 2.4 Інтерфейс додатку All trails

Основні функції: пошук маршрутів, відстеження GPS, офлайн-карти, огляди, фотографії та журнали подорожей. Спочатку AllTrails був орієнтований на туристів, але здобув популярність серед любителів роликових ковзанів, відкриваючи та досліджуючи маршрути, зручні для катання на роликах. Платформа пропонує детальну інформацію про

маршрути, відгуки користувачів і фотографії, що допомагає у виборі маршруту та дає зрозуміти, чого очікувати.

Таким чином, поточні інтерактивні карти для катання на роликових ковзанах пропонують низку функцій для покращення досвіду катання на роликах. Ці програми задовольняють різноманітні потреби, від відстеження маршрутів і ефективності до виховання почуття спільності. Однак подальший розвиток, зосереджений на функціях безпеки, інтеграції оновлення погоди в режимі реального часу та безперебійному соціальному залученні, може значно підвищити корисність і привабливість цих інтерактивних карт для любителів катання на роликових ковзанах.

Базуючись на перевагах та недоліках вже існуючих рішень, можна виділити потребі і місця для покращення у власному застосунку:

Зручний інтерфейс: Інтуїтивно зрозумілий дизайн для легкої навігації та доступності.

Налаштування маршруту: Можливість для користувачів створювати та налаштовувати маршрути для катання на роликах відповідно до своїх уподобань.

GPS-відстеження та картографування: Функції точного GPS-відстеження та картографування для моніторингу місцезнаходження та планування маршруту в реальному часі.

Профілювання висоти: Відображення інформації про висоту, щоб допомогти користувачам підготуватися до різноманітних місцевостей і викликів.

Залучення спільноти: Інтеграція функцій соціальних медіа, щоб користувачі могли ділитися досвідом, маршрутами та досягненнями.

Попередження безпеки: Впровадження сповіщень у режимі реального часу про можливі небезпеки або проблеми з безпекою на вибраних маршрутах.

Відстеження продуктивності: Відстеження та аналіз таких показників катання на роликах, як відстань, швидкість і спалені калорії.

Офлайн доступність: Можливість для користувачів завантажувати карти для використання в автономному режимі, забезпечуючи функціональність у місцях з обмеженим підключенням.

Мінуси, які слід усунути та покращити:

Оптимізація продуктивності: Підвищити швидкість і швидкість реагування додатків, щоб забезпечити безперебійну роботу користувача.

Ефективність батареї: Оптимізоване енергоспоживання, щоб подовжити термін служби батареї пристрою під час використання.

Конфіденційність і безпека даних: Посилити заходи щодо захисту конфіденційності даних і переконатися, що дані користувачів обробляються безпечно й етично.

Виправлення помилок і стабільність: Усунення будь-які існуючі помилки, збої або збої, щоб підвищити стабільність програми .

Оновлення в реальному часі: Функції для оновлення в реальному часі стану маршруту, погоди та потенційних перешкод.

Персоналізовані рекомендації: Персоналізовані рекомендації щодо маршруту на основі вподобань користувачів і минулих дій.

Механізм зворотного зв'язку: Система зворотного зв’язку для збору думок користувачів і пропозицій щодо постійного вдосконалення.

Інклюзивність і доступність: Програма доступна для різноманітної бази користувачів, враховуючи різні мови, функції доступності та різні рівні знань.

2.2. Визначення вимог

Будь-яка веб-система повинна відповідати переліку вимог, щоб вдало виконувати поставлені перед нею задачі.

2.2.1. Призначення розробки

Система розробляється для забезпечення роботи веб-додатку інтерактивних мап для катання на роликах.

2.2.2. Функціональне призначення

Підсистема має забезпечувати наступні можливості:

- авторизація та аутентифікація користувачів
- збереження та відтворення інформації
- можливості сортування та пагінації
- засоби перегляду графічних зображень
- елементи оцінки
- логування

2.2.3. Експлуатаційне призначення

Підсистема розміщується на веб-сервері в середовищі під управлінням системою Linux. Безпосередньо до веб-додатку мають доступ всі користувачі з сучасною версією інтернет-браузера й стабільним доступом до мережі Інтернет. Ліцензійне програмне забезпечення може використовуватись лише аутентифікованими особами.

2.2.4. Функціональні вимоги

Функціональні вимоги описують ті можливості, які повинна мати система після завершення розробки.

Передбачаються можливості:

- створити акаунт користувача, що буде збережений у базі даних;
- пройти авторизацію у системі, використовуючи дані, вказані при створенні акаунту;
- обрати тип акаунту: любитель\спортсмен\тренер;

- у режимі тренера:
    - створити маршрут;
    - інтеграція з спортсменом\підопічним;
    - створення програми тренувань
- у режимі любителя:
    - переглянути список маршрутів;
    - залишити коментар;
    - задати питання автору маршрута;
    - переглянути історію пройдених маршрутів;
    - оцінити маршрут;
- у режимі спортсмена:
    - переглянути список маршрутів;
    - створити маршрут
    - залишити коментар;
    - задати питання автору маршрута;
    - переглянути історію пройдених маршрутів;
    - оцінити маршрут;
    - переглянути свою статистику
    - інтеграція з тренером

2.2.5. Нефункціональні вимоги

Описують, якою система має бути при взаємодії з користувачем.

Характеристики системи:

- система має високу швидкодію;
- бути легкою до освоєння;
- бути досить відмовостійкою при навантаженнях;
- не мати «бєкдорів» - багів, через які звичайний користувач може отримати неправомірний доступ до інформації

- може обробляти запити відразу великої кількості користувачів;
- правильно зберігати та відтворювати данні без спотворень;

2.3. Проектування системи

Проектування системи включає у себе огляд обраних інструментів, що будуть використовуватися під час розробки програмного забезпечення та створення прототипів у вигляді різноманітних діаграм, наприклад діаграми класів та взаємодій.

Завдяки описаним вище вимогам, буде значно легше зрозуміти, із яких елементів буде створена система. Аналіз вимог також дає краще розуміння того, які технології та фреймворки краще за все стануть у нагоді у якості безпосередніх частин системи.

2.3.1. Обрана мова програмування

Для розробки програмної системи вирішено обрати мову програмування Java. Спричинено це декількома факторами, які будуть розглянуті вище. У сукупності усі ці фактори створили чітку картину розуміння актуальності, переваг та недоліків обраного підходу до розробки.

Мова програмування Java – це мова високого рівня абстракції. Виходячи з цього, буде кращим вибором при роботі для створення складних навантажених систем, що мають відповідати певним стандартам у досить короткий час;

Завдяки великій популярності, мова Java має величезну підтримку суспільства, отже, можна розраховувати як на велику кількість актуальних стандартів та фреймворків, так й на велику кількість специфічного навчального матеріалу, завдяки чому розробляти системи стає набагато зручніше та легше. Так, серед наявних інструментів у мові, ключову роль займає Java Platform, Enterprise Edition (Java EE), що стало стандартом

корпоративного програмного забезпечення, керованого спільнотою. Java EE [4] розроблено з використанням Java Community Process [9] за участі великих експертів галузі, комерційних організацій та організацій з відкритим кодом, груп користувачів Java й незліченної кількості мотивованих людей. Кожен випуск інтегрує нові функції, що відповідають потребам галузі, покращують роботоздатність додатків і підвищують продуктивність розробників.

На основі Java EE [4] і було розроблено наступний ключовий елемент, завдяки якому розробка веб-додатків стає у рази ефективнішою.

2.3.2. Додаткові фреймворки та технології

Жодна розробка не обходиться без використання вже перевірених оптимальних рішень й сторонніх технологій. Ця теза ще більш актуальна в ентерпрайз-середовщі, до вкрай необхідна якість, гарна протестованість, швидкість та відмово стійкість.

Spring Framework дає комплексну модель програмування та конфігурації для сучасних корпоративних додатків на базі Java - на будь-якій платформі. Ключовим елементом Spring і є інфраструктурна підтримка на рівні додатку: Spring зосереджується на «підготовці» корпоративних додатків, щоб команди мали змогу зосередитися на бізнес-логіці на рівні додатку без зайвих зв’язків з конкретними середовищами роботи додатку [2, 10];

Spring Data JPA, частина більшого сімейства Spring Data, дозволяє дуже легко реалізувати репозиторії на основі JPA. Цей модуль має справу безпосередньо з розширеною підтримкою рівнів доступу до даних на основі JPA. Це значно полегшує створення програм на основі Spring, які використовують технології доступу до даних. Реалізація рівня доступу до даних програми була громіздкою протягом досить тривалого часу. Для виконання простих запитів, а також для розбиття на сторінки та аудиту потрібно написати занадто багато шаблонного коду. У розробників Spring Data JPA є на меті значно покращити реалізацію рівнів доступу до даних

системи , зменшуючи зусилля до необхідного обсягу. Як розробник, ви пишете інтерфейси свого репозиторію, включаючи лтше спеціальні методи пошуку, і Spring автоматично забезпечить реалізацію;

Hibernate ORM дозволяє сучасним розробникам легше писати програми, дані яких перебувають у процесі застосування. Як фреймворк об’єктного/реляційного відображення (ORM), Hibernate займається збереженням даних, оскільки він застосовується до реляційних баз даних (через JDBC) [12];

Gradle — це інструмент для керування проектами програмного забезпечення та його розуміння. На основі концепції саме об’єктної моделі проекту Grdle може керувати збіркою проекту, звітами та документацією з центральної частини інформації.

2.3.3. Обрані інструменти для розробки

Дуже важливою частиною під час роботи над додатком є наявність самме зручних та доступних інструментів для розробки, які мають актуальний та широкий функціонал.

Серед необхідних інструментів можна виділити як середовище розробки, так і додатки для дизайну та роботи з API.

- Середовище розробки IntelliJ IDEA [14]. Це інтелектуальна IDE, враховуючи контекст. Вона призначена саме для розробки різноманітних програм на Java та інших мов JVM, наприклад Kotlin, Scala та Groovy. Крім того, IntelliJ IDEA Ultimate допоможе у розробці веб-додатків: вона пропонує дуже ефективні вбудовані інструменти, підтримку JavaScript й пов’язаних з ним технологій, а також розширену підтримку таких популярних фреймворків, як Spring, Spring Boot, Jakarta EE, Micronaut, Quarkus й Helidon. Безкоштовні плагіни, розроблені JetBrains, дозволяють додатково розширити можливості IntelliJ IDEA та використовувати її для роботи з іншими мовами програмування в тому числі Go, Python, SQL, Ruby і PHP;

- DataGrip надає інструменти для роботи з об'єктами бази даних. У ситуації коли користувач створює чи змінює таблицю, додає чи змінює колонку, йому пригодиться графічний інтерфейс додатку JetBrains DataGrip. Подібні зміни супроводжуються генерацією відповідного до коду скрипту: можна відразу виконати зроблені зміни у базі або скопіювати згенерований DDL-запит у редакторі та працювати безпосередньо з кодом;
- Postman – це API-платформа задля створення та використання API. Postman спрощує кожен крок життєвого циклу API й спрощує співпрацю, щоб ви могли створювати кращі API – швидше й легше. Платформа Postman включає у себе повний набір інструментів, що допомагають прискорити саме життєвий цикл API – від проектування, тестування, документації й проектування до спільного використання та доступності ваших API.

2.3.4. Структурна схема

Проектування веб-системи починається з найвищого рівня абстракції, отже, із структурної схеми. Ця схема дозволяє виділити логічні робочі елементи вищого рівня, такя як коннектори, модулі, бази даних й інші сервіси, й безпосередні зв’язки між ними. Якісне проектування такої схеми дозволить грамотно розпределити роботу й обов’язки між командами розробників, створити план подальшої розробки. Взаємодія основнх блоків додатку зображена на рис. 2.5.

Після окреслення основних блоків, можна переходити до нижчих рівнів абстракції. Структурна схема додатку окреслює найвищий рівень абстракції та демонструє зв'язок між елементами системи. На цій діаграмі представлено всю систему з висоти пташиного польоту, ілюструючи основні компоненти або підсистеми та їхні взаємозв’язки.

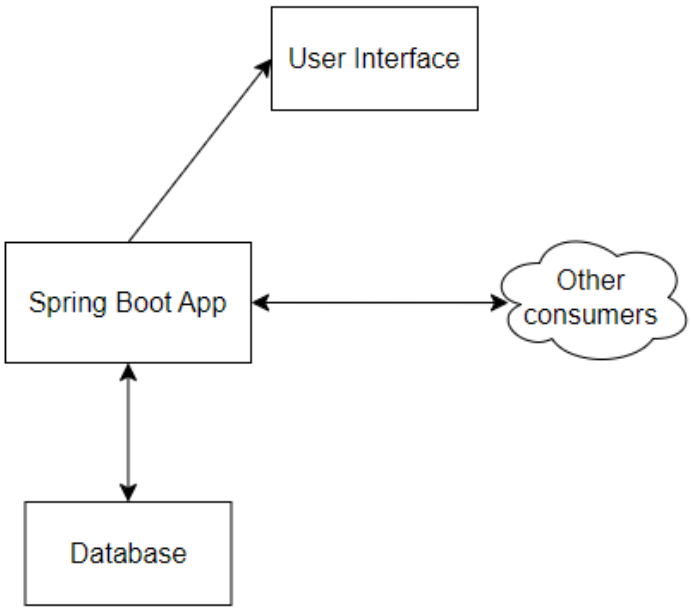

Рис. 2.5. Структура додатку

На цьому рівні увага зосереджена на інкапсуляції фундаментальних сутностей у системі без заглиблення в складні деталі. Абстракційна структурна діаграма найвищого рівня діє як базова карта, допомагаючи зацікавленим сторонам і особам, які приймають рішення високого рівня, з першого погляду зрозуміти архітектуру системи. Це допомагає встановити загальне розуміння меж системи, її зовнішніх інтерфейсів і первинних взаємодій між її основними компонентами, закладаючи основу для більш детальних і конкретних структурних діаграм на нижчих рівнях абстракції в ієрархії системи.

Тож, далі розроблено абстрактну діаграму класів, х яких буде складатися підсистема обробки даних користувача (рис. 2.4). Подібні діаграми взагалом використовуються для моделювання структури додатків, що розробляються за об’єктно-орієнтованою методологією [15].

Але, на діаграмі високого рівня, зазвичай, не зазначаються конкретні методи чи поля, фінальні назви класів. Має місце бути також зазначення

даних, які переходять між класами та елементами для більшої зрозумілості та зручності.

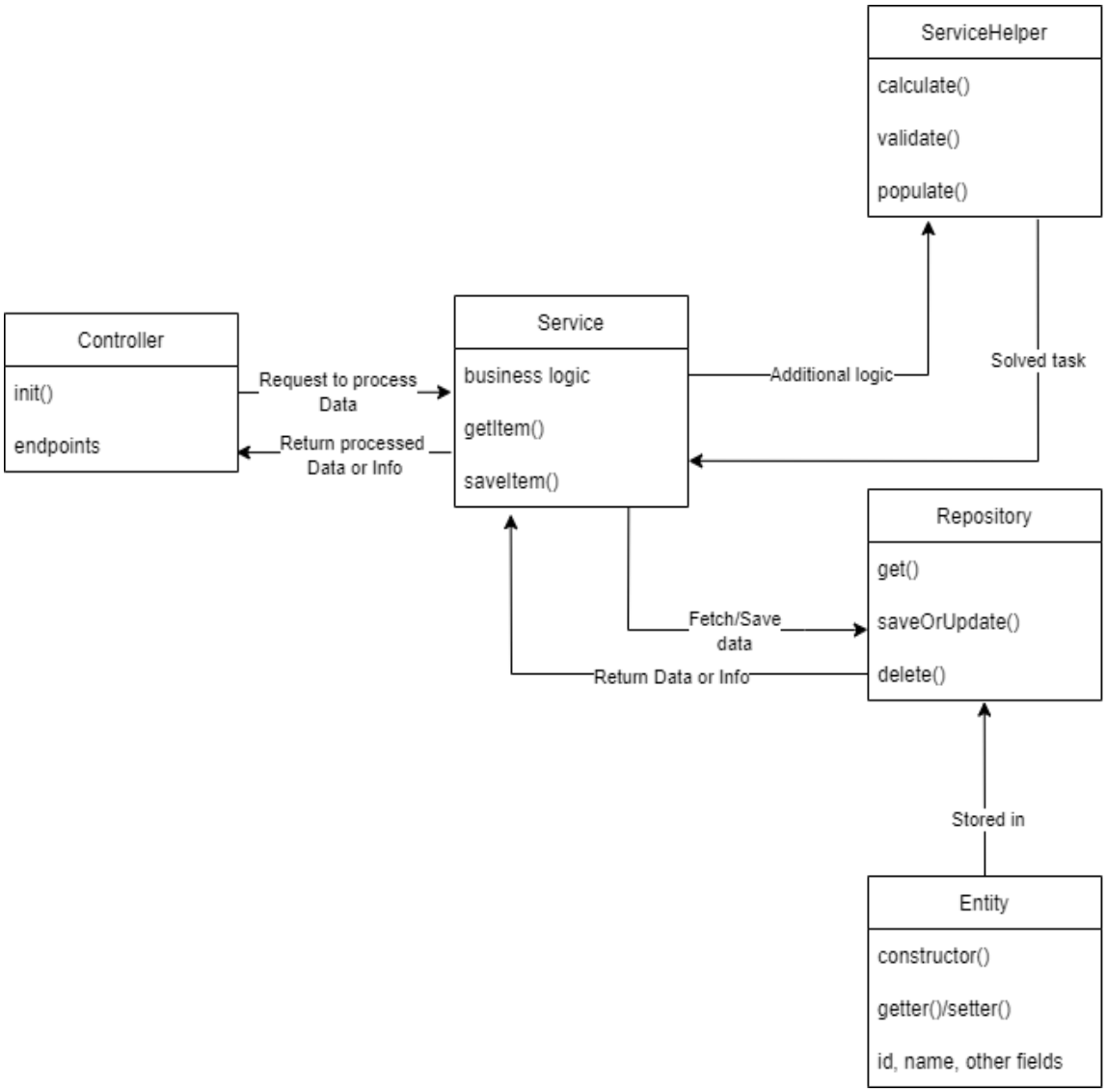

Рис. 2.6 – Абстрактні класи додатку

Далі можна розробити наступну діаграму прецедентів. Подібна схема визначає та описує бізнес логіку високого рівня, що дозволяє зрозуміти, на які класи та сутності буде найбільший сенс декомпозувати додаток. Функціональність зображається у вигляді «прецедентів використання» [15], які являють собою типову взаємодію всередені системи, що в той самий час описує взаємодію користувача з самою системою. Діаграму зображено на рис. 2.6.

При розробці великих та сильнонавантажених систем із складною бізнес логікою, дуже сачто мають місце бути дуже детальні діаграми, що будуть торкатись конктертних та атомарних транзакцій. Проте, подібні деталі зазвичай виносяться у окремі діаграми.

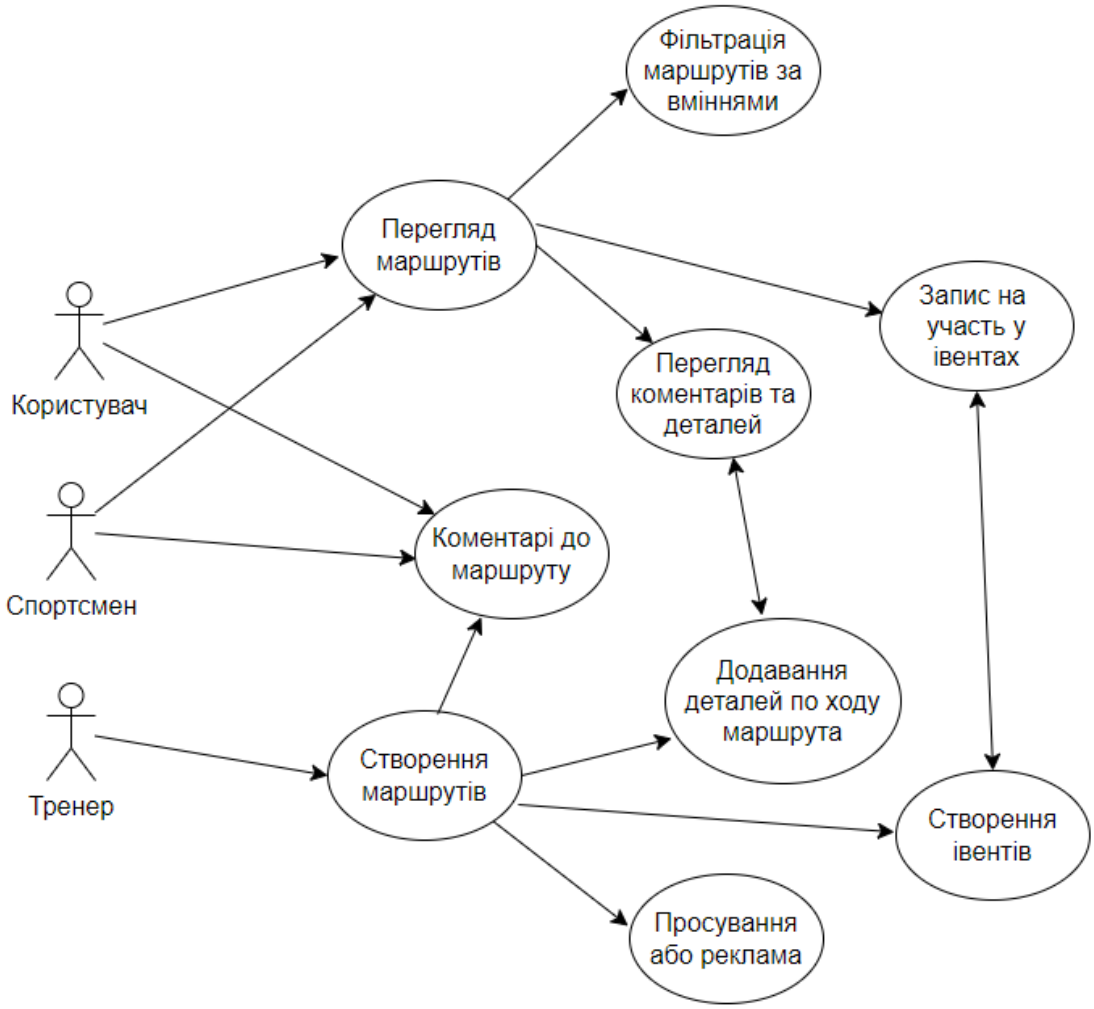

Рис. 2.7. Діаграма прецедентів

Прецедент, по існуючому стандарту створення таких діаграм, позначається на ній овалом, пов'язаним лініями із іншими користувачами, яких прийнято називати «діючими особами» (актори, actors). Дійові особи, у свою чергу, взаємодіють із самоюсистемою (або використовуються системою) в даному конкретному прецеденті.

У такому сценарії дійова особа бере на себе чітко визначену роль. Хоча діаграма може зображати лише одного основного учасника, у системі цю роль можуть виконувати численні фактичні користувачі. Компіляція всіх таких прецедентів ефективно окреслює функціональні вимоги до програми, формуючи основу для розробки наступних технічних специфікацій для створення системи. У діаграмах прецедентів, крім ілюстрації зв'язків між прецедентами та акторами, два додаткових типи зв'язків, а саме «використання» та «розширення», можуть пов'язувати прецеденти. Відношення «розширення» стає застосовним, коли один прецедент схожий на інший, але несе дещо підвищене функціональне навантаження. Цей зв'язок використовується для формулювання змін у звичайній поведінці системи. [15].

Далі, на рис. 2.8-2.10, зображено дікілька прикладів уточнення якогось з наявних прецедентів, у вигляді так званої «діаграми послідовностей». Такі діаграми спрямовані на уточнення конкретних деталей бізнес логіки.

Рис. 2.8. – Діаграма послідовностей створення маршруту

Рис. 2.9. – Діаграма послідовностей пошуку маршрута користувачем

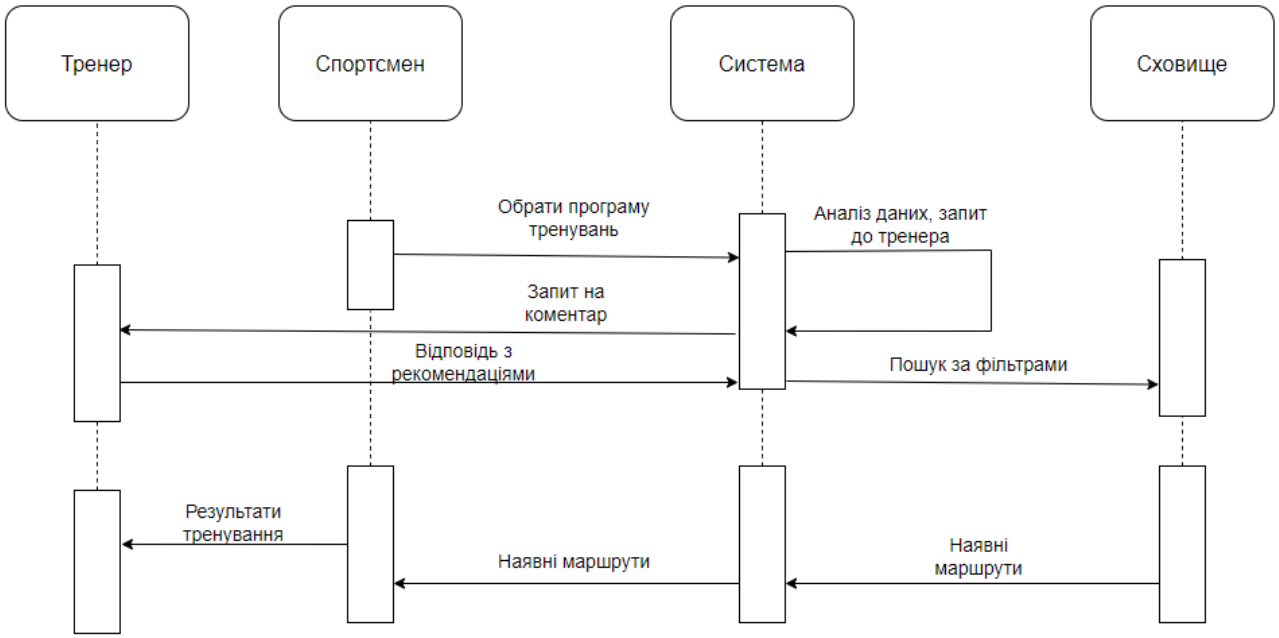

Рис. 2.10. – Діаграма послідовностей взаємодії спортсмена з тренером

2.4. Висновки до розділу

Після заглиблення в тему даного розділу стає очевидним, що попереднє визначення вимог є ключовим етапом розробки додатків, що передує етапу проектування. Аналіз подібних систем допомагає виявити й пом’якшити потенційні недоліки, виявлені в порівнянних програмах, дозволяючи розробити обґрунтовану систему, яка вирішує ці проблеми. Чітко визначений набір вимог забезпечує надійну основу для побудови інфраструктури програми, яка згодом набуває форми у вигляді більш детальних структурних схем.

Структурні діаграми та діаграми класів відіграють вирішальну роль у створенні абстрактної, але раціональної моделі програми. Ретельний процес декомпозиції додатково полегшує розуміння того, чи модулі проекту були належним чином розділені та чи розумно продовжувати або переглядати верхню структуру програми. Не менш важливими є діаграми прецедентів, які пояснюють бізнес-логіку як високого, так і нижчого рівнів. Кожен актор у прецеденті виконує окрему роль, вирішальну для процесу побудови.

Хоча діаграма може зображувати лише одного оригінального актора, важливо визнати, що може бути багато реальних користувачів, які беруть на себе цю роль по відношенню до системи. Вичерпний перелік усіх прецедентів служить основою для визначення функціональних вимог програми, формуючи основу для подальшого технічного завдання на створення системи. Після завершення діаграм і розробки структури наступним кроком є перехід безпосередньо до реалізації програмного додатку.

На основі даних та знахідок з цього розділу, вважається, що можна перейти до безпосередньої реалізації застосунку. Звісно, у процесі розробки можуть і будуть розкриватися нові деталі та проблеми, які необхідно буде вирішувати досить оперативно. Але, кожному розробнику слід це розуміти та бути готовим до оперативного внесення змін.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ ІНТЕРАКТИВНОЇ МАПИ ДЛЯ КАТАННЯ НА РОЛИКАХ

3.1. Розробка веб-додатку

У цьому розділі описана бозпосередня розробка веб-додатку для перегдяу мап. Для грамотного ведення розробки за ітеративним циклом, буде доречним створення системи контролю виконання завдань. Хоча, повний функціонал системи не буде використано, так як це потребує наявності команди хочаб з декількох людей. Проте, навіть при роботі у команді з однією людиною, середовище Jira виступає як дуже звичний та зручний інструмент для трекінгу завдань щодо розробки системи. В цьому випадку, фактично, дії в кожній колонці буде виконувати одна людина. Спершу завдання поступатимуть у так званий «беклог», де буде проведено перевірку та додавання додаткової інормації. Далі, виконане завдання переходить у стовпчик «тестування», де та сама людина буде йти по необхідним тест-кейсам та звітує чито про успішне завершення, чито про необхідність виправлення непрацюючого коду.

3.1.1. Система контролю розробки

Беручи наявний досвід із комерційної розробки, вирішено використовувати систему Jira, що гарно підходить саме для роботи із методолгією SRUM, та для контролю розробки взагалом. На рисунку 3.1 зображено «борд», що демонструє набір завдань, кожне з яких знаходиться на своїй стадії виконання та готовності.

По мірі розробки, нові завдання додаються у так званий «беклог» (рис.3.1), що було описано у розділі 1. Безпосереднь звідси завдання набираються у «спрінт» та своєчасно виконуються під час нього. Хоч робота і ведеться однією людиною, поняття «спрінту» все-одно допомагає

витримувати часові рамки та вписуватись у дедлайни. При відсутності таких рамок розробка дуже часто затягується на непередбачувано довгий час.

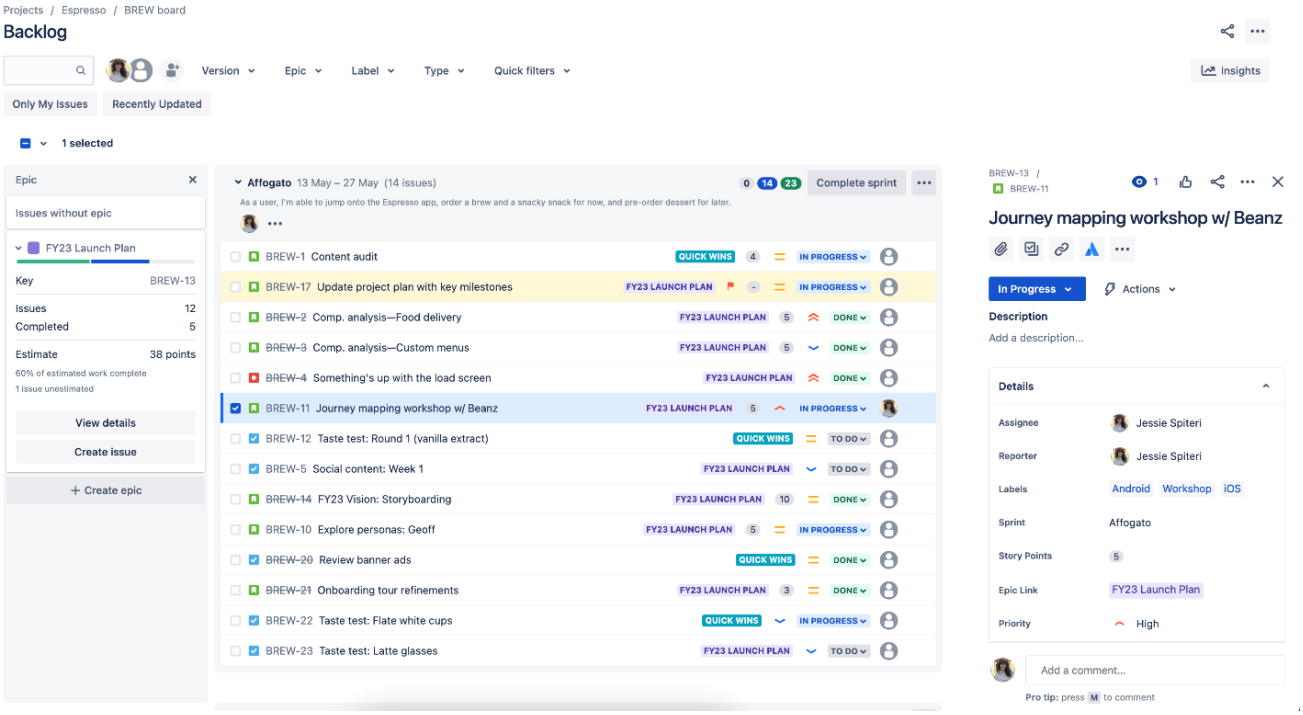

Рис. 3.1 – Приклад вигляду беклога проекту

Час, витрачений на кожне із завдань документується, за для контролю на часом виконання та ресурсами команди уцілому. Так, після проведення певної кількості хвилин за роботою над завданням, розробник «логує», скільки вже було витрачено (Рис. 3.2). Логування точної кількості часу витраченого на завдання допомагає оцінити темп розробки проекту та по можливості внести корективи.

Також, це слугує як маркер правильності оцінки складності завдань. Таким чином, якщо завдання призначені на недь виконуються за два і більше – це явний признак того, що складність завдання було оцінено невірно, а саме – недооцінено. Грамотна оцінка складності напрацьовується з досвідом, тож в перший час відхилення є нормою.

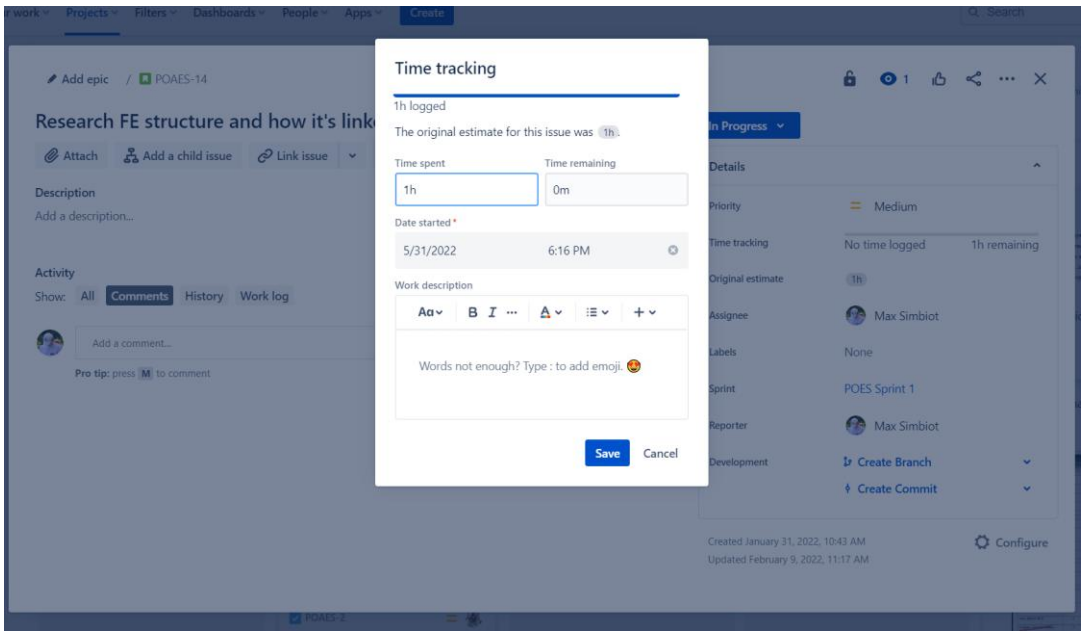

Рис. 3.2 – Логування часу

Важливою частиною у організації розробки програмного забезпечення завжди є правильна організація та ведення контролю версій. З цієї мети було створено приватний GitHub репозиторій, на який завантажувалась кодова база. (Рис. 3.3). Звісно, за домопогою подібного інструмента можна реалізувати й CI\CD систему, проте це є актуальним при більшому розмірі як самого проекта, так і команди вцілому.

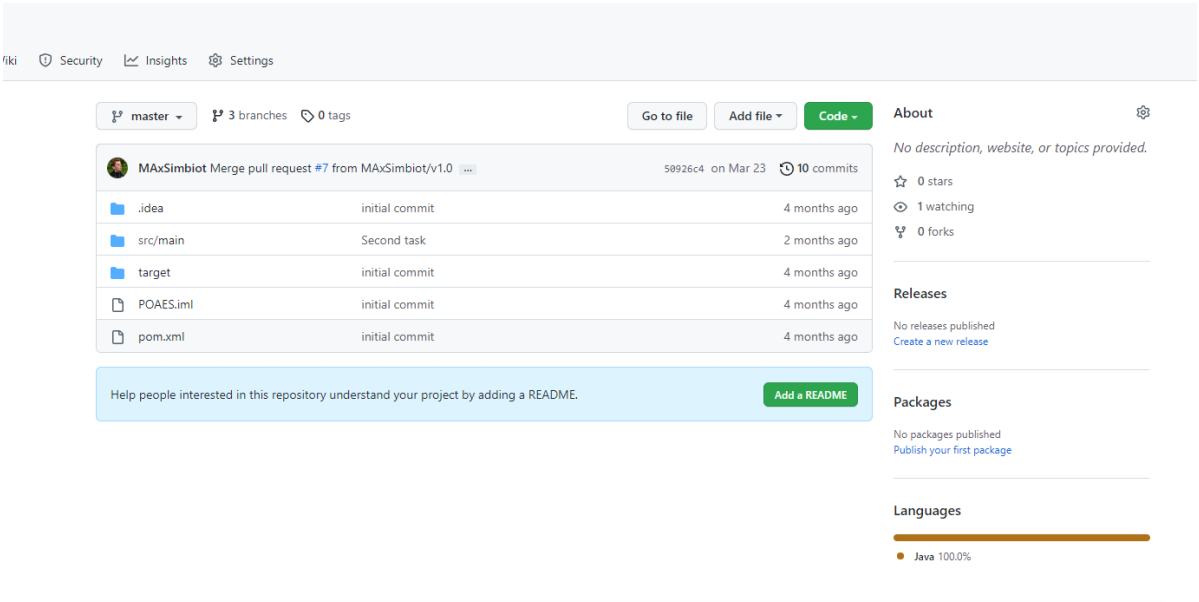

Рис. 3.3 – Репозиторія GitHub

3.1.2. Створення кодової бази

Другим за тривалістю після проектування та підготовки є написання коду, створення та налаштування елементів додатку. Перший етап – створення структури директорій та основних класів, файлів налаштувань [1] (Рис. 3.4). Подібна структура може бути згенерована автоматично, відповідно до типу проекту. Проте, деякі додаткові директорії чи файли налаштувань, як-от settings.gradle необхідно створити самостійно.

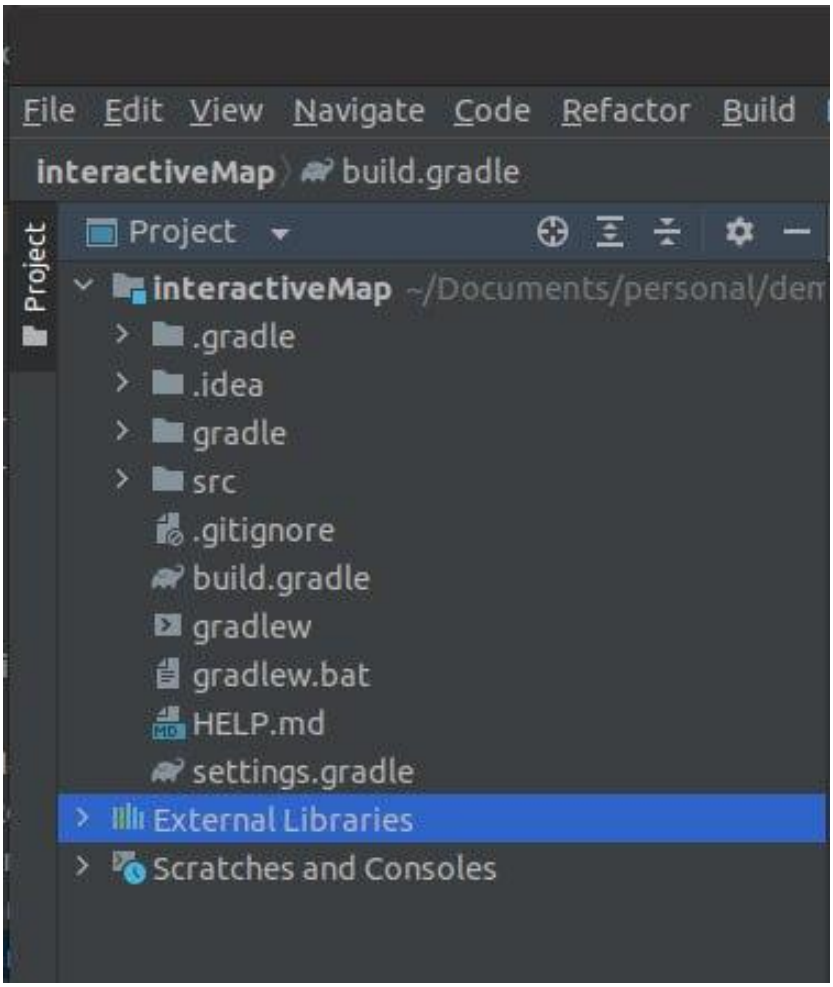

Рис. 3.4 – Структура файлів додатку

При використанні gradle, необхідні налаштування та залежності для проекту необхідно прописати у відповідному файлі build.gradle (Рис. 3.5). В першу чергу необхідно вказати мову програмування та версію. Далі, вказано версію самого додатку та назву артефакту. Окремі репозиторії необхідно вказати у випадках, коли базова mavenCentral не має необхідної залежності.

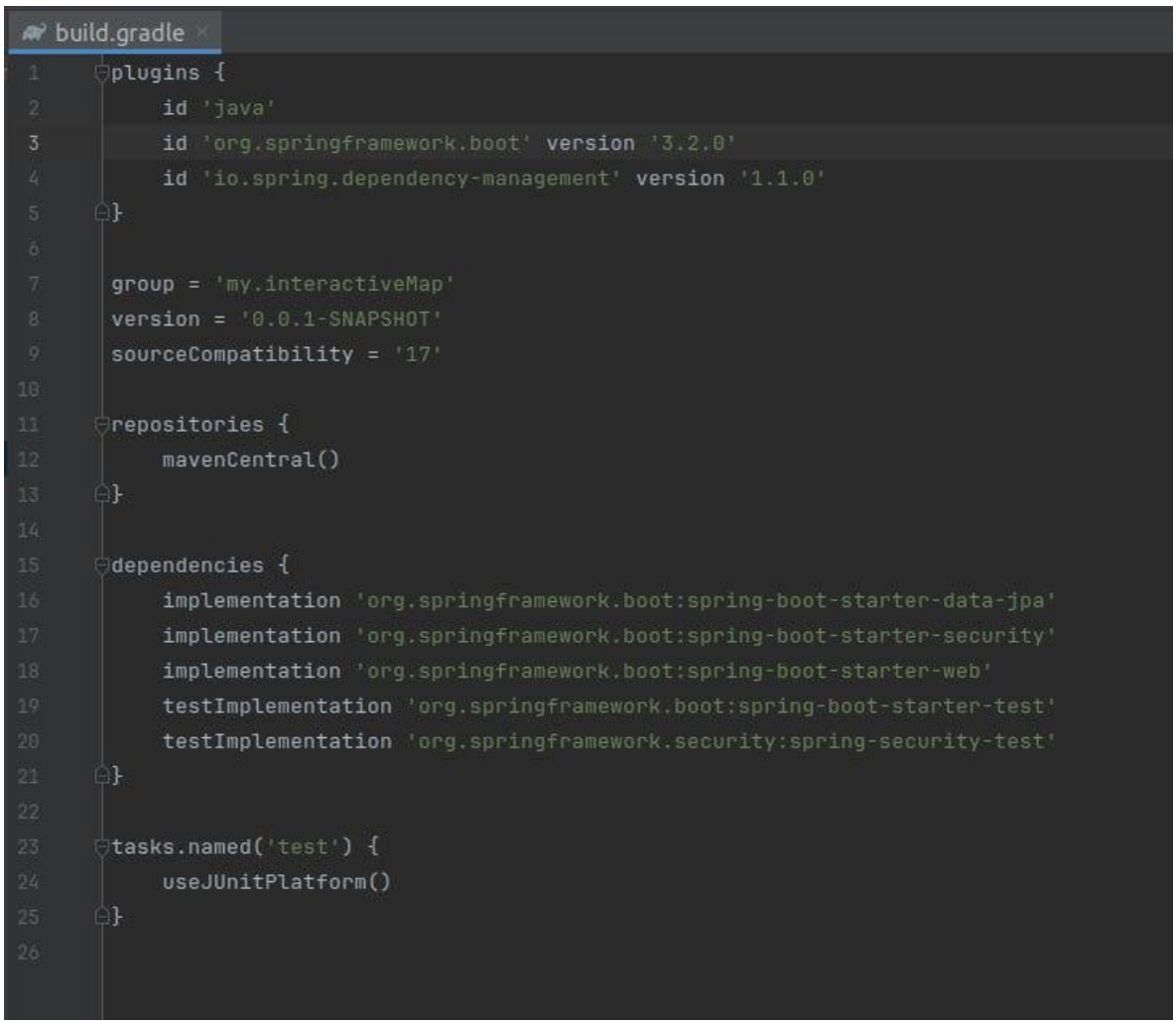

Рис. 3.5 – Файл build.gradle

У файл відразу додано залежності на spring boot у вигляді «стартерів». Стартери Spring Boot — це попередньо налаштовані шаблони або залежності, які спрощують процес налаштування та розробки різних аспектів програми Spring Boot. Вони розроблені, щоб забезпечити швидкий і зручний спосіб завантаження різних типів проектів з мінімальними зусиллями. Стартери Spring Boot зазвичай включають комбінацію залежностей, файлів конфігурації та шаблонного коду, який узгоджується з конкретними випадками використання або функціями.

Далі, створювалися класи-сутності, інтерфейси, контроллери, сервіси, та інше необхідне наповнення. Для роботи деяких фреймворків необхідно створити спеціальні класи-конфігуратори[10], помічені відповідними

аннотаціями. Такі класи виконують функцію конфігурації у коді, що має свої недоліки і переваги. Перевагою в нашому випадку є те, що конфігурація значно спрощується.

Лістинг 1.1.

```
// Route.java
@Entity
public class Route {

      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private Long id;

      @OneToMany(cascade = CascadeType.ALL) private List<Point> points;

      // getters and setters, equals and hashcode
}


// Point.java
@Entity
public class Point {
      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private Long id;

      private double latitude;
      private double longitude;

    // getters and setters, equals and hashcode
}
```

У листингу 1.1 зазначено уривок класу Route, що представляє собою список точок (Point), з яких і створюється маршрут. Так як у додатку справа йде з картою, логічним буде наступна структура класів(рис. 3.6), при якому клас шлях буде у відносинах типу “has a” класу точка.

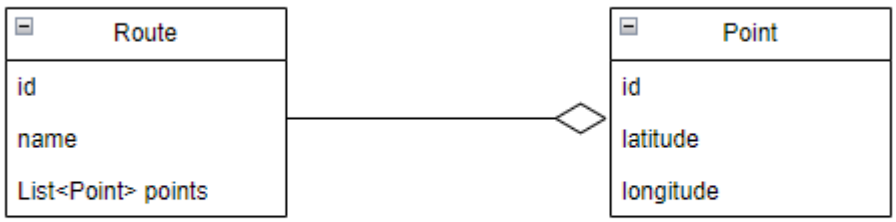

Рис. 3.6 – Зв'язок класів

Точка, в свою чергу, слугує як географічна координата на карті, через яку в результаті і буде проходити маршрут. Таким самим чином було розроблено і інші необхідні для роботи додатку класи, такі як коментар, користувач, акаунт, програма тренувань та ін.

Також було вирішено використовувати фреймворк Project Lombok для прискорення розробки. Він відповідає за автогенерацію коду з використанням аннотацій. Наприклад, клас, помічений анотацією @AllArgsConstructor, @Getter, @Setter та @ToString буде автоматично доповнений одноіменними методами під час компіляції, що значно прискорює процесс розробки та робить код більш читабельним.

Далі було створено класи, що необхідні для роботи додатку та комунікації між сторінкою, бізнес-логікою та базою даних. Так, на листингу 1.2 можна бачити стислу версію так званого контролера. Контролер слугує методом-точкою входу для запитів, що приймає виклики за адресою, вказаною у анотації `@GetMapping`.

У Spring `@GetMapping` і `@PostMapping` — це анотації, які використовуються для відображення методів HTTP на певні методи контролера. Вони є частиною веб-модуля Spring і зазвичай використовуються в розробці веб-служб і веб-додатків RESTful.

Листинг 1.2.

```
// MapController.java
@Controller
public class MapController {
    @Autowired
    private RouteService routeService;

    @GetMapping("/map")
    public String showMap(Model model) {
        // Retrieve and pass data to the view
        return "map";
    }

    @PostMapping("/addRoute")
    public String addRoute(@RequestParam List<Point> points, Model model)
{
        // Save the route with the specified points
        return "redirect:/map";
    }
```

Також необхідно створити так званий service layer – набір класів, які будуть визиватись у методах контролера та матимуть найбільшу частину бізнес-логіки.

Сервісні класи, частіше за все, слугують як основна частина додатку. У контексті інтерактивної карти для програми катання на роликових ковзанах рівень обслуговування відіграє вирішальну роль у обробці бізнес-логіки та координації взаємодії між контролером і рівнем доступу до даних.

Основні класи, що є сенс створити в першу чергу:

MapService:

Призначення: Керує загальною функціональністю, пов’язаною з інтерактивною картою.

Обов'язки: Отримання та обробка картографічних даних. Керування послугами геолокації для відстеження користувачів. Надати методи розрахунку та оптимізації маршруту.

UserService:

Призначення: Керує функціями, пов’язаними з користувачем. Обов'язки: Аутентифікація та авторизація користувача. Керування профілем користувача. Обробка параметрів і налаштувань, пов’язаних із картою. Керування створеним користувачами контентом, наприклад оглядами та коментарями.

RouteService:

Призначення: керує функціями, пов’язаними з маршрутами для катання на роликах. Обов'язки: Отримання та оновлення інформації про маршрут. Розрахунок і пропозиція нових маршрутів на основі вподобань користувача. Керування збереженими або улюбленими маршрутами користувача.

EventService:

Призначення: Керує подіями, пов’язаними з катанням на роликах. Обов'язки: Створення, оновлення та видалення подій на карті. Організація

реєстрації та відвідування заходів. Надання інформації про майбутні роликові події.

ReviewService:

Призначення: керує відгуками користувачів і оцінками маршрутів і визначних місць. Обов'язки: Надсилання та отримання відгуків про маршрути чи місця. Розрахунок середніх оцінок для маршрутів і місць. Модерування та керування контентом, створеним користувачами.

Загальна структура сервіс-класу приведена на листингу 1.3. Сервіс-клас єдиний має прямий доступ до репозиторії та мають реалізовану складу бізнес-логіку, відповідно до виставлених вимог.

Листинг 1.3.

```
// RouteService.java
@Service
public class RouteService {
    @Autowired
    private RouteRepository routeRepository;

    @Autowired
    private PointRepository pointRepository;

    // Implemented methods and business logic omitted
}
```

Останнім кроком розробки є створення візуального інтерфейсу, що надасть можливість створення, редагування та перегляду мап. Для спрощення розробки системи введення у наявний стислий час, вирішено використати API Mapbox.

Mapbox — це картографічна платформа, яка надає розробникам інструменти та API для інтеграції настроюваних інтерактивних карт у свої програми. Mapbox API дозволяє розробникам отримувати доступ до різноманітних картографічних сервісів, включаючи візуалізацію карт, геокодування, напрямки та просторовий аналіз.

Для даної версії додатку необхідні два основних елементи Mapbox API:

Mapbox Maps API: Візуалізація карт.

Основна функція Mapbox — це рендеринг карт. Надає можливість використовувати Mapbox Maps API, щоб вставляти інтерактивні карти з можливістю налаштування у власні веб-та мобільні програми. Це включає параметри додавання маркерів, спливаючих вікон і шарів на карту.

Стилі та теми:

Mapbox дозволяє швидко та легко створювати власні стилі карт або використовувати попередньо розроблені теми, щоб відповідати заданому стилю зовнішньому вигляду програм. Цей рівень налаштування включає можливість керувати кольорами, шрифтами та елементами карти.

Для створення візуального інтерфейсу вирішено використати шаблонізатор Thymeleaf. Thymeleaf — це механізм шаблонів Java на стороні сервера, розроблений для Інтернету та автономних середовищ. Він зручний для використання у веб-додатках на основі Java, зокрема у Spring Framework, для створення динамічних та інтерактивних веб-сторінок. Thymeleaf служить механізмом шаблонів, що дозволяє легко інтегрувати динамічний вміст у шаблони HTML.

На листингу 1.4 передано основний вміст шаблону HTML з використанням Thymeleaf. На представленій у лістингу сторінці описується створення маршруту, з подальшим викликом ендпоінта для валідації та запису у додаток.

Листинг 1.4.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Map Application</title>
    <!-- Include Mapbox CSS -->
    <link href="https://api.mapbox.com/mapbox-gl-js/v2.6.1/mapbox-gl.css"
rel="stylesheet">
    <style>
        #map {
            height: 400px;
        }
    </style>
</head>
<body>
    <h2>Interactive Map Application</h2>
```

```
    <div id="map"></div>

    <!-- Add Route -->
    <form id="routeForm">
        <label for="pointLat">Latitude:</label>
        <input type="text" id="pointLat" name="pointLat" required>

        <label for="pointLng">Longitude:</label>
        <input type="text" id="pointLng" name="pointLng" required>

        <button type="button" onclick="addPoint()">Add Point</button>
        <button type="button" onclick="submitRoute()">Submit
Route</button>
    </form>

    <!-- Include Mapbox JS -->
    <script src="https://api.mapbox.com/mapbox-gl-js/v2.6.1/mapbox-
gl.js"></script>


    <script th:inline="javascript">
        // Mapbox API key
        const mapboxApiKey = /*[[ ${'secret'} ]]*/

        // Initialize Mapbox map
        mapboxgl.accessToken = mapboxApiKey;
        const map = new mapboxgl.Map({
            container: 'map',
            style: 'mapbox://styles/mapbox/streets-v11',
            center: [0, 0], // Set your default map center
            zoom: 2 // Set your default zoom level
        });

        const routePoints = [];
        function addPoint() {
            const lat =
parseFloat(document.getElementById('pointLat').value);
            const lng =
parseFloat(document.getElementById('pointLng').value);

            if (!isNaN(lat) && !isNaN(lng)) {
                routePoints.push({ lat, lng });
                // Add code to display markers on the map for the added
points
            }


            document.getElementById('pointLat').value = '';
            document.getElementById('pointLng').value = '';
        }


        function submitRoute() {

Spring Controller)
            console.log(routePoints);
            routePoints.length = 0; // Clear the routePoints array
        }
    </script>
```

```
</body>
</html>
```

Під час наступних кроків, ітеративно, невиликими та прорахованими кроками було розроблено кодову базу проекту. В результаті кропіткої розробки маємо підсистему обробки даних користувача та простий візуальний інтерфейс що використовує вищезазначені фреймворки та розроблена за архітектурою REST API, що дозволяє досить легко інтегрувати додаткові інтерфейси із бєк-енд підсистемою, так як у інтерфейсі буде достатньо звернутися до ендпоінтів для спілкування.

Такий підхід додає гнучкості, так як дозволяє у подальшому розширювати додаток на інші платформи. За умови, що інтерфейс введення (бекенд) залишається незмінним, будь-який фронтенд, що буде здатен інтерпретувати передані точки у мапу та зможе показати мапу з точками для введення даних зможе працювати з системою. Таким чином, розширення на різні браузери та навіть на мобільні платформи стає значно легшим.

3.2 Тестування додатку

Тестування є критично важливим аспектом розробки програмного забезпечення, і його важливість поширюється навіть на проекти з єдиним розробником. Незважаючи на, здавалося б, невеликий масштаб проекту, який виконує одна особа, тестування служить запобіжником від потенційних проблем, забезпечуючи надійність, функціональність і загальну якість програмного забезпечення.

Застосовуючи методи ретельного тестування, розробник може виявити та усунути помилки, помилки або несподівані дії на ранніх стадіях процесу розробки. Цей проактивний підхід не тільки веде до більш надійної та стабільної програми, але й економить час і зусилля в довгостроковій перспективі. Тестування забезпечує певний рівень довіри до кодової бази, дозволяючи розробнику вносити зміни або запроваджувати нові функції з

упевненістю, що існуючі функції залишаться незмінними. Крім того, він слугує документацією, надаючи розуміння очікуваної поведінки коду. По суті, тестування є фундаментальною дисципліною, яка сприяє загальному успіху проекту, навіть якщо воно виконується окремим розробником, сприяючи надійності коду, зручності обслуговування та позитивному досвіду користувача.

Далі представлено декілька тест-сценаріїв для перевірки базових функцій додатку. Звісно, при наявності достатньої кількості часу та росту проекту, тестові сценарії ставатимуть все детальнішими.

Таблиця 2.1

Тестуваня за тест-кейсами

Номер	**Опис**	**Тестовий сценарій**	**Статус**
1	Карта завантажується правильно та спочатку центрується на місці за замовчуванням.	Відкрити додаток для катання на роликах. Переконатись, що карта відображається. Переконатись, що карта розташована в центрі попередньо визначеного місця.	**Пройдено**
2	Користувачі можуть збільшувати та зменшувати масштаб, а також панорамувати карту	Використати кнопки масштабування, щоб збільшувати або зменшувати масштаб карти. Перетягувати карту, щоб панорамувати в різних напрямках. Переконатись, що масштабування та панорамування плавні та точні.	**Пройдено**

Продовження таблиці 2.1.

3	Об’єкти міста точно відображаються на карті.	Переконатись, що попередньо визначені POI (наприклад, парки, орієнтири) видно на карті. Натисніть POI та переконайтеся, що відображається додаткова інформація.	**Пройдено**
4	Події правильно позначені та відображеніу відповідному меню	Створити подію катання на роликах за допомогою програми. Переконатись, що маркер події додано у розділ «події». Натиснути на маркер події, щоб переглянути деталі події.	**Пройдено**
5	Інтерфейс карти реагує на екрани різних розмірів.	Відкрити програму на пристроях із різними розмірами екрана (наприклад, смартфонах і планшетах). Переконатись, що карта відповідає розміру екрана без втрати функціональності.	**Пройдено**
6	Реакція програми на несподівані сценарії.	Ввести недійсні координати або адреси для розрахунку маршруту. Переконатись, що програма надає змістовні повідомлення про помилки.	**Пройдено**

Закінчення таблиці 2.1.

7	Створення акаунту	Створіть акаунт користувача Створіть акаунт для кожної групи користувачів (Любитель, Спортсмен, Тренер)	**Пройдено**
8	Права користувача працюють коректно	Зайдіть у акаунт користувача «Любитель», «Спортсмен» та переконатись, що він не має тих самих прав, що і тренер	**Пройдено**
9	Залишити рейтинг маршрута	Залишити рейтинг для маршрута Переконатись що рейтинг збережено коректо Переконатись що не можна залишити рейтинг менше 0 та більше 5 балів	**Пройдено**
10	Залишити коментар до маршрута	Залишити коментар на маршруті Переконатись що коментар зберігається та відображається коректно	**Пройдено**

Після перевірки основних тестових сценаріїв, можна вважати, що додаток готовий до так званого «бета-тестування». Бета-тестування — це етап розробки програмного забезпечення, на якому попередня версія продукту, яку часто називають бета-версією, стає доступною для вибраної групи зовнішніх користувачів для тестування та оцінки.

Цей етап тестування дозволяє розробникам збирати відгуки про продуктивність продукту, виявляти потенційні помилки чи проблеми та оцінювати задоволеність користувачів у реальному середовищі. Бета-тестери, які зазвичай не є частиною команди розробників, досліджують програмне

забезпечення за звичайних умов використання та надають цінну інформацію про його функціональність, зручність використання та загальну взаємодію з користувачем.

Бета-тестування допомагає розробникам виявити проблеми, які, можливо, не були очевидними під час внутрішнього тестування, і дозволяє вносити вдосконалення до офіційного випуску, забезпечуючи більш досконалий і надійний кінцевий продукт.

Лише після проходження якісного бета-тестування та виправлення знайдених помилок та проблем, можна вважати додаток готовим до релізу та запуску у широкі маси, де користувачі по всьому світу зможуть насолоджуватись функціоналом додатку.

3.3. Висновки до розділу

Роблячи висновки з роботи над даним розділом можна зазначити, що для розробки веб-додатку було вдало обрано мову програмування Java 21. Мова за своєю суттю є високорівневою, що дозволяє зосередитися на абстрактній конструкції програми, не заглиблюючись у складні деталі, такі як організація пам’яті.

Велика кількість і якість фреймворків відіграють ключову роль у перетворенні завдання розробки на зрозумілу та логічно побудовану справу. Примітно, що такі фреймворки, як Spring, беруть на себе такі функції, як керування транзакціями, значно спрощуючи взаємодію з базою даних. Для інструменту розробки було вибрано середовище IntelliJ Idea завдяки його перевіреній функціональності та можливостям, продемонстрованим у роботі над промисловими проектами, де компроміс не є варіантом у виборі щоденних інструментів розробника. У поєднанні з DataGrip, надійним середовищем керування базами даних, ці інструменти стали кращим вибором для професіоналів у різних галузях.

Їх надійність і ефективність гарно показали себе протягом усього процесу розробки програми. На шляху розробки додатка було пройдено

початковий етап, що включав налаштування основних параметрів проекту, щоб забезпечити його безперебійну роботу. Згодом розробка включала створення класів сутностей, класів обслуговування, репозиторіїв, що полегшують доступ до бази даних, і контролерів. Крім того, на етапі розробки було введено спеціальні класи конфігурації, які слугували параметрами для вищезгаданих фреймворків, додатково оптимізуючи продуктивність і функціональність програми.

По мірі плину розробки постійне тестування й удосконалення були невід'ємними компонентами, які гарантували, що програма не лише відповідає початковим параметрам проекту, але й узгоджується з вимогами, що розвиваються, і найкращими галузевими практиками. Прагнення до досконалості як у виборі інструментів, так і в розробці підкреслює зацікавленість до створення високоякісної та надійної інтерактивної карти для любителів катання на роликових ковзанах.

ВИСНОВКИ

Результатом успішного завершення дипломного проекту стала розробка інтерактивної карти, створеної спеціально для ентузіастів катання на роликових ковзанах, яка відповідає переважаючій тенденції цифровізації суспільства. У сучасному ландшафті важливість універсальної веб-програми, доступної для користувачів із сучасними пристроями, визнається в різних доменах. Розроблена веб-система прагне забезпечити ентузіастів катання на роликах швидкий доступ до маршрутів, подій і відповідної інформації, сприяючи безперебійному зв’язку між спільнотою катання на роликах і цифровою ерою.

Перед розробкою було проведено ретельний аналіз подібних систем і формулювання комплексних вимог, щоб забезпечити створення ефективної системи в обмежені терміни. Розумний вибір технологій, фреймворків і інструментів розробки, таких як Spring Framework, Hibernate і Maven, був ключовим моментом для оптимізації швидкості та якості розробки. Прийняття методології SCRUM у поєднанні з інтеграцією програмного забезпечення Jira з практик промислової розробки прищепило структурований ритм процесу розробки, перетворивши його з хаотичних зусиль на циклічні робочі цикли, що характеризуються вдосконаленням кодової бази та інтеграцією ключових особливості.

Цей організований підхід не тільки підвищує моральний дух команди розробників, але й прискорює темп і забезпечує якість процесу розробки, причому документація розвивається синхронно з розширенням кодової бази. Інтерактивна карта для любителів катання на роликах служить динамічним інструментом для навігації та вивчення маршрутів, подій і цікавих місць для катання на роликах. Зі зростанням поширення цифрових платформ першочерговим стає наявність зручного та доступного веб-додатку для спільноти любителів роликових ковзанів. Карта не лише надає практичну

інформацію про популярні маршрути та події, але й сприяє створенню відчуття спільності, дозволяючи користувачам ділитися досвідом, відкривати нові місця та бути в курсі майбутніх зустрічей.

Ретельний аналіз подібних систем і комплексних вимог у поєднанні зі стратегічним вибором таких технологій, як Spring Framework, Hibernate і Maven, проклали шлях до ефективної розробки цієї спеціалізованої карти. Структурований підхід до розробки, який підтримується методологією SCRUM і програмним забезпеченням Jira, гарантує, що карта не тільки відповідає функціональним потребам любителів катання на роликових ковзанах, але й відповідає високим стандартам якості, зручності використання та адаптивності в постійно змінюваному ландшафті цифрового картографування для відпочинку. діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rollerblading: Grubic, T., & Vrdoljak, G. (2006). Kinematics of Roller Skating. Facta Universitatis, Series: Physical Education and Sport, 4(1), 77-83. Discusses the kinematics of roller skating, providing insights into the mechanics of rollerblading.
2. Структура і правила оформлення. ДСТУ 3008-95
3. GIS Fundamentals: A First Text on Geographic Information Systems" by Paul Bolstad – 250 с
4. Anonymous users' article about Apache String Utils class [Electronic resource] – Mode of access: https://javarush.ru/groups/posts/3187-razbiraem-po-polochkam-klass-stringutils
5. Making Maps: A Visual Guide to Map Design for GIS" by John Krygier and Denis Wood – 540 с
6. Lee, K. J. (2003). An Introduction to Inline Skating. Human Kinetics. Provides a comprehensive introduction to inline skating, which includes rollerblading, covering various aspects such as techniques, safety, and equipment.
7. І. Н. Блінов, В. С. Романчик – Методи програмування Java – 740 с
8. Jerome Jaglale – Spring CookBook – 540 с
9. Boyarsky Selikoff – Oracle Certified Java – 350 с
10. Vlad Mihalcea – High-Performance Java Persistence – 380 с
11. The Nature of Maps: Essays Toward Understanding Maps and Mapping" by Arthur H. Robinson – 450 с
12. Paul Fisher, Brian D. Murphy – Spring Persistence with Hibernate – 179 с
13. Information about Java Runtime Environment [Electronic resource] – Mode of access: https://uk.wikipedia.org/wiki /Java_Runtime_Environment
14. Програмні засоби ЭВМ. Показники і методи оцінки якості. ДСТУ 2850-94

15. Герберт Шилдт – Java 21. Повне керівництво – 425 с
16. Давыдов С.В. *Intellij IDEA ultimate IDEA ultimate*. Профессиональное программирование на *Java*. – БХВ-Петербург, 2005. – 800 с
17. Схеми алгоритмов, даних й систем. ГОСТ 19.701-90
18. Положення про дипломні роботи (проекти) випускників Національного Авіаційного Університету Київ 2006. – 72 с
19. Oracle – Glance at Java EE enterprise technology [Electronic resource] – Mode of access: https://www.oracle.com/cis/java/technologies/java-ee-glance.html
20. Article about microservices by fellow Web Developer [Electronic resource] – Mode of access: https://articles.microservices.com/monolithic-vs-microservices
21. Clear, J. (2018). Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones. Avery. Although not directly about rollerblading, it offers insights into habit formation and how it can be applied to improve rollerblading practices.
22. Державний стандарт України. Документація, звіти в сфері науки і техніки.
23. Джошуа Блох – Java Ефективне програмування – 460 с
24. Spring Boot develiper full course [Electronic resource] – Mode of access: https://www.educative.io/SpringDeveloper
25. Spring Framework – Brief understanding [Electronic resource] – Mode of access: https://spring.io/projects/spring-boot
26. Interactive Data Visualization for the Web: An Introduction to Designing with D3. O'Reilly Media, Inc. Focuses on D3.js, a popular JavaScript library for creating interactive data visualizations, including maps. – 350 с