МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук фізики та математики
Кафедра комп'ютерних наук та програмної інженерії

ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-СЕРВІСУ ДЛЯ ЗАНЯТТЯ СПОРТОМ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу 12-441 групи Спеціальності 121 Інженерія програмного забезпечення Освітньо-професійної програми «Інженерія програмного забезпечення»

Войнов Владислав Сергійович

Наукові керівники: Полторацький М.Ю. доктор філософії зі спеціальності 121 Інженерія програмного забезпечення, доцент Співаковський О.В. доктор педагогічних наук, професор

Рецензент: Жиряков Олександр, Fullstack Developer, EzCloud

Херсон — 2024

ЗМІСТ

ВСТУП

У сучасному суспільстві, що визначається стрімким ритмом життя та постійним стресом, питання здоров'я та фізичної активності набувають все більшого значення. В умовах цього вимушеного акценту на здоров'ї, використання інноваційних технологій для підтримки фітнесу та спорту стає важливою складовою здорового способу життя. У цьому контексті, спрямована на підтримку та удосконалення тренувань, ця робота спрямована на створення веб-додатку, що слугуватиме в якості щоденника для тренувань.

Завдяки стрімкому розвитку інформаційних технологій та їхньому широкому впровадженню у спорт, актуальність веб-додатків в якості інструмента для ефективного тренувального процесу стрімко зростає.

Розробка інтерактивного щоденника спортивних тренувань має на меті забезпечити спортсменів зручним та інтуїтивним інструментом для ведення детального обліку тренувань, аналізу досягнень та планування майбутніх зусиль.

Предметом дослідження є ефективність використання веб-додатків як інструменту для підтримки та удосконалення тренувального процесу.

Об’єкт дослідження: веб-додатки.

Основні цілі роботи включають:

- Формулювання функціональних та нефункціональних вимог до готового продукту
- Проектування архітектури веб-додатку
- Розробка веб-додатку
- Створення документації

РОЗДІЛ 1. Вимоги до веб-додатку

1.1 Способи складання вимог

На сьогодні існує багато різних форматів вимог. Це можна робити за допомогою Use Case діаграм, User Stories, Volere Requirements Specification Templates тощо. В нашому випадку ми розглянемо фреймворк для опису вимог Behavior Driven Development. Або якщо точніше, то мову специфікацій, яка в ньому використовується - Gherkin.

Gherkin - це мова специфікацій, яка використовується в Behaviour Driven Development[1]. Gherkin прагне забезпечити дотримання чітких, однозначних вимог, починаючи з початкових етапів визначення вимог керівництвом бізнесу та на інших етапах життєвого циклу розробки.

Behaviour Driven Development - це підхід до розробки програмного забезпечення, який зосереджений на спільному розумінні між розробниками, тестувальниками та інвесторами за допомогою створення специфікацій в читабельній формі. Gherkin служить мовним посередником між усіма учасниками процесу.

Опис специфікації дуже простий, все починається з ключового слова “Функціонал”. Тут описується основний функціонал. Далі йде “Приклад”, або як його ще називають “Сценарій”. Тут вже описується конкретний випадок за допомогою слів “Нехай”, “Коли”, “Якщо”, “Тоді”, “І”, “Але” тощо. Таким чином ми отримуємо просту структуру для вимоги з конкретними сценаріями і очікуваною поведінкою програми. Також варто зазначити что синтаксис Gherkin віддалено нагадує мову програмування Python та мову розмітки YAML.

1.2 Функціональні вимоги

Функціональні вимоги — це вимоги до програмного забезпечення, які описують внутрішню роботу системи та її поведінку. Наприклад, процес обробки даних, алгоритми виконання коду та інші специфічні функції.

Для опису функціональних вимог ми використаємо мову Gherkin. Почнемо з глобальної поведінки.

Функціонал: Початок тренування

Передумова: Користувач авторизований в системі

Сценарій: Користувач заходить на сторінку з тренувальними майданчиками

Тоді вмикає фільтр “Спочатку найближчі”

Тоді обирає потрібний йому майданчик

Тоді відмічає кількість підтягувань

Тоді система зберігає введені дані

Сценарій: Користувач заходить на сторінку з тренувальними майданчиками

Тоді обирає фільтр “тільки обрані”

І бачить тільки обрані тренувальні майданчики

Опишемо ще декілька сценаріїв взаємодії з нашим веб-сервісом.

Функціонал: Статистика

Передумова: Користувач авторизований в системі

Сценарій: Користувач заходить в свій профіль

І бачить загальну кількість підтягувань

Та бачить максимальну кількість підтягувань

Також кількість відвіданих перекладин за весь час

Сценарій: Користувач заходить в свій профіль

І натискає кнопку “Візуалізація”

І бачить графіки з своїм прогресом

Функціонал: Реєстрація

Передумова: Користувач не авторизований

Сценарій: Користувач заходить на сторінку реєстрації

І вводить коректний емейл

І свій пароль

Коли він відправляє форму

То отримує повідомлення про успішну реєстрацію

І перенаправляється до авторизації

Функціонал: Авторизація

Передумова: Користувач зареєстрований, але не авторизований

Сценарій: Користувач заходить на сторінку авторизації

І вводить коректний емейл

І вводить свій пароль

Коли він відправляє форму

То перенаправляється на головну сторінку

Але вже як авторизований користувач

Сценарій: Користувач заходить на сторінку авторизації

І вводить коректний емейл

І вводить неправильний пароль

Коли він відправляє форму

То отримує повідомлення про неправильні дані для входу

І залишається на сторінці авторизації

Функціонал: Обрані перекладини

Передумова: Користувач авторизований

Сценарій: Користувач знаходиться на сторінці з перекладинами

І обирає “Додати в обрані” на одній з перекладин

Тоді вона змінює колір

І користувач отримує повідомлення про успішність дії

Сценарій: Користувач знаходиться на сторінці обраних перекладин

І обирає “Видалити з обраного” на обраній перекладині

Тоді вона зникає з сторінки обраних перекладин

І користувач отримує повідомлення про успішність дії

Таким чином ми маємо за основу декілька прикладів поведінки нашого сервісу, які ми маємо задовольнити під час розробки.

1.3 Нефункціональні вимоги

Нефункціональні вимоги – це вимоги, які визначають якою система повинна бути, при цьому ніяк не впливаючи на те, що система повинна робити.

Нефункціональні вимоги можна поділити на дві категорії[2]:

- покращення системи
- вдосконалення властивостей системи

Визначимо вимоги категорії покращення до нашої системи:

- безпека
- надійність та доступність
- обробка та стійкість до помилок
- збереження даних
- валідація введених даних

Тепер визначимо вимоги для вдосконалення системи:

- можливість масштабування
- відновлюваність

Тепер наведемо по одному прикладу до кожної з вимог в таблиці 1.1 Приклади вимог.

Таблиця 1.1 Приклади вимог

Вимога	Приклад
Безпека	Зберігати пароль користувача в вигляду хешу
Надійність та доступність	Веб-сервіс доступний для використання майже 100% часу
Обробка на стійкість до помилок	При отриманні помилок система продовжує свою роботу, при цьому повідомляє користувачу що він зробив щось неправильно
Збереження даних	Дані зберігаються після перезапуску системи
Валідація введених даних	Користувач не може відправити порожню форму
Можливість масштабування	Підтримка як горизонтального так і вертикального принципів масштабування
Відновлюваність	В разі збою системи та втраті даних мати можливість їх відновити

1.4 Оцінка важливості вимог методом Кано

Метод Кано - це модель, яка була розроблена японським інженером та професором Норіакі Кано. Цей метод використовується для визначення та розуміння очікувань клієнтів від продукту чи послуги. Модель Кано визначає різні види атрибутів продукту і класифікує їх відносно того, як вони впливають на задоволення користувача.

Основні елементи методу Кано включають:

1. Базові атрибути - основні функції або характеристики, які користувачі вважають очевидними. Їх відсутність може призвести до невдоволення, але їх наявність не спричиняє особливого задоволення.
2. Лінійні атрибути - це атрибути, які прямо пропорційні рівню задоволення. Чим більше цих атрибутів, тим більше задоволення користувача.
3. Зворотні атрибути - це неочікувані або додаткові функції, які можуть суттєво підняти рівень задоволення, але їх відсутність не викликає невдоволення.

Створимо таблицю з функціями використовуючи метод Кано.

Таблиця 1.2 Розподіл Кано

№	Функціонал	Атрибут
1	Автентифікація	Базовий
2	Адаптивний дизайн	Лінійний
3	Карта з тренувальними майданчиками	Лінійний

Продовження таблиці 1.2

4	Обрані тренувальні майданчики	Лінійний
5	Досягнення	Зворотній
6	Трекінг прогресу	Базовий
7	Центр нотифікацій	Зворотній
8	Статистика користувача	Зворотній
9	Візуалізація статистики	Зворотній
10	Експорт даних	Зворотній
11	Відстань до тренувальних майданчиків	Лінійний
12	Прокласти маршрут на карті	Лінійний
13	Сортування по відстані від користувача	Лінійний

РОЗДІЛ 2. Проектування архітектури веб-додатку

2.1 Вибір мов програмування

Розробка веб застосунків зазвичай складається в двох частин - бекенду та фронтенду. Вибір мови програмування для фронтенду зазвичай дуже простий - це JavaScript.

JavaScript з'явилася в 1995 році і була розроблена Бренданом Ейком під час його роботи в компанії Netscape. На відміну від Java, це була скриптова мова, яка використовувалася для реалізації динамічної взаємодії на веб-сторінках.

У 1997 році JavaScript була передана в Ecma International для стандартизації, і тепер мова має офіційну назву ECMAScript. Специфікації ECMAScript визначають основні характеристики мови, і різні реалізації, такі як JavaScript у браузерах, повинні відповідати цим стандартам.

Сьогодні JavaScript є ключовою мовою для веб-розробки. Вона використовується для створення різноманітних веб-додатків, від простих веб-сайтів до складних односторінкових застосунків. Розвиток фреймворків та бібліотек робить JavaScript все більш потужною та універсальною мовою програмування.

Для бекенду ми маємо більше варіантів для вибору, серед яких:

- Java
- Python
- Ruby
- JavaScript

- C#
- PHP
- Go

На нашу думку вибір мови програмування більше відноситься до власних уподобань, адже ви можете зробити одне й ту саму програму використовуючи різні мови програмування. Різниця буде лише в синтаксисі та способах досягнення мети. Також не забуваємо про інструменти, тобто бібліотеки та фреймворки, що полегрують розробку в рази. Вони орієнтовані а особливості мови програмування для якої вони написані.

Ми обрали Python як мову програмування для бекенду. Серед його переваг можна виділити:

- Простий синтаксис - щоб легше розбиратися в існуючому коді
- Об'єктно-орієнтованість - допоможе з рівнем абстракції та перевикористані вже написаного коду
- Широкий вибір бібліотек - також допоможе досягати більшого при цьому писати менше коду

Сам Python є високорівневою інтерпретованою мовою програмування, яка виникла завдяки роботі Гвідо ван Россума в 1991 році. Назва мови була вибрана на честь популярного телешоу "Монті Пайтоновський летючий цирк", улюбленого Гвідо.

Стандартизація Python відбулася у 2000 році, коли вперше був випущений стандартний процес розробки (PEP - Python Enhancement Proposal). У 2008 році вийшла версія 3.0, також відома як "Python 3000" або "Py3k", яка внесла суттєві зміни у мову, спрямовані на поліпшення читабельності коду та інших аспектів.

Протягом 2010-х років Python набув великої популярності, стаючи однією з найбільш використовуваних мов програмування. Це було визнано завдяки великій спільноті розробників, а також його застосуванню в різних галузях, включаючи веб-розробку, аналіз даних та машинне навчання.

Активний розвиток мови проявляється у випуску нових версій з новими можливостями та удосконаленнями. Введення асинхронного програмування, поліпшення системи управління пакетами і інші інновації підтверджують важливе положення Python у світі програмування.

Python продовжує залишатися популярною мовою програмування, завдяки своїй простоті, ефективності та великій гнучкості, що робить її привабливою для широкого кола розробників та проектів.

2.2 Вибір веб-фреймворків

Веб-фреймворк - це набір програмних інструментів, бібліотек та стандартів, які допомагають розробникам створювати та розгортати веб-додатки. Фреймворки надають структуру та готові компоненти для полегшення роботи з веб-технологіями, такими як HTTP-протокол, маршрутизація, шаблони, бази даних, аутентифікація та інші.

Оберемо спочатку фреймворк для бекенд частини. А саме порівняємо Django та Flask.

Django - це високорівневий веб-фреймворк для розробки веб-додатків на мові програмування Python[6]. Він надає багато готових інструментів і можливостей для швидкої розробки веб-застосунків.

Плюси Django:

1. Швидка розробка - забезпечує високий рівень абстракції що дозволяє швидко створювати функціональні веб-додатки.
2. Вбудовані інструменти - має вбудовані інструменти для роботи з базами даних, формами, аутентифікацією, адміністративним інтерфейсом і багато іншого.
3. Масштабованість - здатний обробляти великі обсяги трафіку і масштабується з ростом проекту.
4. Безпека - включає захист від багатьох атак, таких як CSRF, SQL-ін'єкції, XSS тощо.
5. Спільнота та документація - активна спільнота і добре документований код.
6. Адміністративний інтерфейс - Доступний готовий адміністративний інтерфейс для управління даними в базі даних, що дозволяє швидко наповнити базу даних для тестування і навіть адміністрування.

Серед мінусів варто виділити велику вимогу до ресурсів - для невеликих проектів Django може здатися занадто великим, що може вплинути на швидкодію.

Flask - це легкий веб-фреймворк для розробки веб-додатків на мові програмування Python[5]. Його основною метою є забезпечення простоти і гнучкості. Розглянемо його плюси:

1. Простота - Flask має простий і зрозумілий синтаксис, що полегшує навчання і використання для початківців.

2. Гнучкість - Flask не вводить жорстких структур або вимог, що дає розробникам більше свободи в обранні архітектури проекту.

3. Легкий - Flask - це мінімалістичний фреймворк, що вимагає менше коду порівняно з іншими фреймворками.

4. Розширюваність - Flask має багато розширень, які дозволяють додавати функціонал без необхідності включення всіх можливостей фреймворку.

5. Спільнота і документація - має активну спільноту та добре документований код.

Не забуваємо про мінуси:

1. Менше готових інструментів - Порівняно з Django, Flask надає менше готових компонентів, таких як адміністративний інтерфейс або система аутентифікації.

2. Не підходить для великих проектів - Для деяких великих або складних проектів Flask може вимагати більше ручного конфігурування, що впливає на швидкість розробки.

3. Відсутність жорсткої структури - Відсутність жорсткої структури може призвести до того, що розробники використовують різні підходи до організації коду, що може ускладнити розуміння для інших розробників.

Розглянувши плюси і мінуси обох фреймворків було прийнято рішення використовувати Django. Тому що проект у нас доволі масштабний, що дозволить нам розкрити більшу частину його потенціалу.

Щодо вибору фронтенд фреймворка, то на нашу думку тут ситуація дуже схожа на вибір мови програмування. Бо популярні JavaScript

фреймворки дозволяють розбити код на реюзабельні компоненти, просто роюлять це кожен своїм шляхом. Обираючи між Vue, React та Angular ми зупинилися на Vue через його легкість вивчення та освоєння.

2.3 Структура бази даних для зберігання інформації

Спланувати архітектуру бази даних - не проста задача, яка має дуже високий коефіцієнт корисності та повністю виправдовує витрачений час. Звичайно, відразу ідеально створити архітектуру вкрай складно, проте мати чудовий макет для майбутній вдосконалень - не захмарна мрія.

Почнемо нашу структуру з основного - користувача.

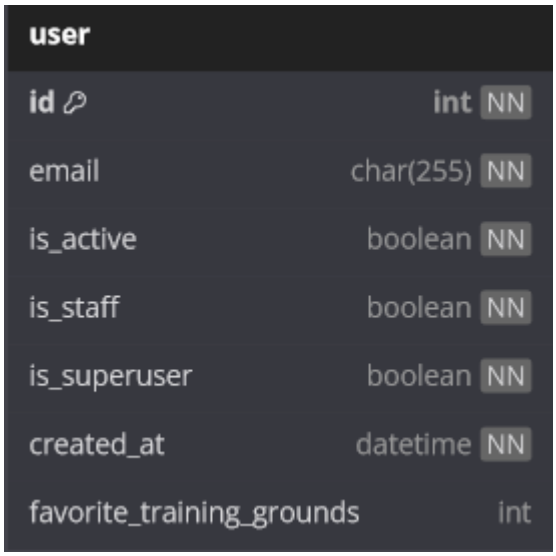


Рисунок 2.1 Таблиця користувача

Проста модель користувача, серед незвичайного тут лише favorite_training_grounds, що слугують для збереження обраних майданчиків. Для цього використовується відношення багато до багатьох.

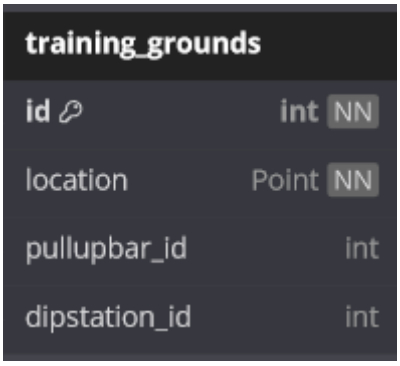


Рисунок 2.2 Модель тренувального майданчика

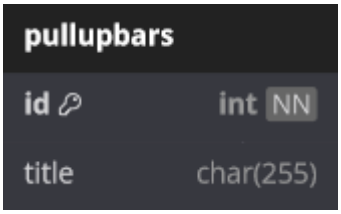


Рисунок 2.3 Модель турніка

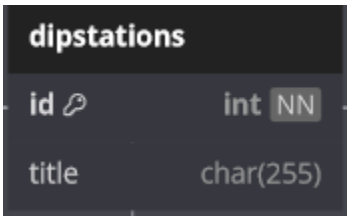


Рисунок 2.4 Модель брусів

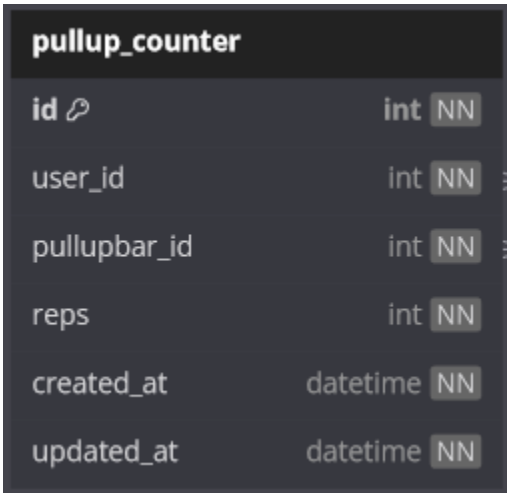


Рисунок 2.5 Модель лічильника для підтягувань

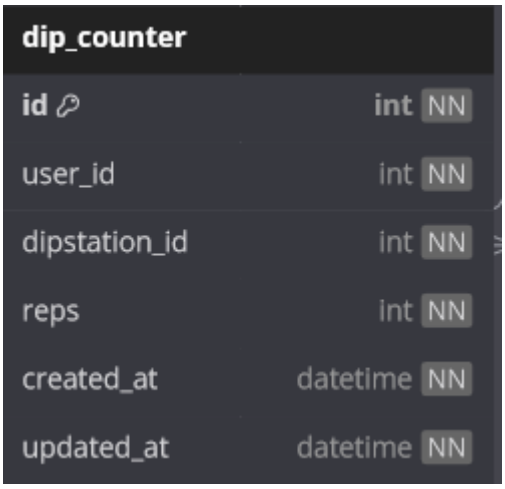


Рисунок 2.6 Модель лічильника для віджимань на брусах

Лічильник використовується для збереження кількості робочих повторень, зроблених певним користувачем на певному спортивному снаряді. А маючи поле created_at ми в майбутньому зможемо виконати аналіз прогресу користувача за певний проміжок часу.

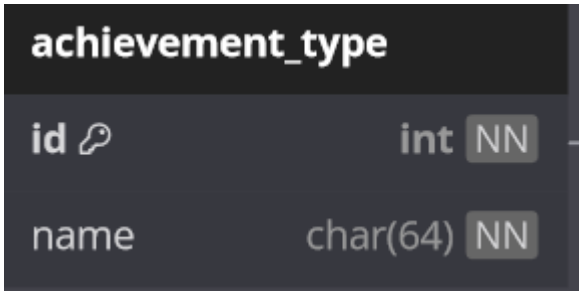


Рисунок 2.7 Модель типу досягнення

Рисунок 2.8 Модель зображення досягнення

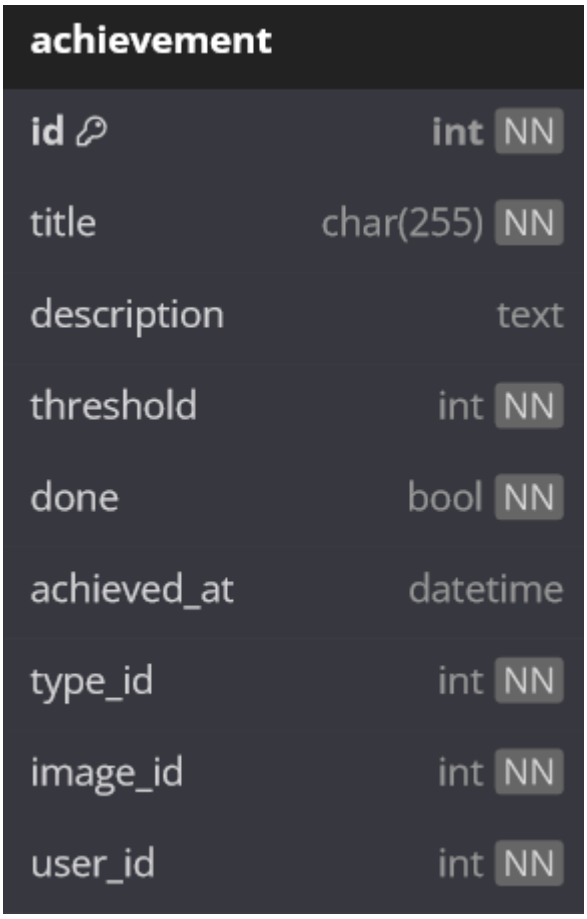


Рисунок 2.9 Модель досягнення

На рисунках 2.7, 2.8, 2.9 можна побачити реалізацію системи досягнень. Слід зазначити що досягнення розділені за типами та бар'єром отримання. Таким чином ми можемо мати досягнення по типу “Підтягнутися 100 разів загалом” і “Підтягнутися 1000 разів загалом”. Обидва досягнення матимуть один тип, щось накштал “підтягування загалом”, при цьому матимуть різні пороги отримання в 100 та 1000 відповідно. Це полегшить нам процесс розблокування досягнень в рази. Це логічно зробити, адже алгоритм отримання один і той самий, відрізняється лише межа отримання.

Також є окрема таблиця для зображення. Вона існує для того щоб оптимізувати збереження зображень і полегшити нам роботу. Так як досягнення у кожного користувача свої, і мають свій прогрес, то дублювати ще й зображення не звучить як гарна ідея. Тому знаючи тип і поріг відкриття досягнення, можна легко приєднати до нього потрібне зображення.

І маємо саму таблицю досягнення, в якій ми “збираємо докупи” тип, зображення та інші необхідні поля.

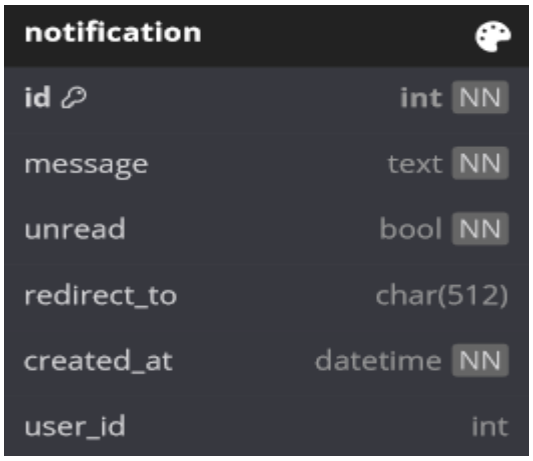


Рисунок 2.10 Модель нотифікацій

Проста таблиця для нотифікацій. Вони унікальні для кожного користувача, та можуть містити в собі посилання на якусь сторінку сайту. В основному використовуватимуться для повідомлення користувача про нове досягнення. В цьому випадку посилання буде на сторінку з досягненнями.

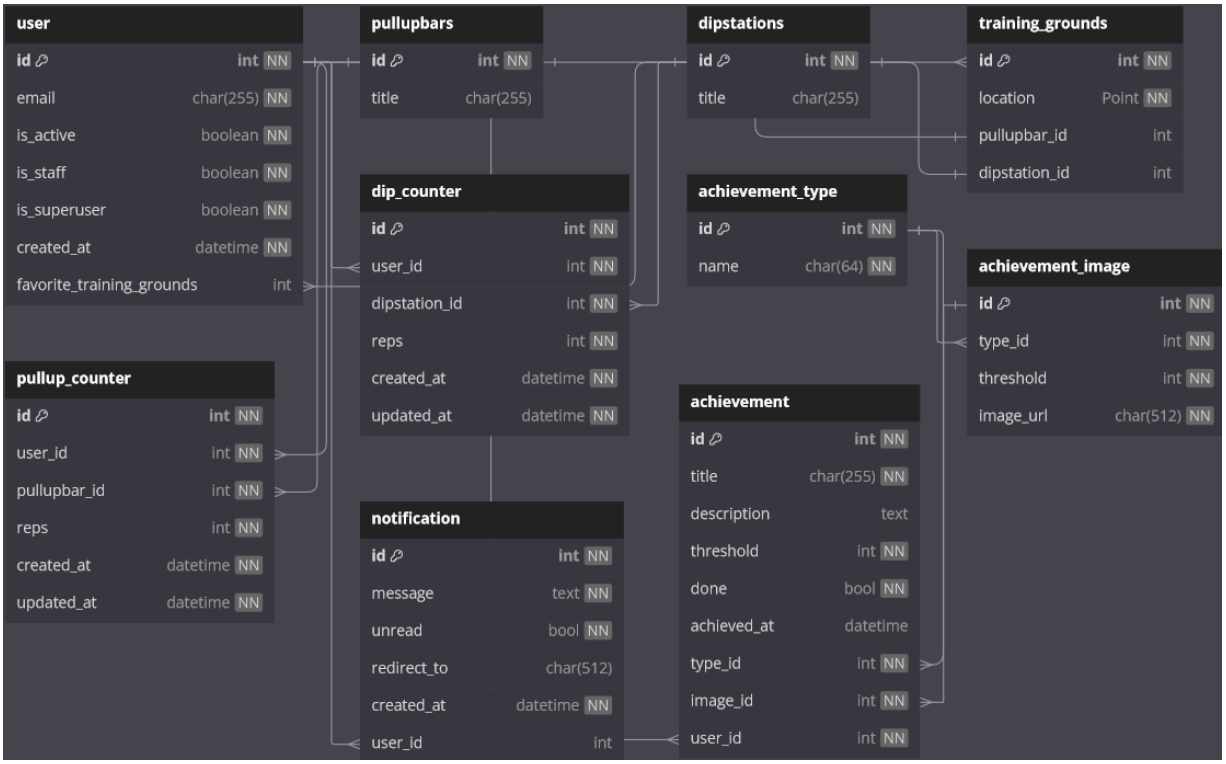


Рисунок 2.11 Повна схема таблиць бази даних

2.4 UML діаграми

UML – що перекладається як уніфікована мова моделювання (англ. Unified Modeling Language) що широко використовується у парадигмі об'єктно орієнтованого програмування. Сама мова була створена для визначення, проектування, візуалізації та документації програмних систем.

Використаємо діаграму прецедентів (Use Case Diagram) для опису поведінки та користувачів нашого продукту.

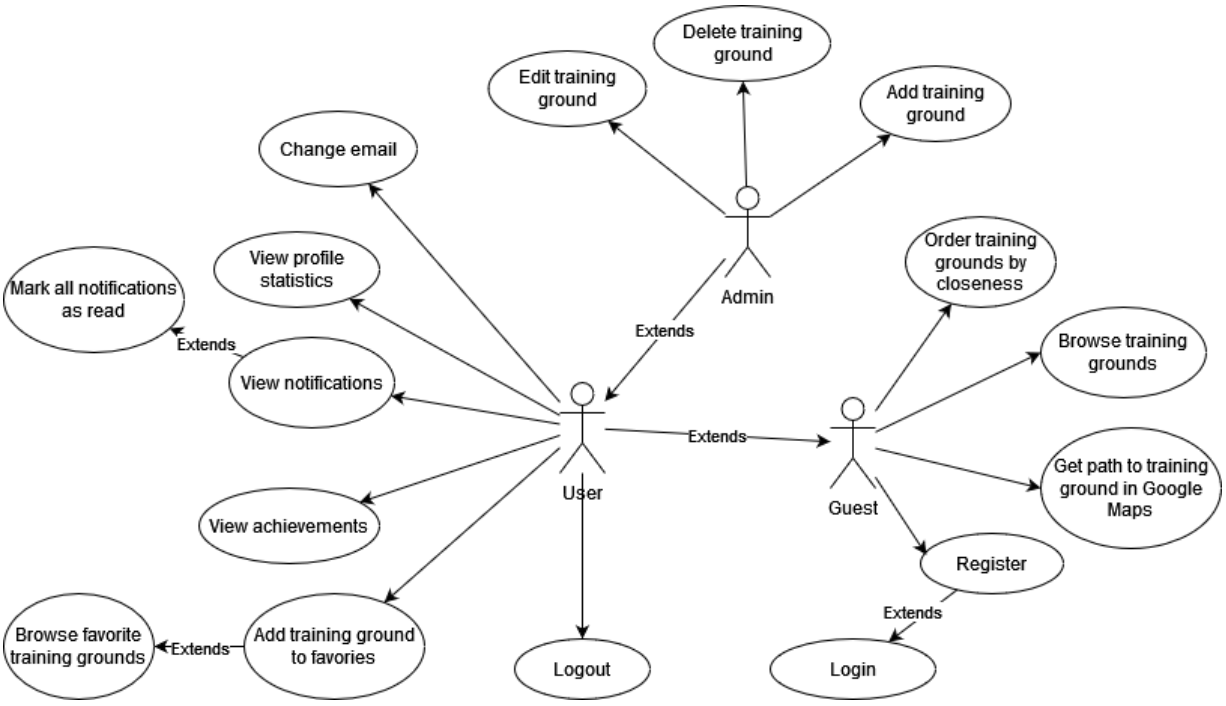


Рисунок 2.12 Діаграма прецедентів

Також варто описати приклади сутностей в базі даних. Якраз для такий випадків існує діаграма об’єктів (Object Diagram).

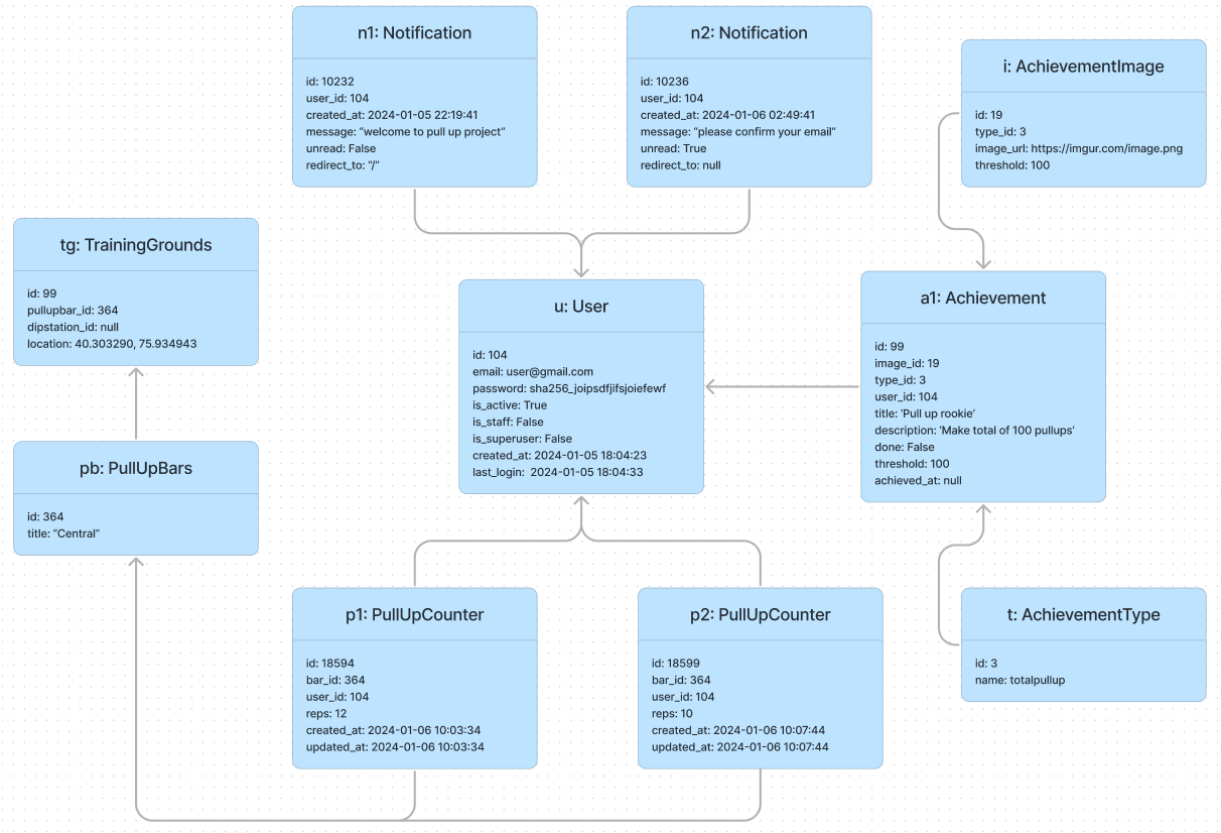


Рисунок 2.13 Діаграма об'єктів

РОЗДІЛ 3. Розробка веб-додатку

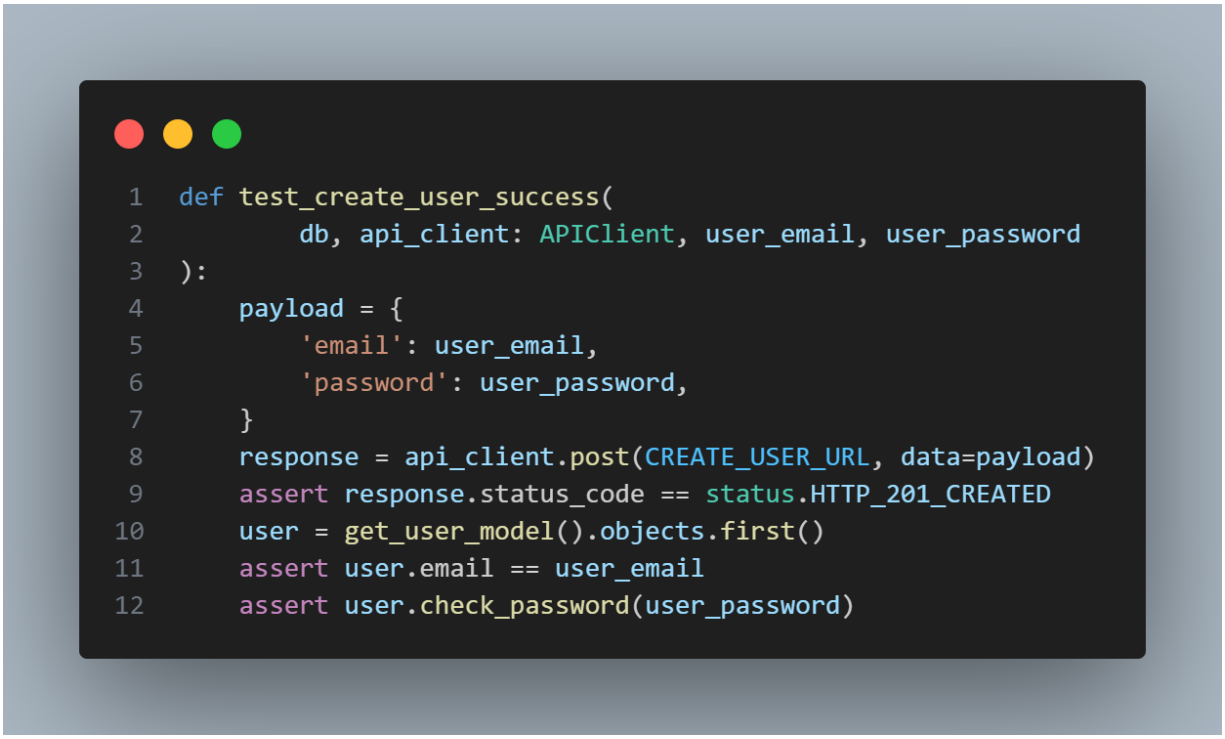
3.1 Тестування описаних вимог

Описавши функціональні вимоги до нашого веб-сервісу ми отримали представлення того, які процеси мають відбуватися. Тепер наша задача полягає в тому, щоб задовольнити всі вимоги протягом циклу розробки. Для цього нам потрібно тестувати нашу програму, для того щоб переконатися що додавання новий функцій не перешкоджає роботі старих. Але робити це вручну досить складно і довго, при цьому не враховуючи людський фактор, який з відносно великим шансом може надати хибний результат.

В таких випадках найкращою практикою буде автоматизувати сам процес тестування. Це дозволить нам мати більше часу на саму розробку, і в той же час вирішить проблему людського фактору.

Для автоматизації тестування існує дуже багато інструментів, але для нашого проекту, на нашу думку, буде достатньо бібліотеки pytest. Фреймворк pytest дозволяє легко писати невеликі, зрозумілі тести та може масштабуватися для підтримки складного функціонального тестування для програм і бібліотек[12].

Почнемо з автоматизації реєстрації. Як правило тести розпочинаються з позитивного кейсу, що в нашому випадку означатиме успішну реєстрацію. Під час тесту, що зображений на рисунку 3.1, ми перевіряємо працездатність API, що відповідає за створення користувачів, відправивши емейл і пароль нового користувача.



```python
def test_create_user_success(
        db, api_client: APIClient, user_email, user_password
):
    payload = {
        'email': user_email,
        'password': user_password,
    }
    response = api_client.post(CREATE_USER_URL, data=payload)
    assert response.status_code == status.HTTP_201_CREATED
    user = get_user_model().objects.first()
    assert user.email == user_email
    assert user.check_password(user_password)
```

Рисунок 3.1 Тест успішної реєстрації

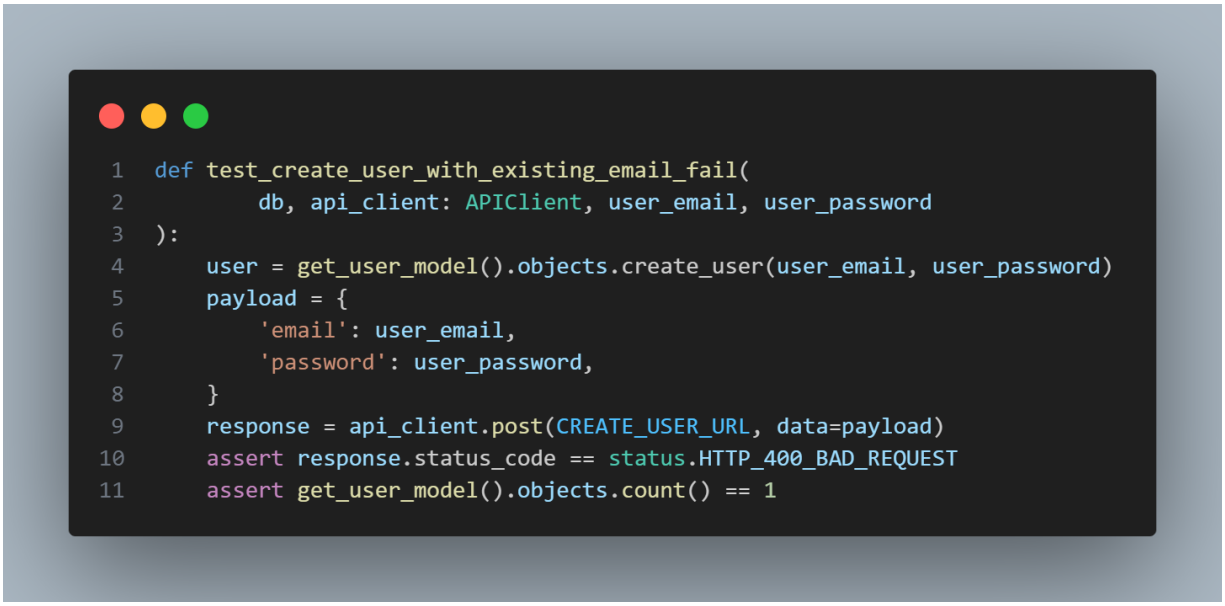
Після тестування успішного кейсу можна переходити на випадки, які передбачають отримання помилок. Зазвичай тут перевіряється валідація та поведінка програми з неправильними вхідними даними. Як приклад можна навести тест-кейс на рисунку 3.2, який перевіряє валідацію електронної пошти користувача.

```python
@pytest.mark.parametrize(
    'email_name,response_code',
    [
        ('usergmail.com', status.HTTP_400_BAD_REQUEST),
        ('user@gmailcom', status.HTTP_400_BAD_REQUEST),
        ('user gmail@gmail.com', status.HTTP_400_BAD_REQUEST),
        ('user@gmail..com', status.HTTP_400_BAD_REQUEST),
        ('user@gmail_com', status.HTTP_400_BAD_REQUEST),
        ('юзер@gmail.com', status.HTTP_400_BAD_REQUEST),
    ],
    ids=[
        'MISSING @ SYMBOL',
        'MISSING DOT IN TOP-LEVEL DOMAIN',
        'SPACE USAGE',
        'INVALID DOMAIN FORMAT',
        'USE OF FORBIDDEN CHARACTER',
        'WITH CYRILLIC CHARACTERS'
    ]
)
def test_create_user_wrong_emails_fail(
        db,
        api_client: APIClient,
        user_password,
        email_name,
        response_code,
):
    payload = {
        'email': email_name,
        'password': user_password
    }
    response = api_client.post(CREATE_USER_URL, data=payload)
    assert response.status_code == response_code
    assert get_user_model().objects.count() == 0
```

Рисунок 3.2 Тест на валідацію

З позитивний кейсом, а також з валідацією розібралися, тож можна перейти до останнього типу кейсів. В ньому всі дані заповнені правильно,

але, звичайно, є нюанс в самій логікі і поведінці програми. На рисунку 3.3 тест допомагає нам зрозуміти як програма повинна себе поводити при реєстрації користувача з електронною поштою, яка вже була попередньо зареєстрована.



```python
def test_create_user_with_existing_email_fail(
        db, api_client: APIClient, user_email, user_password
):
    user = get_user_model().objects.create_user(user_email, user_password)
    payload = {
        'email': user_email,
        'password': user_password,
    }
    response = api_client.post(CREATE_USER_URL, data=payload)
    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert get_user_model().objects.count() == 1
```

Рисунок 3.3

Таким чином ми перетворили вимоги до реєстрації на юніт тести, які дають змогу протестувати наш програмний засіб в будь-який час, при цьому не потребуючи від нас якихось додатковий дій.

3.2 Порівняння створеної структури бази даних із запланованою

Під час розробки веб-сервісу для спортивних тренувань для створення схеми бази даних використовувалася раніше створена схема на рисунку 2.8. Для взаємодії з системою управління базами даних ми використали моделі Django, які дозволили нам отримати високорівневий інтерфейс, також відомий як ORM (object-relational mapping).

Для перевірки того, як ми справилися з відтворенням схеми бази даних, ми можемо візуалізувати все наявну схему за допомогою бібліотеки django-extensions[9]. Якщо точніше, то дана бібліотека регенує графове представлення нашої схеми бази даних використовуючи DOT language[10].

Маючи DOT файл, ми можемо візуалізувати його за допомогою програмного забезпечення з відкритим кодом Graphviz[11]. Отримані результати представленні на рисунку 3.4.

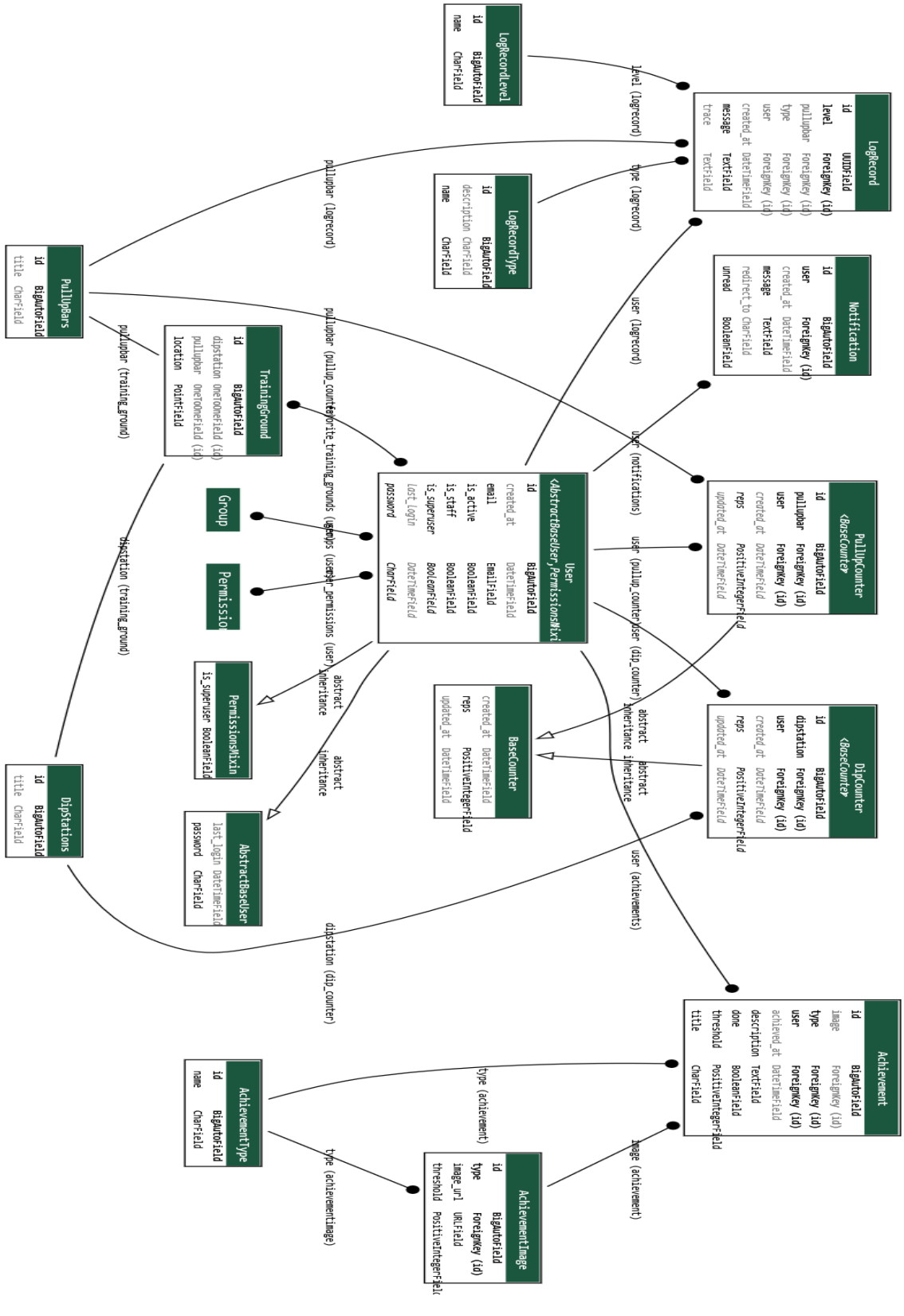


Рисунок 3.4 Діаграма відношення сутностей

Проведемо порівняння сутностей.

Таблиця 2.1 Порівняння таблиць бази даних

Таблиця	Різниця з спроектованою схемою
user	наявне додаткове поле last_login
training grounds	ідентичні
pull up bars	ідентичні
dipstations	ідентичні
pull up counter	ідентичні
dip counter	ідентичні
achievement type	ідентичні
achievement image	ідентичні
achievement	ідентичні
notification	ідентичні

Також варто зазначити, що була створена додаткова модель/таблиця для зберігання логів. Вона не впливає на логіку програми і не була включена в основну схему

3.3 Візуалізація API структури

API – невід'ємна частина будь-якого веб-сервісу, що передбачає в своїй архітектурі окремо бекенд і фронтенд частини. В такому випадку бекенд надає фронтенду зручний інтерфейс для отримання потрібної інформації. Тут і виникає питання: як фронтенд розробникам знати які взагалі ендпоінти має API бекенду?

Для такий випадків існють бібліотеки для створення автоматичної документації і/або тестувальних майданчиків та основі специфікацій OpenAPI, де розробник матиме змогу протестувати API, при цьому не написавши жодного рядка коду. Серед популярних рішень в цьому сегменті є Swagger та Redoc.

Незважаючи на велику інформативність, що надають вищезгадані інструменти, на нашу думку, вони не дозволяють отримати загальне представлення про ендпоінти та їх структуру.

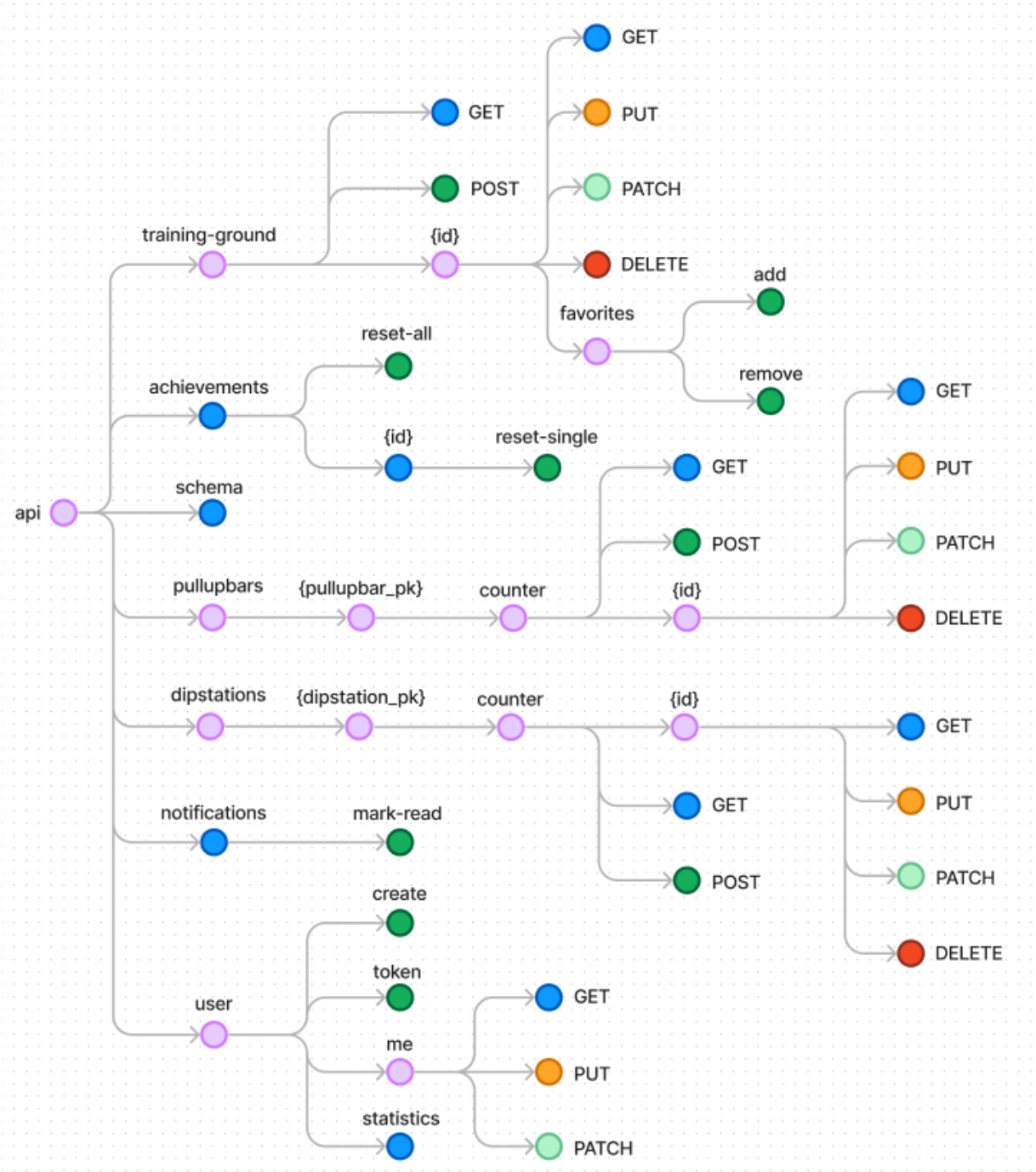


Рисунок 3.5 Загальні структура API

Розглянувши структуру API, зображену на рисунку 3.5, можна зазначити, що API має 4 основних шляхи: notifications, achievements, user, bars, перечислені у відповідності до комплексності від меншої до більшої відповідно.

Також слід виділити, що на схемі присутні ендпоінти з, так званими, “вкладеними маршрутами”. Одним з таких буде шлях /api/pullupbars/{pullupbar_id}/counter/{id}/, який в свою чергу надає можливості для отримання, редагування та видалення зазначеного лічильника на обраній перекладині.

ВИСНОВОК

У даній роботі було розглянуто актуальне питання підтримки та удосконалення тренувального процесу за допомогою веб-додатків у сучасному спортивному середовищі. Зосереджуючись на стрімкому розвитку інформаційних технологій та їхньому впровадженні у сферу спорту, була сформульована мета розробки інтерактивного щоденника спортивних тренувань.

Основними цілями роботи були формулювання функціональних та нефункціональних вимог до готового продукту, визначення ступеня важливості вимог, проектування архітектури веб-додатку, розробка самого додатку та створення документації.

Результатом проведених досліджень є розроблений веб-додаток, який має забезпечити спортсменів зручним та інтуїтивним інструментом для ведення детального обліку тренувань, аналізу досягнень та планування майбутніх зусиль. Впровадження такого інноваційного рішення сприятиме покращенню якості тренувань та досягненню більш високих спортивних результатів у сучасному суспільстві, де здоров'я та фізична активність набувають все більшого значення.

ДЖЕРЕЛА

1. Gherkin Reference - Cucumber Documentation: https://cucumber.io/docs/gherkin/reference/ (Дата звернення 05.12.2023)
2. Секрети Шліфування Якості - http://lvivqaclub.blogspot.com/2008/10/blog-post_17.html (Дата звернення 02.01.2024)
3. Python Official Website: https://www.python.org. (Дата звернення 12.12.2023)
4. JavaScript Documentation: https://developer.mozilla.org/en-US/docs/Web/JavaScript. (Дата звернення 15.12.2023)
5. Flask Documentation:https://flask.palletsprojects.com/en/3.0.x. (Дата звернення 19.12.2023)
6. Django Documentation:https://www.djangoproject.com. (Дата звернення 19.12.2023)
7. Vue Documentation: https://vuejs.org. (Дата звернення 21.12.2023)
8. Database diagram tool: https://dbdiagram.io/home.(Дата звернення 22.12.2023)
9. Django-extensions graph models: https://django-extensions.readthedocs.io/en/latest/graph_models.html (Дата звернення: 09.02.2024)
10. DOT language: https://www.graphviz.org/doc/info/lang.html (Дата звернення: 11.02.2024)
11. Graphviz https://www.graphviz.org/ (Дата звернення: 11.02.2024)
12. Pytest Documentation https://docs.pytest.org/en/8.0.x/ (Дата звернення 20.02.2024)