МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

Прикладний програмний інтерфейс управління задачами з використанням RestAPI та Spring Framework

Здобувач освіти гр. ІН-81 **Анна ЛАВРЕНКО**

Науковий керівник, **Олена ПРОЦЕНКО**
доцент кафедри комп'ютерних наук, к.ф.-м.н,
доцент

Завідувач кафедри, **Анатолій ДОВБИШ**
доктор технічний наук.професор

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп’ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи

Здобувача вищої освіти четвертого курсу, групи ІН-81 спеціальності «122 – Комп’ютерні науки» денної форми навчання Лавренко Анни Олександрівни.

Тема: «Прикладний програмний інтерфейс управління задачами з використанням RestAPI та Spring Framework»

Затверджена наказом по СумДУ

№_____ от _____2022 г.

Зміст пояснювальної записки: 1) аналіз предметної області 2) теоретичний огляд архітектури Rest API 3) вивчення особливостей застосування Spring Framework 5) побудова моделі бази даних та програмна реалізація сервісу 6) наведення прикладів роботи прикладного програмного інтерфейсу

Дата видачі завдання “_____” _____2022 р.

Керівник випускної роботи _____ Олена ПРОЦЕНКО

Завдання приняла до виконання _____ Анна ЛАВРЕНКО

РЕФЕРАТ

Записка: 62 стор., 33 рис., 1 табл., 1 додаток, 16 джерел**.**

Об’єкт дослідження — сервіс управління задачами

Мета роботи — розробка прикладного програмного інтерфейсу для організації роботи з задачами за допомогою RestAPI та Spring Framework.

Методи дослідження — метод аналітично-статистичний.

Результати — розроблено прикладний програмний інтерфейс управління задачами з використанням архітектури Rest та Spring Framework. API створено на базі мови програмування Java та фреймворку Spring.

REST API, JAVA, SPRING FRAMEWORK, MY SQL, JIRA

ЗМІСТ

ВСТУП

Дипломна робота присвячена реалізації прикладного програмного інтерфейсу за допомогою Rest та Spring Framework. RestAPI є розповсюдженою архітектурою для проектування прикладних програмних інтерфейсів, вирізняється логічністю структури, простотою реалізації та тестування запитів. SpringFramework вважається зручним інструментом для реалізації API, оскільки представляє комплексну структуру програмування та конфігурації для корпоративних програм на базі мови Java на будь-якій платформі розгортання. Основним елементом Spring є інфраструктурна підтримка на рівні програми, що дає змогу командам розробників більше зосередитись на бізнес-логіці продукту.

Метою кваліфікаційної роботи є реалізація прикладного програмного інтерфейсу управління задачами. Дистанційна робота, онлайн навчання, підвищення вимог до ефективності бізнесу, складність взаємодії з клієнтами змушує шукати нові рішення та застосунки, щоб збільшити продуктивність та покращити організацію роботи працівників. Вважається, що кожній компанії, особливо сфери IT, в епоху діджиталізації та швидкості бізнес-процесів, необхідні системи таск менеджменту. Сучасний ринок пропонує велику кількість додатків, що відрізняються вартістю, функціоналом та мобільністю, але часто компанії бажають мати сервіс, кастомізований під власні потреби. Саме тому, тематику систем управління задачами було обрано в якості предмету досліження та реалізації в рамках кваліфікаційної роботи.

Визначено головні задачі щодо реалізації API, які включено у виконання дипломної роботи:

- огляд аналогів на сучасному ринку, їх особливості та функції;
- теоретичний огляд архітектурного підходу RestAPI:
- проектування бази даних, визначення основних таблиць та зв’язків між ними, розробка програмної частини додатку;
- тестування та наведення прикладів роботи додатку;
- підведення підсумків, щодо подальшої перспективи масштабування та розвитку додатку;

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття систем управління задачами та їх основні функції

Система управління задачами – це програмний засіб для структуризації задач та управління проектами різної складності та масштабів. Застосування подібних додатків для систематизації роботи та комунікації значно пришвидшує вирішення проблем та підвищує рівень задоволенності клієнтів.

Системи управління задачами потрібні для:

- створення загального робочого простору для комунікацій, відслідковування прогресу, постановки задач;
- систематизація задач;
- планування: календарі, канбан/скрам дошки, діаграми Ганта;
- аналіз результатів: створення звітів, облік робочого часу;
- встановлення комунікації між працівниками та з клієнтом;

1.2 Аналоги на сучасному ринку

Сучасний ринок пропонує велику кількість систем управління задачами. Вони відрізняються функціоналом, особливостями інтерфейсу та, звичайно, вартістю.

Топ додатків за популярністю сьогодні:

- JIRA
- TRELLO
- TODOIST

Jira Software – найбільш багатофункціональний додаток, нараховує до 24 мільйонів користувачів, топ в користуванні серед великих компаній, які займаються розробкою та тестуванням програмного забезпечення. Jira Software є багатокомпонентним сервісом, доступна в хмарній та серверій версіях [1].

Розглянемо особливості реалізації та функціонал Jira. Основною одиницею є тікет (Рисунок 1.1). Кожен тікет може мати тип: дефект, тест-кейс, таск і т.д. Користувач може вказати короткий опис тікету, додати зображення

або інші файли, встановити пріоритетність, дедлайн, вказати компоненту програмного продукту та призначити задачу виконавцю.

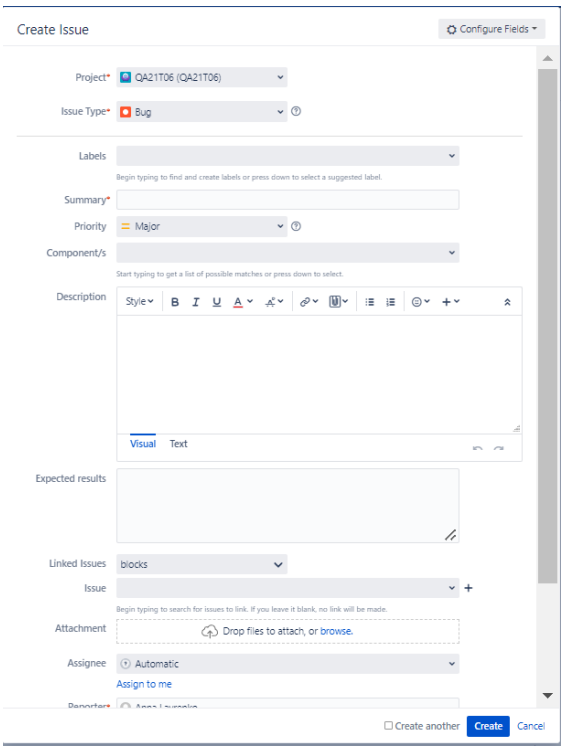

Рисунок 1.1 — Створення тікету в Jira

Створений тікет відображається в зручному форматі. Кожен користувач може редагувати опис, додавати коментарі та змінювати статус. Також є можливість змінювати пріоритетність, додавати лейбли та версію (Рисунок 1.2).

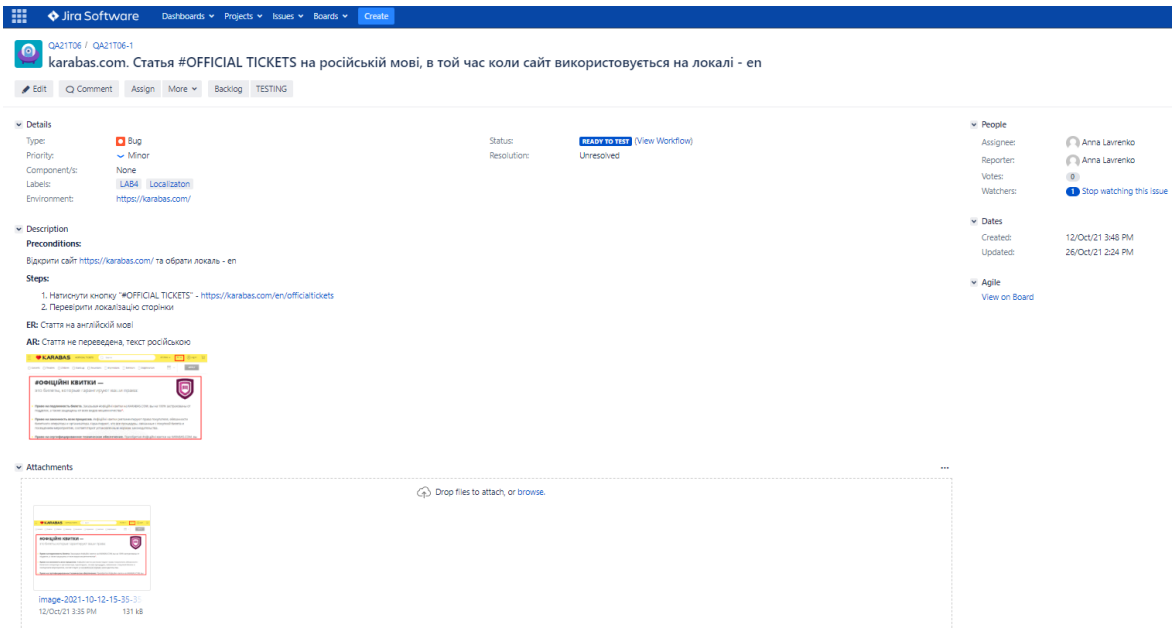

Рисунок 1.2 — Створений тікет Jira

Jira включає функцію фільтрації тікетів за різними критеріями: за статусом, типом, пріоритетністю, виконавцем, ключовими словами та іншими параметрами (Рисунок 1.3).

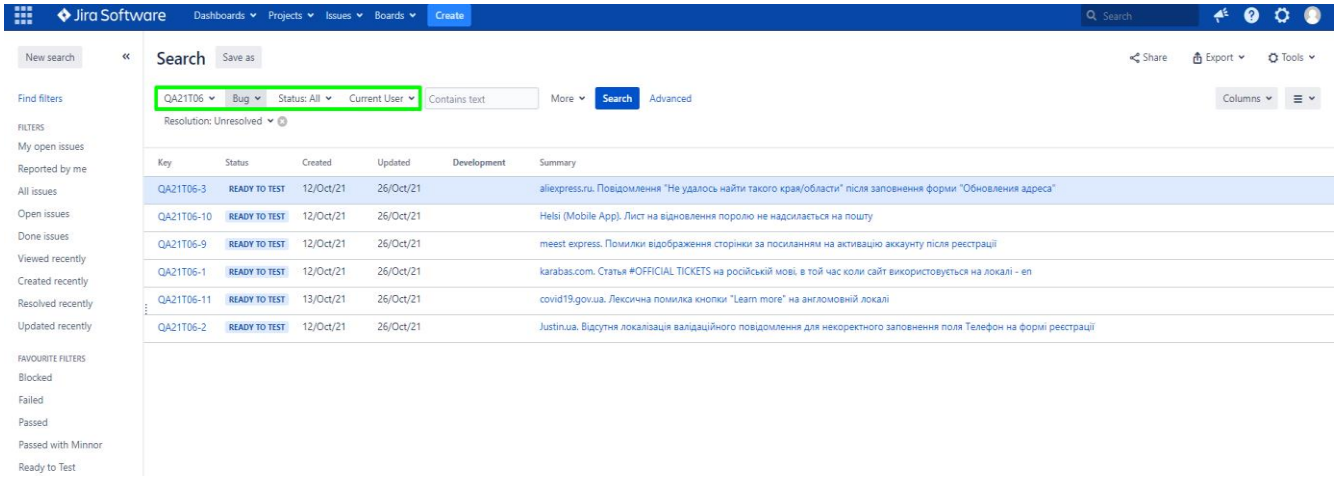

Рисунок 1.3 — Фільтрація

Результат, отриманий в результаті фільтрації, можна завантажити в декількох форматах: HTML, XML, CSV, RSS. Найбільш зручним форматом для експорту даних є CSV (Рисунок 1.4) .

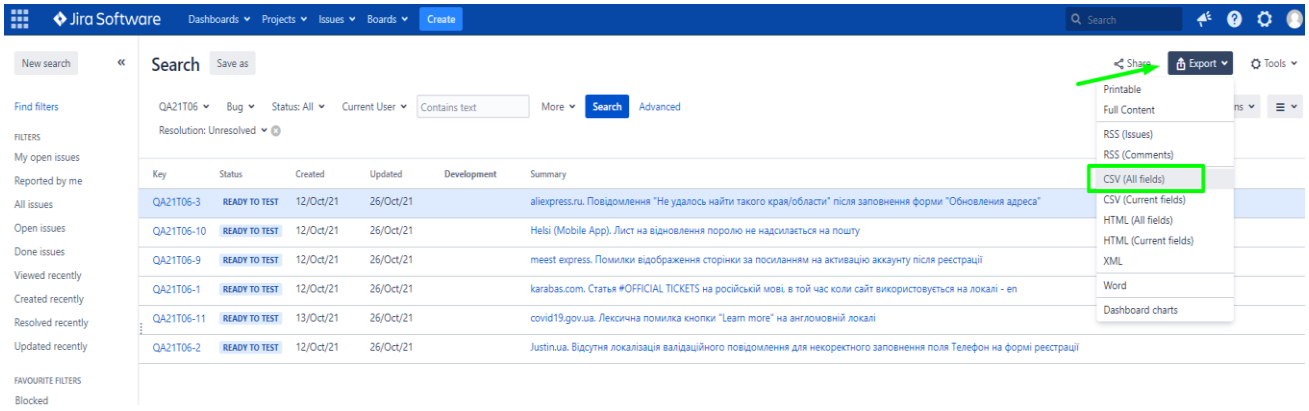

Рисунок 1.4 — Експорт результатів фільтрації

Дашборд – це інформаційна панель, що візуалізує необхідну інформацію (Рисунок 1.5). Jira пропонує широкий спектр можливих форматів, починаючи від списків, статистичних підрахунків, закінчуючи графіками та діаграмами. Користувач має можливість змінювати конфігурацію та розміщувати блоки на дашборді в довільному порядку. Створення таких інформаційних панелей

значно полегшує роботу менеджерів у питаннях відслідковування прогресу, створення звітів та виявлення критичних проблем на проектах.

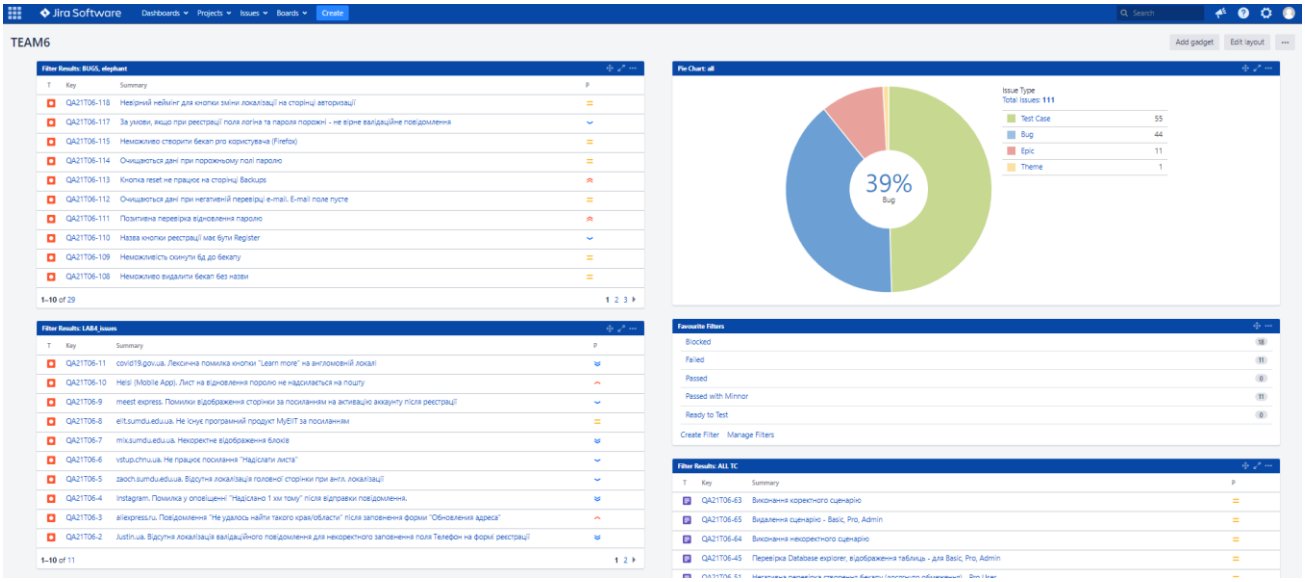

Рисунок 1.5. — Приклад дашборду

Trello – інструмент менеджменту задач. Має більш обмежений функціонал, ніж Jira, але чудово підійде для невеликих команд або особистого користування. Основою додатку є панель, на якій можна створювати списки та додавати картки з задачами (Рисунок 1.6).

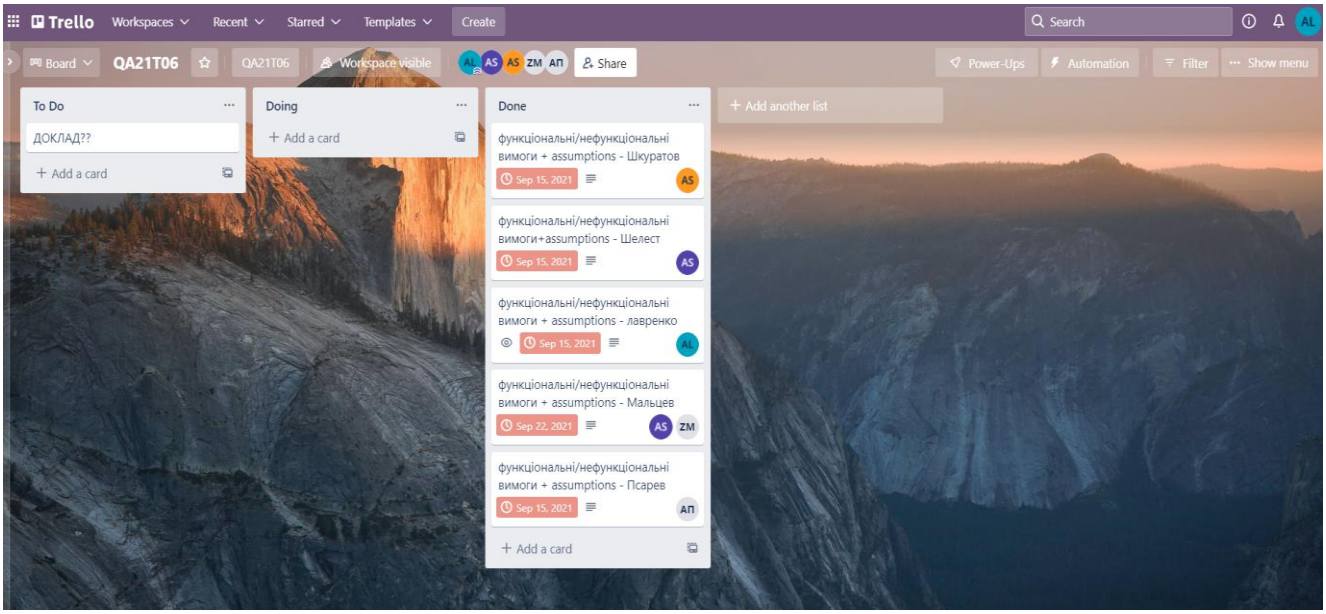

Рисунок 1.6 — Основна сторінка сервісу Trello

Кожну картку можна окремо відкрити та переглянути всю необхідну інформацію: опис задачі, контрольні списки, дати завершення, вкладення та інше (Рисунок 1.7). Користувач може керувати строками, додавати коментарі, надавати ти відстежувати відгуки [2].

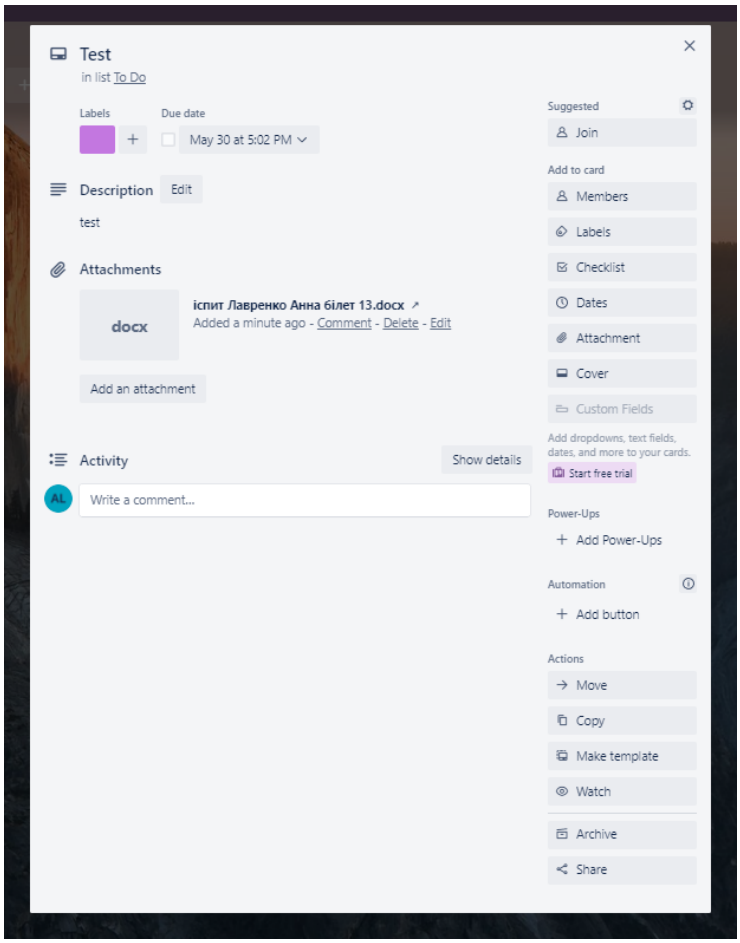

Рисунок 1.7 — Картка Trello

Trello легко інтегрується з іншими інструментами, тому однієї з переваг є те, що інформація про нових учасників, редагування картки, зміни статусу задачі або приближення дедлайну надходить користувачу на поштову скриньку. Trello користується така компанія, як Ebay.

Todoist – таск-менеджер, яким користується 30+ мілліонів людей з 160+ локацій світу. Перевагою Todoist є наявність мобільної версії, що робить коритування більш гнучким та раціональним. Команди Amazon, Disney, NYU користуються даним сервісом [3].

Інтерфейс Todoist має календарну структуру (Рисунок 1.8). Доступний перегляд задач на сьогодня та на найближчі дні. Розробники також пропонують структуризацію задач за допомогою панелей управління та фільтрів. Має зручний інтерфейс для мобільних пристроїв.

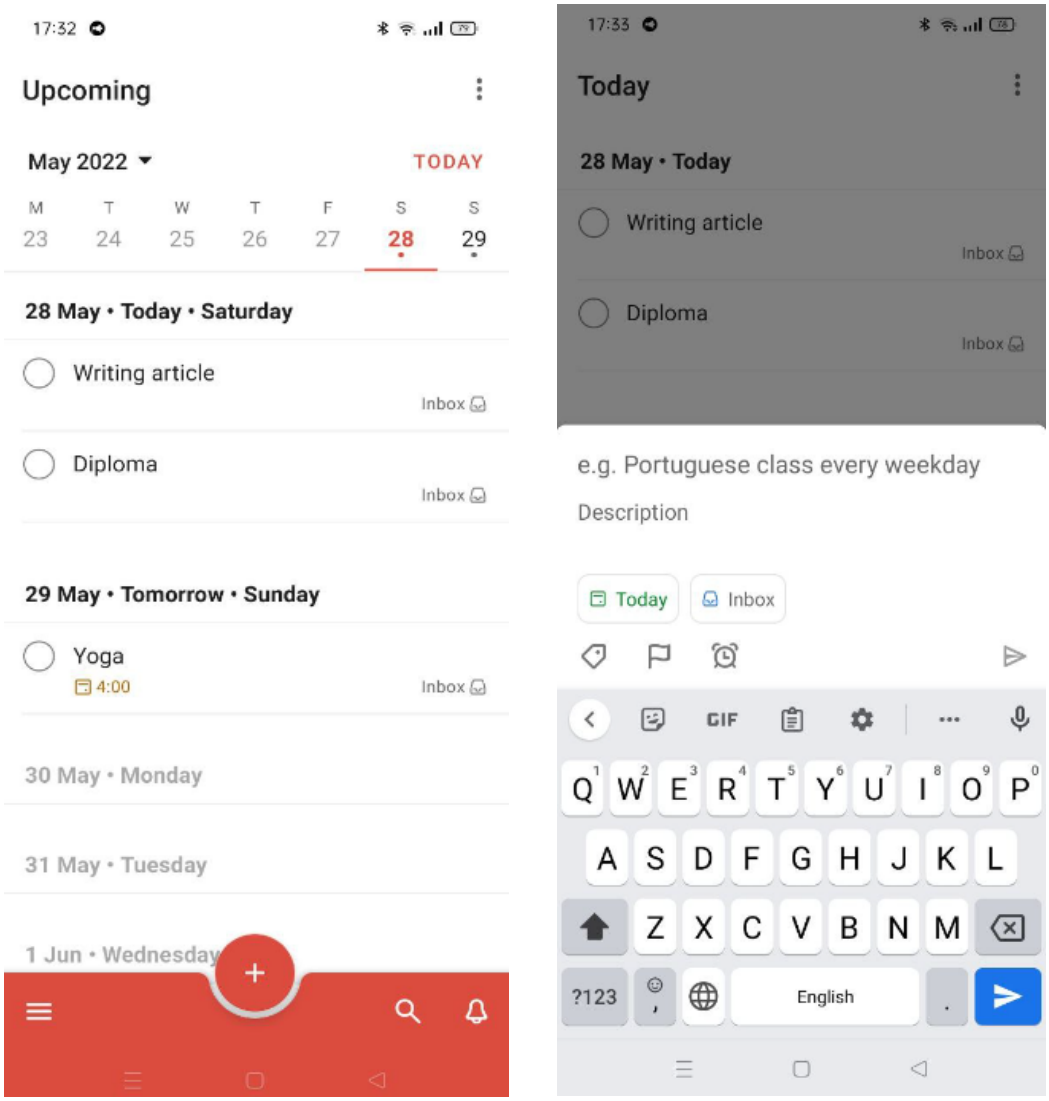

Рисунок 1.8 — Інтерфейс Todoist

Для аналізу було обрано три сервіси, що відрізняються один від одного, але, на практиці, безкоштовні планери можуть легко конкурувати з платними додатками. Наприклад, так Planner від корпорації Microsoft є конкурентом для Trello, оскільки має багато спільного в принципах функціоналу та зовнішнього виду.

Виконаний інформаційний огляд особливостей аналогів буде використано для реалізації дипломної роботи.

1.3 Постановка задачі

Ціль кваліфікаційної роботи бакалавра – реалізація прикладного програмного інтерфейсу управління задачами з використанням Rest API та Spring Framework.

Головні питання для вивчення в рамках роботи:

- теоретинчий огляд архітектурного підходу REST API та його переваги над SOAP;
- ознайомлення з Spring Framework;
- проектування моделі бази даних проекту;
- розробка програмної частини проекту;
- наведення прикладів роботи API;

Виділимо основні функції, які будуть реалізованими REST API TicketManager:

- реєстрація, авторизація, зміна паролю;
- отримання списку усіх користувачів;
- отримання інформації про одного користувача;
- оновлення інформації про користувача;
- отримання списку усіх задач;
- отримання списку задач, які належать користувачу;
- отримання даних про користувача, якому належить задача;
- створення нової задачі;
- зміна задачі (статусу, її опису);
- видалення задачі;
- видалення користувача;

2. ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ API

2.1 Поняття архітектури REST API

API (Application Programming Interface) – прикладний програмний інтерфейс, спосіб взаємодії одної системи з іншою. API є часто основою додатку та з'являється раніше за GUI в життєвому циклі розробки.

REST/SOAP – це дві клієнт-сервер технології, які використовуються для побудови додатків на основі протоколу HTTP. SOAP включає сімейство протоколів та стандартів, що впливає на складність з точки зору машинної обробки, тому його застосовують в більш складних архітектурах. REST – це стиль архітектури програмного забезпечення для побудови розподілених масштабованих Web – додатків. REST вважається швидшим та більш продуктивним, тому може стати ідеальним рішенням для проекту, що не передбачає специфічних маніпуляцій над сутностями [4]. За останні роки REST став переважаючим засобом та витіснює SOAP архітектуру, оскільки є значно простішим у реалізації та використанні. Де-факто REST вже став стандартом для побудови веб-сервісів.

Rest API приймає JSON для реквестів та респосів. Існує інший спосіб передачі даних – XML, але цей формат не так широко підтримується фреймворками.

Архітектура REST включає такі елементи: ресурс, URI, конектори. Ресурс – це все, що розробник хоче відобразити зовнішному світу через його додаток. На прикладі нашого сервісу, екземплярами ресурсів будуть: користувачі та задачі. URI - це універсальний ідентифікатор ресурсу. Найважливішими конекторами є клієнт та сервер. Додатковим типом конектора також може бути кеш [5].

Принципом REST є обмін даними: запит-відповідь. Формат – JSON, а в якості транспорту завжди виступає HTTP. Розлянемо структуру запиту та відповіді, яку визначає HTTP.

Структура запиту:

- рядок запиту (request line) – визначає тип повідомлення

- заголовок запиту (header fields) – характеризує тіло запиту, параметри передачі та іншу інормацію
- тіло повідомлення (body) – необов’язково

Структура відповіді:

- статус (status line)
- поле заголовку відповіді (header fields)
- додаткове тіло повідомлення (body) – необов’язково

Методи HTTP - запиту:

- GET: запит до ресурсу
- POST: створення нового ресурсу
- PUT: оновлення існуючого ресурсу
- DELETE: видалення існуючого ресурсу
- PATCH: часткова зміна ресурсу
- OPTIONS: опис параметрів з’єднання з ресурсом

GET та POST запити є найбільш розповсюдженими. Розглянемо їх особливості та чим відрізнаються (Таблиця 2.1).

Таблиця 2.1 Порівняльна характеристика GET та POST методів

GET	POST
Використовується для відображення інформації, без можливості її змінити.	Використовується для зміни даних, наприклад, оновлення або додавання.
Лише лімітована кількість даних може бути відправлена, оскільки данні відправляються через URL.	Велика кількість даних може бути відправлена, оскільки дані відправляються в тілі HTTP запиту.
Не є безпечним, оскільки дані відображаються в URL.	Є більш безпечним, оскільки дані не відображаються в URL.
Не бажано використовувати для операцій з паролями або іншої чутливої інформації.	Можна використовувати для операцій з паролями та іншою чутливою інформацією.
Сторінка з параметрами може бути кешированa.	Сторінка з параметрами не може бути кешированa.

Як вже було зазначено раніше, складовим елементом відповіді є статус. Розглянемо, яким може бути код відповідей:

- 200 OK: успішне виконання запиту
- 404 Not Found: ресурс не знайдено
- 400 Bad Request: невірний запит
- 201 Created: створено
- 401 Unauthorized: несанкціонований вхід/ помилка авторизації
- 403 Forbidden: користувач не має доступу до ресурсу
- 500 Internal Server error: помилка серверу
- 502 Bad Gateway: невалідний респонс від серверу
- 503 Service Unavailable: помилка на стороні серверу

Теоретично оглянувши архітектуру Rest, можна зробити висновок, що, не дивлячись на простоту та логічність ключових структурних елементів, даний підхід має широкий спектр використання та може ефективно покрити сформовані вимоги кваліфікаційної роботи бакалавра.

2.2 Особливості мови програмування Java

Для реалізації прикладного програмного інтерфейсу було обрано Spring Framework на базі мови програмування Java.

Java – об’єктно-орієнтовна мова програмування, яка була заснована у 1995 році. Сьогодні мовою займається компанія Oracle. Компанія надає віртуальну машину Java та компілятор, який генерує проміжний байткод для JVM (Java Virtual Machine). Головною метою засновників було створення мови, яка не залежить від платформи, може бути застосована для реалізації програмного забезпечення для електронних приладів, засобів зв’язку, веб-сайтів, тощо. Тому, створюючи додатки на Java, розробник може не замислюватися в якій ОС вони працюватимуть [15].

На Java пишуть:

- Мобільні додатки для Android;
- Десктопні додатки;
- Промислові програми;

- Програми для банковських систем;
- Додатки для роботи з Big Data;
- Веб-додатки, веб-сервери;
- Вбудовані системи, наприклад, чіпи;
- Корпоративні додатки будь-якого призначення;

В рейтенгу TIOBE, що є індексом популярності мови, Java зайняла перше місце. За статистикою GitHub – третє. Де-факто Java використовується всюди. Компанії Netflix, Google, Intel, eBay, AliExpress, TripAdvisor та багато інших використовують Java.

Розглянемо основні принципи Java:

1. Незалежність від архітектури.
2. Багатопоточність. Незамінна річ, коли йдеться про створення графічного інтерфейсу. Важливо, щою інтерфейс продовжував відзиватися на будь-які дії користувача в час виконання інших операцій з обробки інформації. Це дає змогу створювати інтерактивні, високоорганізовані та конкурентноздатні додатки.
3. Висока продуктивність. Завдяки використанню інтерпретатором багатьох потоків, виконання байткоду є дуже швидким. Фонові потоки займаються очищенням пам’яті, доки комп’ютер оброляє інші запити.
4. Стійкість до помилок. Java містит автоматичне керування пам’ятю. Вірогідність помилок знижується, оскільки систма автоматично звільняє пам’ять, що була динамічно розподілена. Також мова пропонує механізми для обробки вийняткових ситуацій та помилок.
5. Безпека. Java містить механізми перевірки на рівні JVM та на рівні мови, що забезпечують безпеку для розподілених систем, що часто містять віруси.

В будь-якій мові є слова, які не можна використовувати в якості найменувань. Розглянемо список таких слів для мови Java, на які слід звернути увагу [15]:

- Примітивні типи даних: byte, short, int, long, float, double, char, boolean;

- Написання циклів: do, while, for, continue, else, switch, break, default, case;
- Методи та модифікатори: private, public, protected, final, static, return, void, abstract, synchronized;
- Константи: true, false, null;
- Пакети та оператори імпорту: package, import;
- Обробка вийняткових ситуацій: try, catch, throw, throws, finally;
- Інше: new, class, interface, this, super, extends, implements;

Одиницями структуру мови Java є аплети, простір імен, файли програм, класи, інтерфейси, інше [16]. Розглянемо більш детально особливості деяких з них.

Аплети – це програми, які виконуються під керівництвом інших програм. Необхідні, коли можливостей мови, HTML недостатньо для реалізації веб-додатків та веб-сторінок [16].

Простір імен використовується для того, щоб забезпечити унікальність назв змінних та класів. Виділяють такі рівні вкладеності імен:

1. Простір імен пакетів – 0
2. Простір імен файлів класів – 1
3. Простів імен типів – 2
4. Простір імен методів – 3
5. Простір імен локального блоку – 4

Файли програми - це текстовим файлом з розширенням .java

Пакети. У більшості мов програмування набір зв’язаних класів або методів називається бібліотекою. Java для набору зв’язаних класів використовує термін пакет. Наприклад, java.lang містить базові функції Java.

Оператори імпорту – import java.*. Необхідні для використання вже існуючих класів пакетів.

Оголошення класів. Клас є похідним від системного Object. В Java, навідмінну від C++, доступне лише одинарне наслідування.

Оголошення інтерфейсу. В мові Java інтерфейс – це абстрактний клас, яких необхідний для імплементації наслідування від декількох класів.

2.3 Огляд Spring Framework

Spring Framework – Java фреймворк з відкритим кодом, що активно розвивається. Був розроблений компанією Pivotal Software. Spring – це більше ніж бібліотека, це каркас, що визначає структури системи та надає патерни, забезпечує комплексну програмну та кофігураційну модель для сучасних Java-enterprise додатків. Spring посідає високу позицію серед Java фреймворків, технологія не стоїть на місці, регулярно видаються свіжі версії існуєчих бібліотек та додаються нові. Дана інфраструктурна підтримка дає змогу командам розробників приділяти більше уваги бізнес-логіці, оскільки комплексні технічні аспекти покриваються можливостями фреймворку [8].

Spring Framework представляє собою контейнер впровадження залежностей, включає багато компонентів: Spring Security, Spring MVC, Spring Boot, Spring Web Flow та інше. Це дозволяє швидше та раціональніше створювати Java-додатки різної складності та призначення [8].

Spring Boot – допомагає швидко створити прототип Spring додатку, використовуючи конфігурацію за допомогою анотацій та стандартного коду. Часто використовується для розробки мікросервісів.

Spring Security – JavaEE framework, представляє механізм побудови систем аутентифікації, авторизації та інших можливостей для забезпечення безпеки корпоративних додатків. Ключовими елементами контексту Spring Security є SecurityContextHolder, SecurityContext, Authentication, GrantedAuthority, UserDetails, UserDetailsService. SecurityContext зберігає об’єкт Authentication, який представляє користувача з точки зору Spring Security. GrantedAuthority відображає обмеження в масштабі всього додатку, наприклад ROLE_USER, ROLE_ADMIN. UserDetails представляє необхідну інформацію для створення об’єктів Authentication. UserDetailsService використовується, щоб створити UserDetails об’єкт шляхом реалізації методів цього інтерфейсу [14].

Spring MVC (model – view – controller) – компонента, для створення веб-сайтів або RESTful сервісів. Фреймворк легко інтегрується в екосистему Spring, підтримує контролери та Rest контролери в Spring Boot додатках [11].

Spring Web Flow – базується на Spring MVC, представляє декларатиіну мову для опису потоків з високим рівнем абстракції.

Spring Data JDBC (Java Database Connection) – компонента для реалізації доступу до бази даних. Spring JDBC покриває питання відкриття з’єднання, виконання SQL запитів, обробка виключень, транзакцій та закриття з’єднання. Spring JDBC зручніший, ніж звичайний JDBC, оскільки завдяки наявним шаблонам, розробнику достатньо вказати параметри з’єднання та оператори SQL [12].

Розглянувши декілька компоненів фреймворку Spring, можна зробити висновок, що додаток буде складатися з декількох шарів та залежностей. Типовий сервіс має шари доступу до бази даних контролерів, сервісів та юніт тестів.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ

3.1 Проектування бази даних

Реалізована API не буде мати графічної реалізації, тому значну увагу слід приділити базі даних та представлення її схеми. Коректно названі таблиці та їх вміст значно полегшить подальшу розробку API. Для проекту дипломної роботи буде обрано СУБД MySQL. MySQL – це одна з найпоширеніших система керування реляційними базами даних, з відкритим кодом і є альтернативою комерційним системам. MySQL має підтримку з боку різних мов програмування, тому чудово підходить для створення прикладних програмних інтерфейсів та динамічних веб-сторінок [6].

Визначимо чотири основні сутності та їх атрибути. Відобразимо їх на ER діаграмі [7] (Рисунок 3.1).

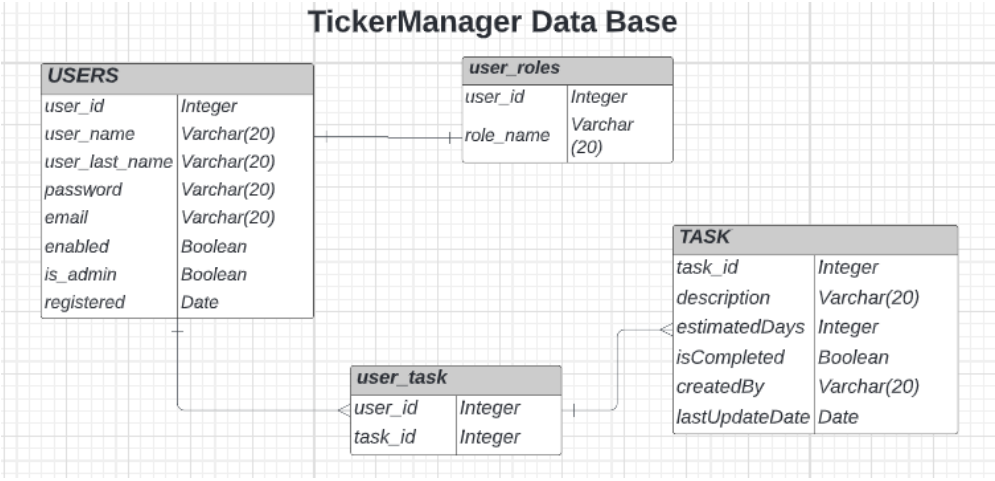

Рисунок 3.1 — ER діаграма БД

USERS:

- user_id: ідентифікатор користувача
- user_name: ім’я користувача
- user_last_name: прізвище користувача
- password: пароль
- email: пошта
- enabled: чи є користувач активним

- is_admin: чи має користувач роль адміністратора
- registered: чи зареєстрований користувач

TASK:

- task_id: ідентифікатор задачі
- description: опис задачі
- estimatedDays: кількість днів на її виконання
- isCompleted: статус задачі
- createdBy: час створення задачі
- lastUpdateDate: час останьої зміни задачі

USER_ROLES:

- user_id: ідентифікатор користувача
- role_name: роль користувача

USER_TASK:

- user_id: ідентифікатор користувача
- task_id: ідентифікатор задачі

3.2 Основні інтерфейси, класи та методи

Визначимо три entity, що визначатимуть ключові об’єкти в нашій системі:

- Role.java - перелік enum Role з двома значеннями: ROLE_USER, ROLE_ADMIN
- Task.java – клас, в якому визначимо конструктор з полями id, description, estimatedDays, isCompleted, createdBy, lastUpdateDate.
- User.java – клас, в якому визначимо конструктор з полями id, username, password, email, enable, isAdmin, registered

Не дивлячись на те, що класи є незначними, вони містять атрибути об’єктів, анотацію @Entity, що робить об’єкти готовими для зберігання в базі даних та створені конструктори, які необхідні, коли ми створюємо новий об’єкт, але ще не маємо для нього ідентифікатор.

Основним понятям в Spring Data є репозиторій. Тому визначимо два репозиторії, для опису взаємодії з Entity:

- TaskRepository.java

- UserRepository.java

Далі визначимо три інтерфейси-сервіси:

- EmailService.java
- TaskService.java
- UserService.java

Створемо відповідно три класи, які імплементують відповідні інтерфейси:

- EmailServiceImpl.java
- TaskServiceImpl.java
- UserServiceImpl.java

Визначимо контроллери, які реалізують виконання запитів: TaskController.java, TaskUsersController.java, UserController.java, UserTasksController.java. Опишемо перелік реалізованих запитів.

1. TaskController:
    2. GET /api/tasks/ - виведення списку всіх наявних задач
    3. GET /api/tasks/{id} – виведення інформації про задачу за її ідентифікатором
    4. PUT /api/tasks/{id} – оновлання інформації про задачу
    5. DELETE /api/tasks/{id} - видалення задачі за її ідентифікатором
2. User Controller:
    1. GET /api/users – виведення списку всіх користувачів
    2. GET /api/users/{id}- виведенн інформації про користувача за його ідентифікатором
    3. POST /api/users/ - збереження інформації про користувача
    4. PUT /api/users/{id} – оновлення інфомрації про користувача
    5. PATCH /api/users/ - надіслати запит на оновлення паролю
    6. PATCH /api/users/{id} – зміна паролю
    7. DELETE /api/users/{id} – видалення користувача за його ідентифікатором
3. UserTaskController:

1. GET /api/users/{id}/tasks – виведення задач, які належать користувачу за його ідентифікатором
2. DELETE /api/users/{id}/tasks – видалення всіх задач, що належать користувачу за його ідентифікатором

4. TaskUserController:
    1. GET /api/tasks/{id}/users – виведення інформації про користувачу, якому належить задача за її ідентифікатором
    2. POST /api/tasks/users/{id} – створення задачі для користувача
    3. PUT /api/task/{id}/users

Також окремо реалізовано вийнятки на випадок помилкових дій користувача:

1. EntityAlreadyExistsException.java
2. EntityNotFoundException.java
3. ValidationException.java

3.3 Перевірка роботи API

Розгляне приклади використання REST API. Тестування буде виконано за допомогою сервісу Postman, що представляє інструменти для роботи з API та спрощує роботу на всіх етапах життєвого циклу створення прикладних програмних інтерфейсів [13].

Розглянемо запит реєстрації користувача (Рисунок 3.2). В тілі запиту представлений JSON з необхідними даними. Статус запиту є успішним.

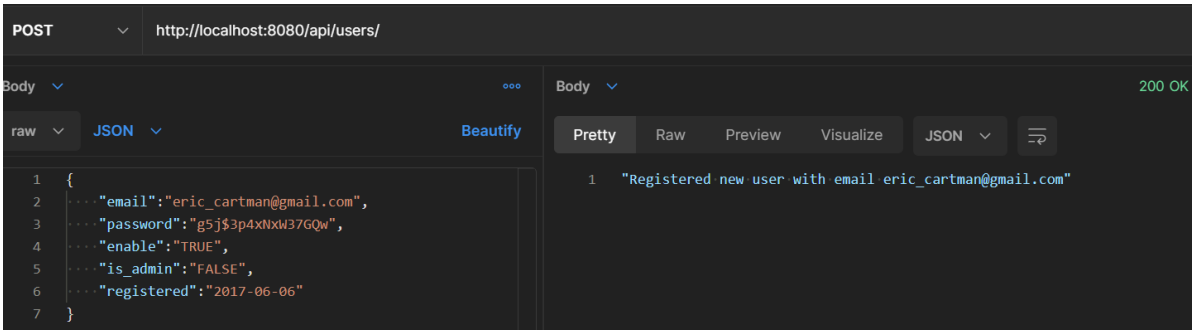

Рисунок 3.2 — Збереження користувача

Якщо користувач намагається використати поштову адресу повторно, то результатом запиту на збереження користувача буде помилка 400 Bad Request (Рисунок 3.3).

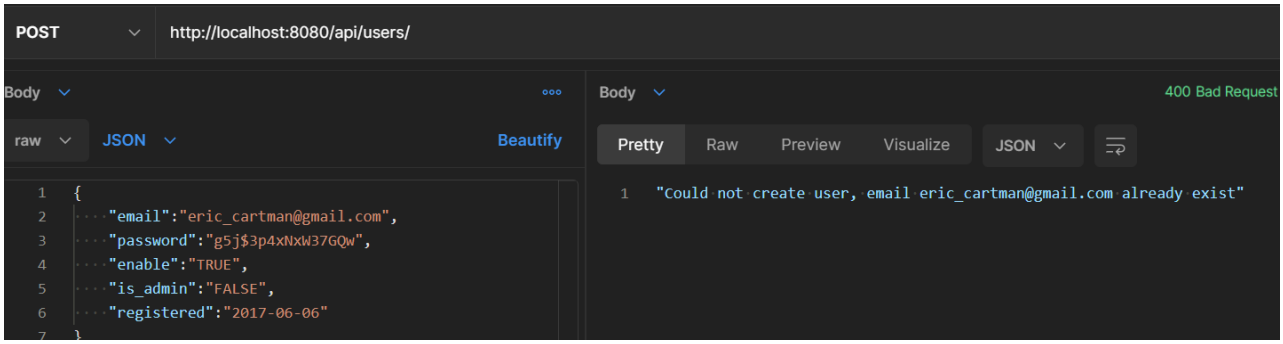

Рисунок 3.3 — Неуспішна реєстрація

Робота з додатком завжди починається з авторизації. Наведемо приклад успішної авторизації активованого користувача (Рисунок 3.4). В тілі запиту є два обов'язкових параметри: адреса електроної пошти та пароль.

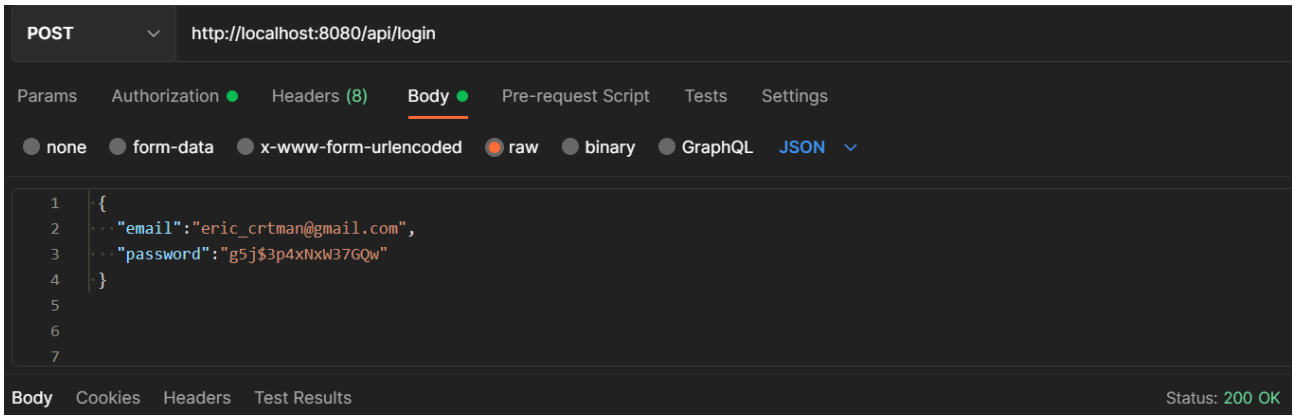

Рисунок 3.4 — Авторизація користувача

Також реалізовано випадок, коли користувач використовує невірний пароль під час процесу авторизації. В результаті виконання запиту отримаємо негативну відповідь з HTTP статус кодом 401 Unauthorized (Рисунок 3.5).

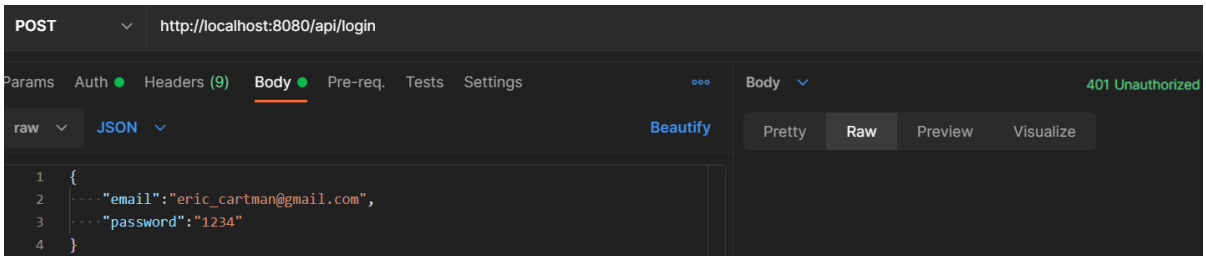

Рисунок 3.5 — Помилка авторизації

У випадку спроби неактивованого користувача пройти авторизацію, результатом запиту буде помилка з HTTP статусом 403 Forbidden (Рисунок 3.6).

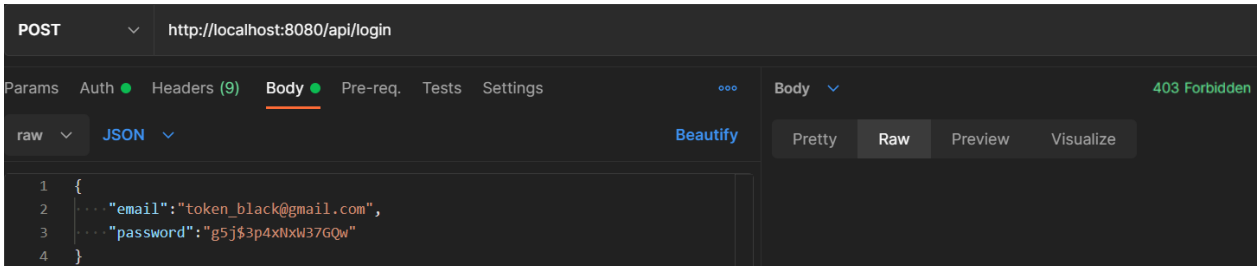

Рисунок 3.6 — Аккаунт не підтверджений

Користувач може оновити пароль. Спочатку викликається запит на оновлення паролю (Рисунок 3.7). Потім користувач створює новий пароль. Обов`язковими параметрами запиту є старий та новий паролі (Рисунок 3.8).

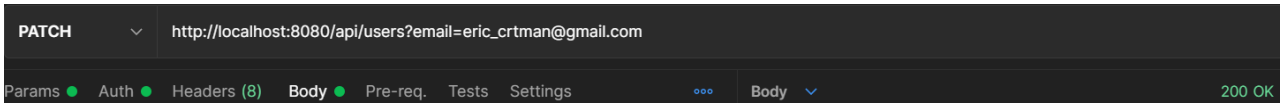

Рисунок 3.7 — Запит на оновлення паролю

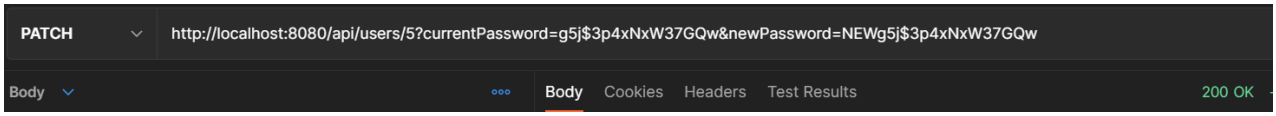

Рисунок 3.8 — Зміна паролю

Передбачено випадок спроби використання невірної пошти для запиту на оновлення паролю. Результатом буде помилка 400 Bad Request та повідомлення, що було використано некоректну поштову адресу користувача (Рисунок 3.9).

У випадку, якщо користувач створює некоректний новий пароль, довжиною меньше 8 символів, результатом запиту теж буде помилка 400 Bad Request та повідомлення, що змінити пароль неможливо, оскільки він не відповідаю вимогам (Рисунок 3.10).

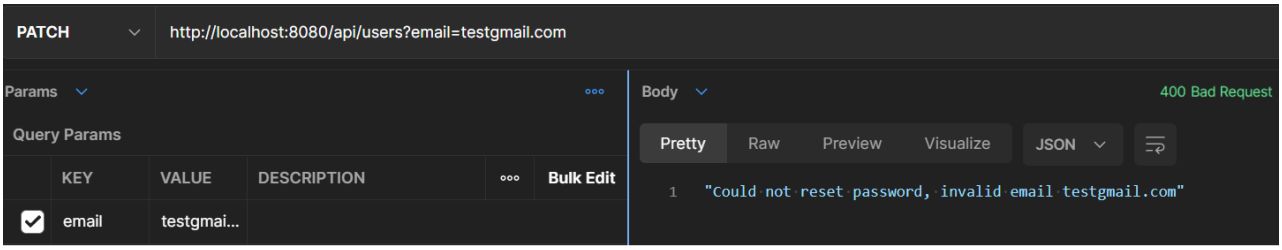

Рисунок 3.9 — Неуспішний запит на оновлення паролю

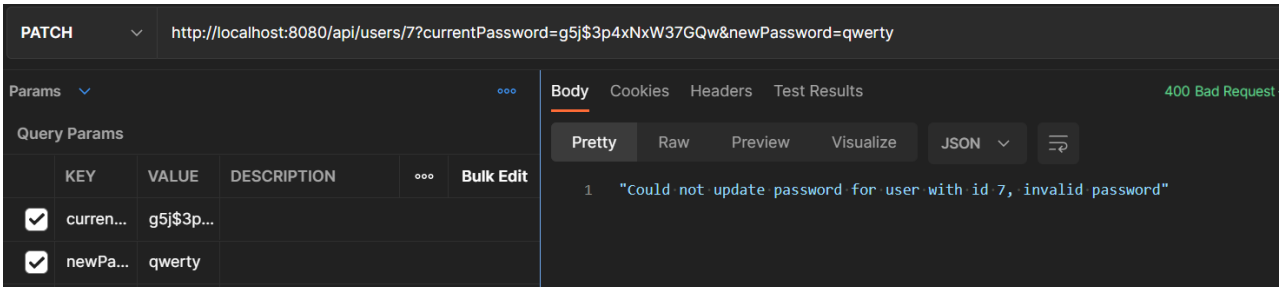

Рисунок 3.10 — Неуспішна зміна паролю

Функціонал також включає запит на оновлення даних про користувача (Рисунок 3.11). Наведемо приклад надання користувачу ролі адміністратора – “is_admin”: “TRUE”.

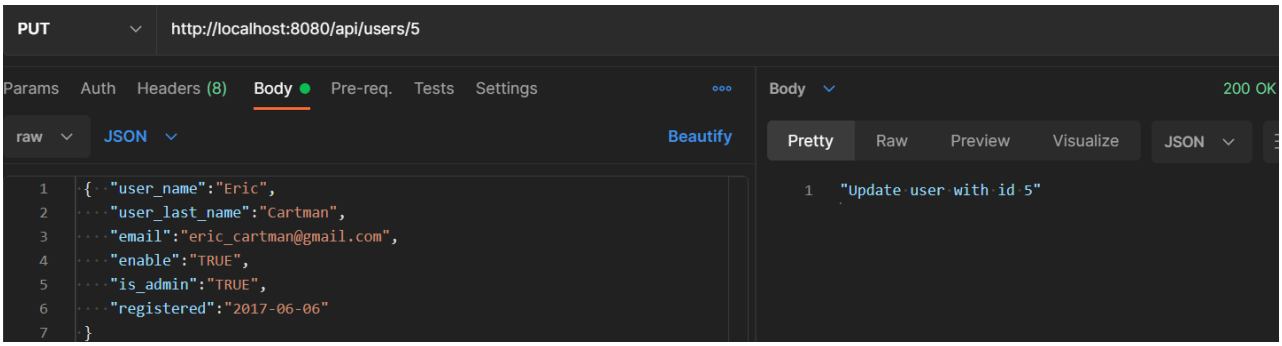

Рисунок 3.11 — Оновлення інфрмації про користувача

Адміністратор може переглянути список всіх користувачів. Розглянемо виконання запиту на виведення списку всіх користувачів та інформації про них (Рисунок 3.12).

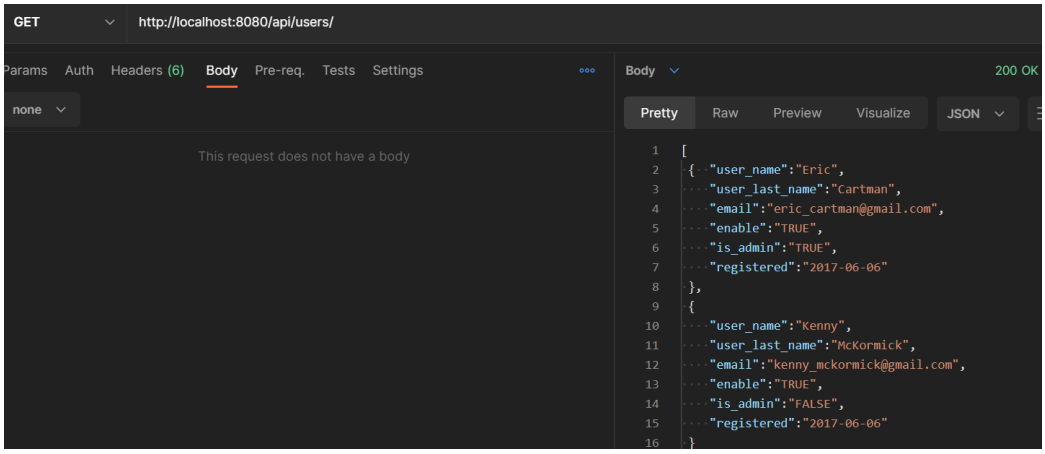

Рисунок 3.12 — Виведення інформації про користувачів

Також можливо переглянути інформацію про конкретного користувача за його ідентифікатором (Рисунок 3.13). Результатом такого запиту є JSON з даними про юзера.

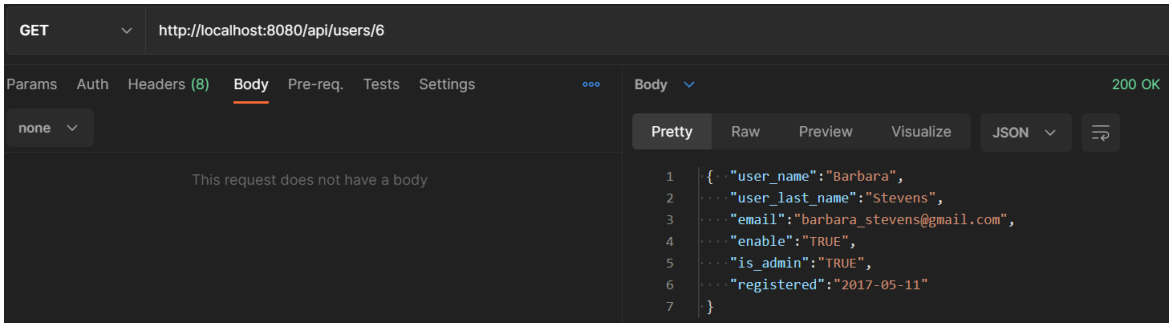

Рисунок 3.13 — Виведення інформації про користувача

Користувач може видалити свій профіль. Обов’язковим параметром для виконання цього запиту є пароль користуача.. Приклад успішного виконання запиту наведено нижче (Рисунок 3.14).

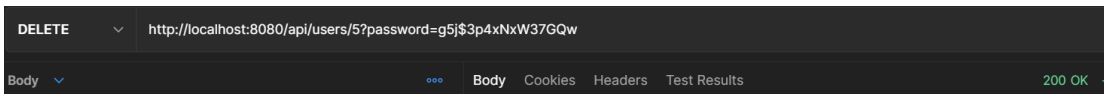

Рисунок 3.14 — Видалення профілю користувача

Адміністратор може переглянути список всіх задач. Наведемо результат роботи запиту на виведення інформація про наявні задачі (Рисунок 3.15).

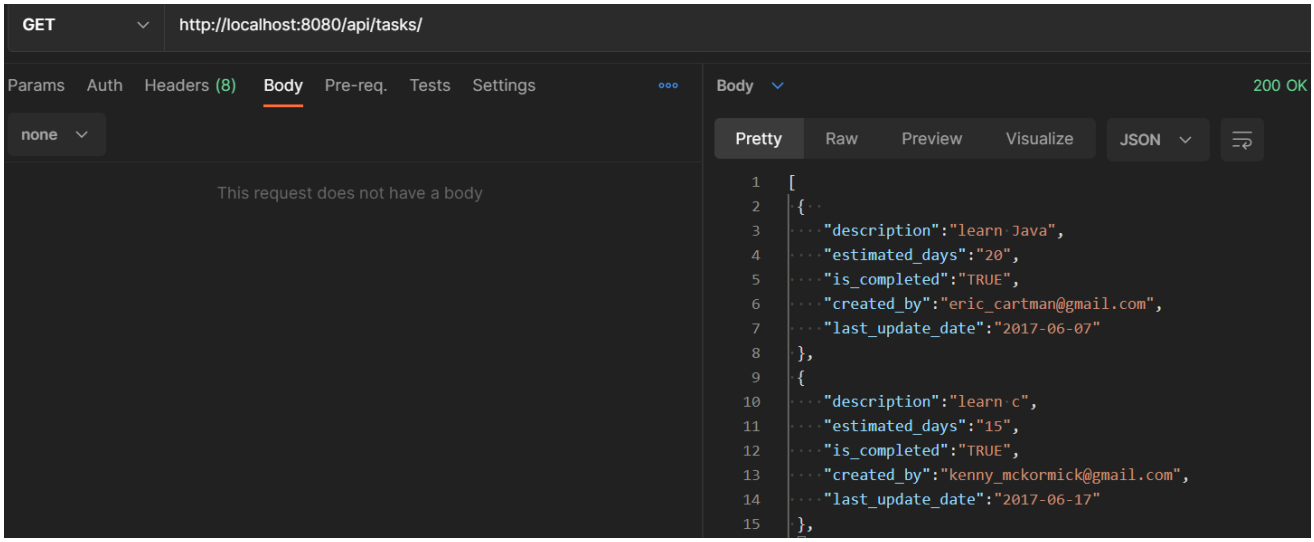

Рисунок 3.15 — Виведеня списку всіх задач

Також реалізовано запит на перегляд інформації про задачу за її ідентифікатором (Рисунок 3.16).

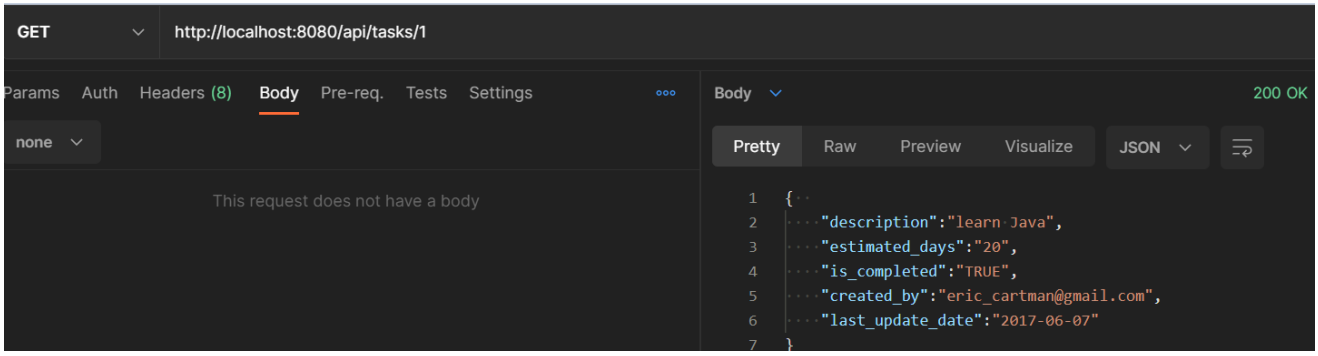

Рисунок 3.16 — Виведення інформації про задачу

Користувач може оновити інформацію про задачу. Наприклад змінити стасус на «виконано» (Рисунок 3.17).

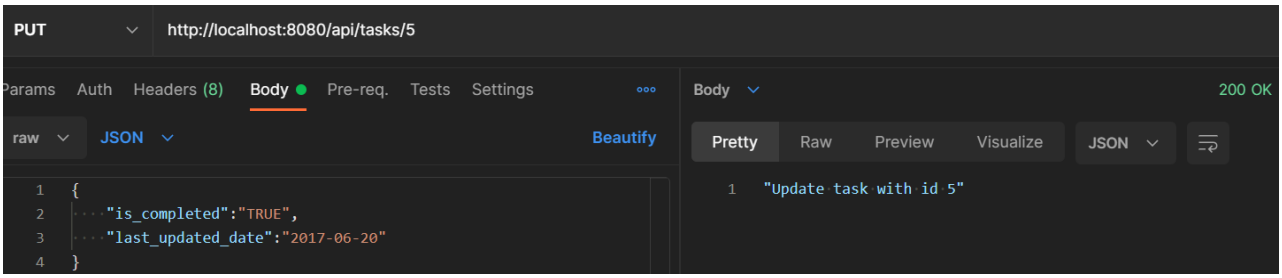

Рисунок 3.17 — Оновлення інформації про задачу за її айді

Якщо анонімний користувач буде намагатися переглянути будь яку інформацію додатку, то результатом такого запиту буде 401 Unautorized. Наведемо приклад запиту на перегляд інформації про задачу анонімним користувачем (Рисунок 3.18).

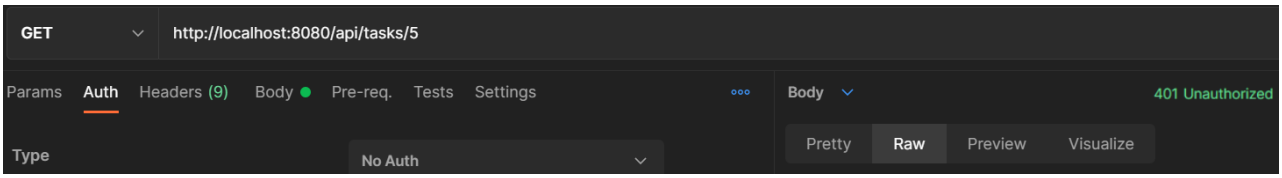

Рисунок 3.18 — Анонімний користувач

Якщо неактивований користувач або користувач без доступу до ресурсу буде намагатися переглянути інформацію, то результатом такого запиту буде 403 Forbidden. Наведемо приклда запит на перегляд інформації про задачу неактивованим користувачем (Рисунок 3.19).

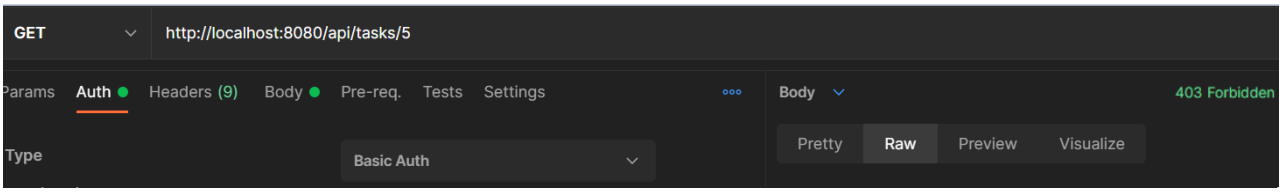

Рисунок 3.19 — Неактивований користувач

Користувач може переглянути кому належить задача. Виконаємо запит по визначенню хто є виконавцем задачі (Рисунок 3.20).

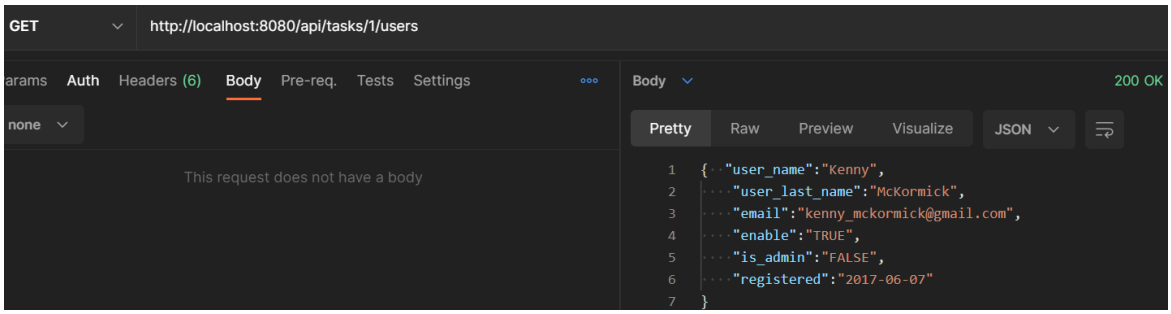

Рисунок 3.20 — Визначення кому належить задача

Також кожному активному користувачу є доступним перегляд всіх задач, що йому належать (Рисунок 3.21).

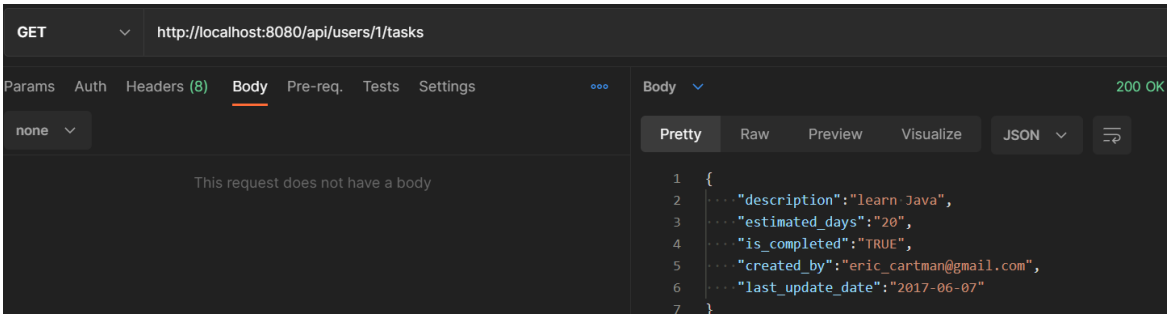

Рисунок 3.21 — Виведення інформації про задачі користувача

Користувач може створити нову задачу (Рисунок 3.22).

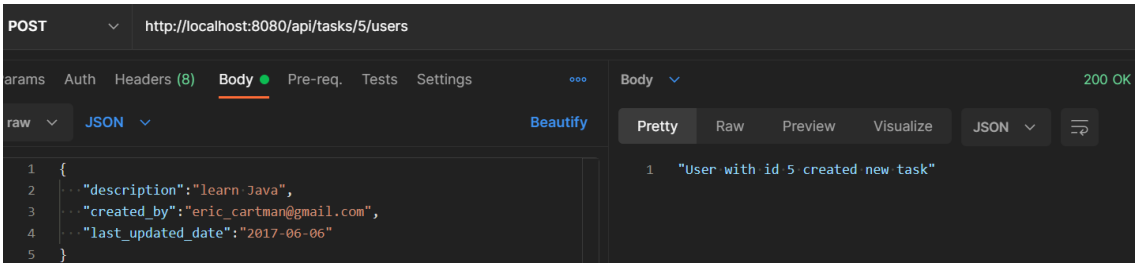

Рисунок 3.22 — Створення задачі та її збереження

Відповідно для кожної новоствореної задачі можна призначити виконавця (Рисунок 3.23).

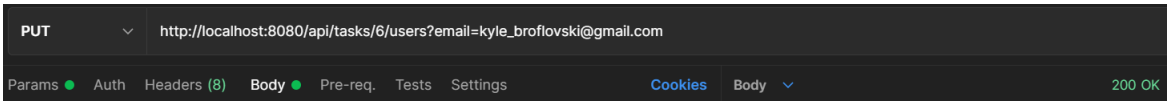

Рисунок 3.23 — Призначення задачі користувачу

Також реалізовані запити на видалення задачі за її ідентифікатором (Рисунок 3.24) та видалення всіх задач, що належать користувачу (Рисунок 3.25).

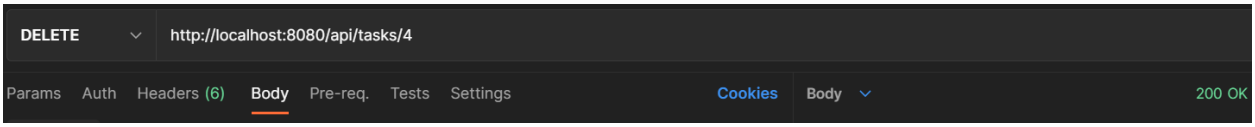

Рисунок 3.24 — Видалення задачі

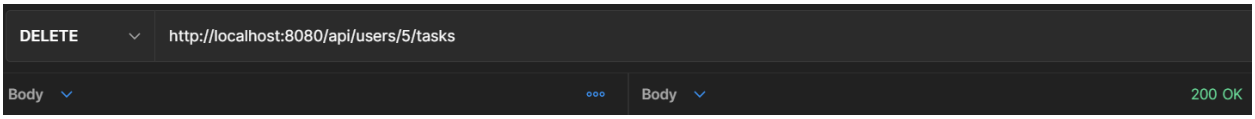

Рисунок 3.25 — Видалення всіх задач, що належать користувачу

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було виконану головну задачу: реалізовано прикладний програмний інтерфейс для управління задачами з використанням RestAPI та Spring Framework.

Перш за все було здійснено аналітичний огляд сервісів, що є аналогами. Розглянутий топ додатків на сучасному ринку став базисом для формування основних функціональних вимог до API.

За основу прикладного програмного інтерфейсу було обрано архітектурний підхід REST, що є стандартом для релізація сучасних веб-сервісів та API. Було розглянуто основні складові елементи REST, структура запитів та відповідей, HTTP статус коди.

Для програмної реалізація було використано Spring Framework. Фреймворк є на основі Java, включає в собі багато залежностей, що допомагають розробнику вирішити значну кількість технічних задач пов’язаних з API: реалізація авторизації, реєстрації, налаштування доступу до бази даних та інше.

Було проведено перевірку реалізованого програмного інтерфейсу, наведено приклади роботи основних функцій за допомогою Postman. У додатку роботи наведено код реалізація прикланого програмного інтерфейсу.

Проаналізувавши наведені приклади роботи реалізованої API, можна зробити висновок, що в майбутньому прикладний програмний інтерфейс можливо масштабувати, додавши нові методи. Доповнивши програму веб-реалізацією, можна створити багатофункціональний додаток зі зручним графічниим інтерфейсом, який буде конкурентноздатним на сучасному ринку.

СПИСОК ЛІТЕРАТУРИ

1. Jira Software features [Електроний ресурс]. – Режим доступу: https://www.atlassian.com/ru/software/jira/features (дата звернення 01.05.2022);
2. Trello – платформа управління проектами [Електроний ресур] – Режим доступу: https://trello.com/uk (дата звернення 01.05.2022);
3. Todoist [Електроний ресурс] – Режим доступу: https://todoist.com/uk (дата звернення 01.05.2022) ;
4. Jim Webber, Savas Parastatidis, Ian Robinson: REST in Practice: Hypermedia and Systems Architecture, 2010. – 448 с.
5. М. Тим Джонс. Використання REST-технологій [Електронний ресурс] - Режим доступу: https://www.ibm.com/cloud/learn/rest-apis (дата звернення 05.05.2022) ;
6. MySQL Features [Електронний ресурс] – Режим доступу: https://www.mysql.com/ (дата звернення 10.05.2022);
7. Entity Relationship Diagram (ERD) [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://drawio-app.com/ (дата звернення: 11.05.2022);
8. Spring Framework Documentation [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-framework/ (дата звернення 13.05.2022);
9. Spring Framework Rest Tutorials [Електронний ресурс] – Режим доступу: https://spring.io/guides/tutorials/rest/
10. SQL in Spring [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-boot/docs/2.1.13.RELEASE/reference/html/boot-features-sql.html (дата звернення: 20.05.2022);
11. Spring MVC tutorial [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html (дата звернення 20.05.2022);

12. Spring Data JDBC [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-data-jdbc (дата звернення 13.05.2022);
13. Postman tutorials [Електронний ресурс] – Режим доступу: https://www.postman.com/ (дата звернення 23.05.2022);
14. Spring Security Overview [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-security (дата звернення 13.05.2022);
15. Шілдт, Герберт: Java 8. Java для початківців, 2015. - 720 с.
16. Java Documentation [Електронний ресурс] – Режим доступу: https://docs.oracle.com/en/java/ (дата звернення 10.05.2022);

ДОДАТОК

Role.java

```
package com.diplom.taskmanager.entity;
public enum Role {
   ROLE_USER,
   ROLE_ADMIN;
}
```

Task.java

```
package com.diplom.taskmanager.entity;

import…

@Entity
@Table(name = "tasks")
public class Task implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "task_id")
   @JsonProperty(access = JsonProperty.Access.READ_ONLY)
   private Integer id;

   @Column(name = "description", length = 500)
   @NotNull
   @Pattern(regexp = ".{3,500}$")
   @SafeHtml
   private String description;

   @Column(name = "estimated_days")
   @NotNull
   @Range(max = 360L)
   private Integer estimatedDays;

   @Column(name = "is_completed")
   @NotNull
   private Boolean isCompleted;
```

```
@Column(name = "created_by")
@Pattern(regexp = ".{8,250}$")
@JsonProperty(access = JsonProperty.Access.READ_ONLY)
private String createdBy;

@Column(name = "last_update_date")
@JsonProperty(access = JsonProperty.Access.READ_ONLY)
private LocalDate lastUpdateDate;

@ManyToMany(fetch = FetchType.LAZY)
@JoinTable(name = "users_tasks",
      joinColumns = {@JoinColumn(name = "task_id")},
      inverseJoinColumns = {@JoinColumn(name = "user_id")})
private List<User> users = new ArrayList<>();

public Task() {
}
public Task(Integer id, String description, Integer estimatedDays, Boolean isCompleted,
         String createdBy, LocalDate lastUpdateDate) {
   this.id = id;
   this.description = description;
   this.estimatedDays = estimatedDays;
   this.isCompleted = isCompleted;
   this.createdBy = createdBy;
   this.lastUpdateDate = lastUpdateDate;
}

public Integer getId() {
   return id;
}
public void setId(Integer id) {
   this.id = id;
}
public String getDescription() {
   return description;
}
public void setDescription(String description) {
   this.description = description;
}
public Integer getEstimatedDays() {
   return estimatedDays;
}
public void setEstimatedDays(Integer estimatedDays) {
   this.estimatedDays = estimatedDays;
```

```
    }

    public Boolean getCompleted() {
        return isCompleted;
    }
    public void setCompleted(Boolean completed) {
        isCompleted = completed;
    }
    public String getCreatedBy() {
        return createdBy;
    }
    public void setCreatedBy(String createdBy) {
        this.createdBy = createdBy;
    }
    public LocalDate getLastUpdateDate() {
        return lastUpdateDate;
    }
    public void setLastUpdateDate(LocalDate lastUpdateDate) {
        this.lastUpdateDate = lastUpdateDate;
    }
    public List<User> getUsers() {
        return users;
    }
    public void setUsers(List<User> users) {
        this.users = users;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || !getClass().equals(Hibernate.getClass(o))) {
            return false;
        }
        Task that = (Task) o;
        return null != getId() && getId().equals(that.getId());
    }

    @Override
    public int hashCode() {
        return (getId() == null) ? 0 : getId();
    }
}
```

User.java

```
package com.diplom.taskmanager.entity;
import …

@Entity
@Table(name = "users")
public class User implements Serializable {

    private static final long serialVersionUID = 1L;
   private static final String PASSWORD_REGEX =
            "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\\S+$).{8,}$";
    private static final String EMAIL_REGEX =
            "^[_A-Za-z0-9-]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

    public static final java.util.regex.Pattern PASSWORD_PATTERN =
            java.util.regex.Pattern.compile(PASSWORD_REGEX);

    public static final java.util.regex.Pattern EMAIL_PATTERN =
            java.util.regex.Pattern.compile(EMAIL_REGEX);

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    @JsonProperty(access = JsonProperty.Access.READ_ONLY)
    private Integer id;

    @Column(name = "user_name")
    @NotNull
    @Pattern(regexp = "[a-zA-Z\\s']{3,250}$")
    @SafeHtml
    private String userName;

    @Column(name = "user_last_name")
    @NotNull
    @Pattern(regexp = "[a-zA-Z\\s']{3,250}$")
    @SafeHtml
    private String userLastName;

    @Column(name = "password")
    @Pattern(regexp = PASSWORD_REGEX)
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
```

```
private String password;

@Column(name = "email", unique = true)
@NotNull
@Pattern(regexp = EMAIL_REGEX)
@SafeHtml
private String email;

@Column(name = "enabled")
private Boolean enabled;

@Column(name = "is_admin")
private Boolean isAdmin;

@Column(name = "registered")
@JsonProperty(access = JsonProperty.Access.READ_ONLY)
private LocalDate registered;

@Column(name = "role")
@JsonIgnore
@Enumerated(EnumType.STRING)
@CollectionTable(name = "user_roles", joinColumns = @JoinColumn(name = "user_id"))
@ElementCollection(fetch = FetchType.EAGER)
private Set<Role> roles;

@ManyToMany(cascade = CascadeType.REMOVE)
@JoinTable(name = "users_tasks",
        joinColumns = {@JoinColumn(name = "user_id")},
        inverseJoinColumns = {@JoinColumn(name = "task_id")})
private List<Task> tasks;

public User() {
}

public User(Integer id, String userName, String userLastName, String password, String email,
            Boolean enabled, Boolean isAdmin, LocalDate registered) {
    this.id = id;
    this.userName = userName;
    this.userLastName = userLastName;
    this.password = password;
    this.email = email;
    this.enabled = enabled;
    this.isAdmin = isAdmin;
    this.registered = registered;
```

```
}

public Integer getId() {
    return id;
}
public void setId(Integer id) {
    this.id = id;
}
public String getUserName() {
    return userName;
}
public void setUserName(String userName) {
    this.userName = userName;
}
public String getUserLastName() {
    return userLastName;
}
public void setUserLastName(String userLastName) {
    this.userLastName = userLastName;
}
public String getPassword() {
    return password;
}
public void setPassword(String password) {
    this.password = password;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public Boolean getEnabled() {
    return enabled;
}
public void setEnabled(Boolean enabled) {
    this.enabled = enabled;
}
public Boolean getAdmin() {
    return isAdmin;
}
public void setAdmin(Boolean admin) {
    isAdmin = admin;
}
```

```
    public LocalDate getRegistered() {
        return registered;
    }
    public void setRegistered(LocalDate registered) {
        this.registered = registered;
    }
    public Set<Role> getRoles() {
        return roles;
    }
    public void setRoles(Set<Role> roles) {
        this.roles = CollectionUtils.isEmpty(roles) ? Collections.emptySet() : EnumSet.copyOf(roles);
    }
    public List<Task> getTasks() {
        return tasks;
    }
    public void setTasks(List<Task> tasks) {
        this.tasks = tasks;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || !getClass().equals(Hibernate.getClass(o))) {
            return false;
        }
        User that = (User) o;
        return null != getId() && getId().equals(that.getId());
    }

    @Override
    public int hashCode() {
        return (getId() == null) ? 0 : getId();
    }

}
```

TaskRepository.java

```
package com.diplom.taskmanager.repository;
import com.diplom.taskmanager.entity.Task;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
```

```
import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Transactional;

@Transactional(readOnly = true)
public interface TaskRepository extends JpaRepository<Task, Integer> {

   @Query("SELECT u.tasks FROM User u WHERE u.id=:userId")
   List<Task> findByUserId(@Param("userId") Integer userId);

   @Query("SELECT (COUNT (t) > 0) AS boolean FROM User u " +
         "INNER JOIN u.tasks t WHERE t.id =:id AND u.id=:userId ")
   Boolean isTaskContainingInUser(@Param("id") Integer id, @Param("userId") Integer userId);
}
```

UserRepository.java

```
package com.diplom.taskmanager.repository;
import com.diplom.taskmanager.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Transactional;
@Transactional(readOnly = true)
public interface UserRepository extends JpaRepository<User, Integer>
{
   User findByEmail(String email);

   @Query("SELECT t.users FROM Task t WHERE t.id=:taskId")
   List<User> findByTaskId(@Param("taskId") Integer taskId);

   @Query("SELECT (COUNT (u) > 0) FROM User u WHERE u.email=:email")
   boolean isEmailAlreadyExists(@Param("email") String email);
}
```

EmailService.java

```
package com.diplom.taskmanager.service;

public interface EmailService {
   void sendEmail(String email, String text);
}
```

EmailServiceImpl.java

```
package com.diplom.taskmanager.service.impl;


import com.diplom.taskmanager.service.EmailService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

import javax.inject.Inject;
import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;

@Service
public class EmailServiceImpl implements EmailService {

    private static final Logger LOG = LoggerFactory.getLogger(EmailServiceImpl.class);

    private final JavaMailSender javaMailSender;

    @Inject
    public EmailServiceImpl(JavaMailSender javaMailSender) {
        this.javaMailSender = javaMailSender;
    }

    @Value("${mail.from}")
    private String from;
    private String subject = "Task Manager";

    @Async
    @Override
    public void sendEmail(String email, String text) {
        MimeMessage message = javaMailSender.createMimeMessage();
        try {
            MimeMessageHelper helper = new MimeMessageHelper(message, "UTF-8");
            helper.setFrom(from);
            helper.setTo(email);
            helper.setSubject(subject);
```

```
            helper.setText(text);
            javaMailSender.send(message);
            LOG.error("Successful sending to mail {}", email);
        } catch (MessagingException e) {
            LOG.error("Couldn't send email to user with email {}: {}", email, e.getLocalizedMessage());
        }
    }
}
```

TaskService.java

```
package com.diplom.taskmanager.service;
import com.diplom.taskmanager.entity.Task;
import org.springframework.security.access.method.P;
import org.springframework.security.access.prepost.PreAuthorize;

import java.util.List;

public interface TaskService {

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForTask(authentication, #id)")
    Task getOne(@P("id") Integer id);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForUser(authentication, #userId)")
    List<Task> findByUserId(@P("userId") Integer userId);

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    List<Task> findAll();

    @PreAuthorize("@permissionEvaluator.hasPermissionForUser(authentication, #userId)")
    Task save(@P("userId") Integer userId, Task task);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForTask(authentication, #id)")
    Task update(@P("id") Integer id, Task task);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForTask(authentication, #id)")
    void share(@P("id") Integer id, String email);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
```

```
        " @permissionEvaluator.hasPermissionForTask(authentication, #id)")
    void delete(@P("id") Integer id);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
        " @permissionEvaluator.hasPermissionForUser(authentication, #userId)")
    void deleteAllForUser(@P("userId") Integer userId);}
```

TaskServiceImpl.java

```
package com.diplom.taskmanager.service.impl;

import com.diplom.taskmanager.entity.Task;
import com.diplom.taskmanager.entity.User;
import com.diplom.taskmanager.exception.EntityNotFoundException;
import com.diplom.taskmanager.exception.ValidationException;
import com.diplom.taskmanager.repository.TaskRepository;
import com.diplom.taskmanager.repository.UserRepository;
import com.diplom.taskmanager.service.EmailService;
import com.diplom.taskmanager.service.TaskService;
import org.springframework.stereotype.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import javax.inject.Inject;
import java.time.LocalDate;
import java.util.Collections;
import java.util.List;

@Service
public class TaskServiceImpl implements TaskService {

    private static final Logger LOG = LoggerFactory.getLogger(TaskServiceImpl.class);

    private final TaskRepository taskRepository;
    private final UserRepository userRepository;
    private final EmailService emailService;

    @Inject
    public TaskServiceImpl(TaskRepository taskRepository, UserRepository userRepository,
                           EmailService emailService) {
        this.taskRepository = taskRepository;
        this.userRepository = userRepository;
        this.emailService = emailService;
    }
```

```
@Override
public Task getOne(Integer id) {
    Task task = taskRepository.getOne(id);
    LOG.debug("Get task with id {}", task.getId());
    return task;
}

@Override
public List<Task> findByUserId(Integer userId) {
    List<Task> taskList = taskRepository.findByUserId(userId);
    LOG.debug("Find all tasks for user with id {}", userId);
    return taskList;
}

@Override
public List<Task> findAll() {
    List<Task> taskList = taskRepository.findAll();
    LOG.debug("Find all tasks");
    return taskList;
}

@Override
public Task save(Integer userId, Task task) {
    User user = userRepository.getOne(userId);
    task.setCreatedBy(user.getEmail());
    task.setLastUpdateDate(LocalDate.now());
    task.getUsers().add(user);
    Task savedTask = taskRepository.save(task);
    savedTask.setUsers(Collections.emptyList());
    LOG.debug("User with id {} create new task", userId);
    return savedTask;
}

@Override
public Task update(Integer id, Task updated) {
    Task task = taskRepository.getOne(id);
    task.setDescription(updated.getDescription());
    task.setEstimatedDays(updated.getEstimatedDays());
    task.setCompleted(updated.getCompleted());
    task.setLastUpdateDate(LocalDate.now());
    LOG.debug("Update task with id {}", id);
    return taskRepository.save(task);
}
```

```
    @Override
    public void share(Integer id, String email) {
        if (!EMAIL_PATTERN.matcher(email).matches()) {
            throw new ValidationException(
                    String.format("Could not share task, invalid email %s", email));
        }
        Task task = taskRepository.getOne(id);
        User user = userRepository.findByEmail(email);
        if (user == null) {
            throw new EntityNotFoundException(
                    String.format("Could not share task, user with email %s not found", email));
        }
        task.setLastUpdateDate(LocalDate.now());
        task.getUsers().add(user);
        taskRepository.save(task);
        emailService.sendEmail(user.getEmail(), "You have a new common task in Task Manager");
        LOG.debug("Share task with id {}, for user with email {}", id, email);
    }

    @Override
    public void delete(Integer id) {
        Task task = taskRepository.getOne(id);
        task.setUsers(Collections.emptyList());
        taskRepository.save(task);
        taskRepository.delete(task);
        LOG.debug("Delete task with id {}", id);
    }

    @Override
    public void deleteAllForUser(Integer userId) {
        User user = userRepository.getOne(userId);
        user.setTasks(Collections.emptyList());
        userRepository.save(user);
        LOG.debug("Delete all tasks for user with id {}", userId);
    }
}
```

UserService.java

```
package com.diplom.taskmanager.service;

import com.diplom.taskmanager.entity.User;
```

```
import org.springframework.security.access.method.P;
import org.springframework.security.access.prepost.PreAuthorize;
import java.util.List;

public interface UserService {

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForUser(authentication, #id)")
    User getOne(@P("id") Integer id);

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    List<User> findByTaskId(Integer taskId);

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    List<User> findAll();

    void save(User user);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForUser(authentication, #id)")
    User update(@P("id") Integer id, User user);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForUser(authentication, #id)")
    void updatePassword(@P("id") Integer id, String currentPassword, String newPassword);

    void resetPassword(String email);

    @PreAuthorize("hasRole('ROLE_ADMIN') OR" +
            " @permissionEvaluator.hasPermissionForUser(authentication, #id)")
    void delete(@P("id") Integer id, String password);

}
```

UserServiceImpl.java

```
package com.diplom.taskmanager.service.impl;

import com.diplom.taskmanager.entity.Role;
import com. diplom.taskmanager.entity.User;
import com. diplom.taskmanager.exception.EntityAlreadyExistsException;
import com. diplom.taskmanager.exception.EntityNotFoundException;
import com. diplom.taskmanager.exception.ValidationException;
```

```
import com. diplom.taskmanager.repository.UserRepository;
import com. diplom.taskmanager.service.EmailService;
import com. diplom.taskmanager.service.UserService;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.core.Authentication;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import javax.inject.Inject;
import java.time.LocalDate;
import java.util.Collections;
import java.util.List;
import java.util.Objects;
import java.util.UUID;

@Service
public class UserServiceImpl implements UserService {

   private static final Logger LOG = LoggerFactory.getLogger(UserServiceImpl.class);

   private final UserRepository userRepository;
   private final EmailService emailService;
   private final BCryptPasswordEncoder bCryptPasswordEncoder;

   @Inject
   public UserServiceImpl(UserRepository userRepository, EmailService emailService,
                          BCryptPasswordEncoder bCryptPasswordEncoder) {
      this.userRepository = userRepository;
      this.emailService = emailService;
      this.bCryptPasswordEncoder = bCryptPasswordEncoder;
   }

   @Override
   public User getOne(Integer id) {
      User user = userRepository.getOne(id);
      LOG.debug("Get user with id {}", user.getId());
      return user;
   }

   @Override
   public List<User> findByTaskId(Integer taskId) {
      List<User> userList = userRepository.findByTaskId(taskId);
      LOG.debug("Find users who have a task with id {}", taskId);
```

```
    return userList;
}

@Override
public List<User> findAll() {
    List<User> userList = userRepository.findAll();
    LOG.debug("Find all users");
    return userList;
}

@Override
public void save(User user) {
    if (userRepository.isEmailAlreadyExists(user.getEmail())) {
        throw new EntityAlreadyExistsException(
                String.format("Could not create user, email %s already exist", user.getEmail()));
    }
    user.setEmail(user.getEmail().toLowerCase());
    user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
    user.setEnabled(true);
    user.setAdmin(false);
    user.setRegistered(LocalDate.now());
    user.setRoles(Collections.singleton(Role.ROLE_USER));
    userRepository.save(user);
    emailService.sendEmail(user.getEmail(), "Welcome to Task Manager");
    LOG.debug("Registered new user with email {}", user.getEmail());
}

@Override
public User update(Integer id, User updated) {
    User user = userRepository.getOne(id);
    if (!Objects.equals(user.getEmail(), updated.getEmail()) &&
            userRepository.isEmailAlreadyExists(updated.getEmail())) {
        throw new EntityAlreadyExistsException(
                String.format("Could not update user with id %d , email %s already exist",
                        id, updated.getEmail()));
    }
    user.setUserName(updated.getUserName());
    user.setUserLastName(updated.getUserLastName());
    user.setEmail(updated.getEmail().toLowerCase());
    if (updated.getEnabled() != null && updated.getAdmin() != null) {
        user.setEnabled(updated.getEnabled());
        user.setAdmin(updated.getAdmin());
    }
    if (Boolean.TRUE.equals(updated.getAdmin()) && !user.getRoles().contains(Role.ROLE_ADMIN)) {
```

```
        user.getRoles().add(Role.ROLE_ADMIN);
    } else {
        user.setRoles(Collections.singleton(Role.ROLE_USER));
    }
    LOG.debug("Update user with id {}", id);
    return userRepository.save(user);
}

@Override
public void updatePassword(Integer id, String currentPassword, String newPassword) {
    if (!PASSWORD_PATTERN.matcher(currentPassword).matches() ||
            !PASSWORD_PATTERN.matcher(newPassword).matches()) {
        throw new ValidationException(
                String.format("Could not update password for user with id %d , invalid password", id));
    }
    User user = userRepository.getOne(id);
    if (!bCryptPasswordEncoder.matches(currentPassword, user.getPassword())) {
        throw new AccessDeniedException(
                String.format("Could not update password, user with id %d has not confirmed password", id));
    }
    user.setPassword(bCryptPasswordEncoder.encode(newPassword));
    userRepository.save(user);
    emailService.sendEmail(user.getEmail(), "Your password is updated in Task Manager");
    LOG.debug("User with id {} update password", id);
}
@Override
public void resetPassword(String email) {
    if (!EMAIL_PATTERN.matcher(email).matches()) {
        throw new ValidationException(
                String.format("Could not reset password, invalid email %s", email));
    }
    User user = userRepository.findByEmail(email);
    if (user == null) {
        throw new EntityNotFoundException(
                String.format("Could not reset password, user with email %s not found", email));
    }
    String newPassword = UUID.randomUUID().toString().replace("-", "P$");
    user.setPassword(bCryptPasswordEncoder.encode(newPassword));
    emailService.sendEmail(email, String.format(
            "Password reset in Task Manager, your new password: %s", newPassword));
    LOG.debug("User with email {} reset password", email);
}

@Override
```

```
    public void delete(Integer id, String password) {
        if (!PASSWORD_PATTERN.matcher(password).matches()) {
            throw new ValidationException(
                    String.format("Could not delete user with id %d, invalid password", id));
        }
        User user = userRepository.getOne(id);
        if (!bCryptPasswordEncoder.matches(password, user.getPassword())) {
            throw new AccessDeniedException(
                    String.format("Could not delete user with id %d, incorrect password", id));
        }
        user.setTasks(Collections.emptyList());
        userRepository.save(user);
        userRepository.delete(user);
        emailService.sendEmail(user.getEmail(), "Your account has been deleted in Task Manager");
        LOG.debug("Delete user with id {}", id);
    }
}
```

UserController.java

```
package com.diplom.taskmanager.controller;

import com.diplom.taskmanager.entity.User;
import com.diplom.taskmanager.service.UserService;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import javax.inject.Inject;
import javax.validation.Valid;
import java.util.List;
@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

    @Inject
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping(value = "/", produces = MediaType.APPLICATION_JSON_VALUE)
    public List<User> getAll() {
```

```
        return userService.findAll();
    }

    @GetMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public User getOne(@PathVariable Integer id) {
        return userService.getOne(id);
    }

    @PostMapping(value = "/", consumes = MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.CREATED)
    public void create(@Valid @RequestBody User user) {
        userService.save(user);
    }

    @PutMapping(value = "/{id}",
            consumes = MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
    public User update(@PathVariable Integer id, @Valid @RequestBody User user) {
        return userService.update(id, user);
    }

    @PatchMapping
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void resetPassword(@RequestParam(value = "email") String email) {
        userService.resetPassword(email);
    }

    @PatchMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void updatePassword(@PathVariable Integer id,
                               @RequestParam(value = "currentPassword") String currentPassword,
                               @RequestParam(value = "newPassword") String newPassword) {
        userService.updatePassword(id, currentPassword, newPassword);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void delete(@PathVariable Integer id,
                       @RequestParam(value = "password") String password) {
        userService.delete(id, password);
    }

}
```

TaskController.java

```
package com.diplom.taskmanager.controller;
import com.diplom.taskmanager.entity.Task;
import com.diplom.taskmanager.service.TaskService;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import javax.inject.Inject;
import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/tasks")
public class TaskController {

    private final TaskService taskService;

    @Inject
    public TaskController(TaskService taskService) {
        this.taskService = taskService;
    }

    @GetMapping(value = "/", produces = MediaType.APPLICATION_JSON_VALUE)
    public List<Task> getAll() {
        return taskService.findAll();
    }

    @GetMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public Task getOne(@PathVariable Integer id) {
        return taskService.getOne(id);
    }

    @PutMapping(value = "/{id}",
            consumes = MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
    public Task update(@PathVariable Integer id, @Valid @RequestBody Task task) {
        return taskService.update(id, task);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void delete(@PathVariable Integer id) {
```

```
        taskService.delete(id);
    }

}
```

TaskUsersController.java

```
package com.diplom.taskmanager.controller;

import com.diplom.taskmanager.entity.Task;
import com.diplom.taskmanager.entity.User;
import com.diplom.taskmanager.service.TaskService;
import com.diplom.taskmanager.service.UserService;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import javax.inject.Inject;
import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/tasks")
public class TaskUsersController {

    private final TaskService taskService;
    private final UserService userService;

    @Inject
    public TaskUsersController(TaskService taskService, UserService userService) {
        this.taskService = taskService;
        this.userService = userService;
    }
    @GetMapping(value = "/{id}/users", produces = MediaType.APPLICATION_JSON_VALUE)
    public List<User> getUsers(@PathVariable Integer id) {
        return userService.findByTaskId(id);
    }
    @PostMapping(value = "/users/{id}",
            consumes = MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.CREATED)
    public Task createTask(@PathVariable Integer id, @Valid @RequestBody Task task) {
        return taskService.save(id, task);
    }
```

```
    @PutMapping("/{id}/users")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void shareTask(@PathVariable Integer id, @RequestParam(value = "email") String email) {
        taskService.share(id, email);
    }

}
```

UserTasksController.java

```
package com.diplom.taskmanager.controller;

import com.diplom.taskmanager.entity.Task;
import com.diplom.taskmanager.service.TaskService;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

import javax.inject.Inject;
import java.util.List;

@RestController
@RequestMapping("/api/users")
public class UserTasksController {

  private final TaskService taskService;

    @Inject
    public UserTasksController(TaskService taskService) {
        this.taskService = taskService;
    }
    @GetMapping(value = "/{id}/tasks", produces = MediaType.APPLICATION_JSON_VALUE)
    public List<Task> getTasks(@PathVariable Integer id) {
        return taskService.findByUserId(id);
    }

    @DeleteMapping("/{id}/tasks")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteTasks(@PathVariable Integer id) {
        taskService.deleteAllForUser(id);
    }

}
```

EntityAlreadyExistsException.java

```
package com.diplom.taskmanager.exception;

public class EntityAlreadyExistsException extends RuntimeException {

    public EntityAlreadyExistsException(String message) {
        super(message);
    }

}
```

EntityNotFoundException.java

```
package com.diplom.taskmanager.exception;

public class EntityNotFoundException extends RuntimeException {

    public EntityNotFoundException(String message) {
        super(message);
    }
}
```

ValidationException.java

```
package com.diplom.taskmanager.exception;

public class ValidationException extends RuntimeException {

    public ValidationException(String message) {
        super(message);
    }

}
```

ApplicationConfiguration.java

```
package com.diplom.taskmanager.configuration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.*;
import org.springframework.context.support.PropertySourcesPlaceholderConfigurer;
```

```
import org.springframework.mail.javamail.JavaMailSenderImpl;
import java.util.Properties;

@Configuration
@ComponentScan
@PropertySource(value = {"classpath:email/email.properties"})
@Import({ServiceConfiguration.class, MvcConfiguration.class, JpaMysqlConfiguration.class,
JpaHsqldbConfiguration.class})
public class ApplicationConfiguration {

   @Bean
   public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {
      return new PropertySourcesPlaceholderConfigurer();
   }

   @Bean
   public JavaMailSenderImpl mailSender(@Value("${mail.host}") String mailHost,
                                        @Value("${mail.port}") Integer mailPort,
                                        @Value("${mail.username}") String mailUsername,
                                        @Value("${mail.password}") String mailPassword,
                                        @Value("${mail.smtp.auth}") String mailSmtpAuth,
                                        @Value("${mail.smtp.port}") String mailSmtpPort,
                                        @Value("${mail.smtp.socketFactory.class}")
                                              String mailSmtpSocketFactoryClass,
                                        @Value("${mail.smtp.socketFactory.port}")
                                              String mailSmtpSocketFactoryPort) {
      JavaMailSenderImpl javaMailSender = new JavaMailSenderImpl();
      Properties props = new Properties();
      props.put("mail.smtp.auth", mailSmtpAuth);
      props.put("mail.smtp.port", mailSmtpPort);
      props.put("mail.smtp.socketFactory.class", mailSmtpSocketFactoryClass);
      props.put("mail.smtp.socketFactory.port", mailSmtpSocketFactoryPort);
      javaMailSender.setJavaMailProperties(props);
      javaMailSender.setHost(mailHost);
      javaMailSender.setPort(mailPort);
      javaMailSender.setUsername(mailUsername);
      javaMailSender.setPassword(mailPassword);
      return javaMailSender;
   }

}
```

JpaMysqlConfiguration.java

```
package com.diplom.taskmanager.configuration;

import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.*;
import org.springframework.core.env.Environment;
import org.springframework.core.io.Resource;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.datasource.init.DataSourceInitializer;
import org.springframework.jdbc.datasource.init.DatabasePopulator;
import org.springframework.jdbc.datasource.init.ResourceDatabasePopulator;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.inject.Inject;
import javax.sql.DataSource;
import java.util.Properties;

@Configuration
@Profile("mysql")
@EnableTransactionManagement
@ComponentScan("com.diplom.taskmanager.entity")
@EnableJpaRepositories("com.diplom.taskmanager.repository")
@PropertySource(value = {"classpath:database/mysql/mysql.properties"})
public class JpaMysqlConfiguration {

    @Value("classpath:database/mysql/populate.sql")
    private Resource dataScript;

    private final Environment env;

    @Inject
    public JpaMysqlConfiguration(Environment env) {
        this.env = env;
    }

    @Bean
    public DataSource dataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setDriverClassName(env.getRequiredProperty("database.driver"));
```

```
        dataSource.setUrl(env.getRequiredProperty("database.url"));
        dataSource.setUsername(env.getRequiredProperty("database.password"));
        dataSource.setPassword(env.getRequiredProperty("database.username"));
        return dataSource;
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
        JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();
        factory.setDataSource(dataSource());
        factory.setPackagesToScan("com.diplo.taskmanager");
        factory.setJpaVendorAdapter(vendorAdapter);
        factory.setJpaProperties(additionalProperties());
        return factory;
    }
    @Bean
    public DataSourceInitializer dataSourceInitializer(DataSource dataSource) {
        DataSourceInitializer initializer = new DataSourceInitializer();
        initializer.setDataSource(dataSource);
        initializer.setDatabasePopulator(databasePopulator());
        return initializer;
    }
    private DatabasePopulator databasePopulator() {
        ResourceDatabasePopulator populator = new ResourceDatabasePopulator();
        populator.addScript(dataScript);
        return populator;
    }

}
```

SecurityConfiguration.java

```
package com.diplom.taskmanager.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
```

```
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.authentication.logout.HttpStatusReturningLogoutSuccessHandler;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

import javax.inject.Inject;
import java.util.HashMap;
import java.util.Map;

@Configuration
@EnableWebSecurity
@ComponentScan(basePackages = {"com.diplom.taskmanager.configuration.security"})
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;
    private final BCryptPasswordEncoder bCryptPasswordEncoder;
    private final RestUnauthorizedEntryPoint restAuthenticationEntryPoint;
    private final AccessDeniedHandler restAccessDeniedHandler;
    private final AuthenticationSuccessHandler restAuthenticationSuccessHandler;
    private final AuthenticationFailureHandler restAuthenticationFailureHandler;

    @Inject
    public SecurityConfiguration(UserDetailsService userDetailsService,
                                 BCryptPasswordEncoder bCryptPasswordEncoder,
                                 RestUnauthorizedEntryPoint restAuthenticationEntryPoint,
                                 AccessDeniedHandler restAccessDeniedHandler,
                                 AuthenticationSuccessHandler restAuthenticationSuccessHandler,
                                 AuthenticationFailureHandler restAuthenticationFailureHandler) {
        this.userDetailsService = userDetailsService;
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;
        this.restAuthenticationEntryPoint = restAuthenticationEntryPoint;
        this.restAccessDeniedHandler = restAccessDeniedHandler;
        this.restAuthenticationSuccessHandler = restAuthenticationSuccessHandler;
        this.restAuthenticationFailureHandler = restAuthenticationFailureHandler;
    }

    @Override
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
        Map<String, String[]> requireCsrfPatterns = new HashMap<>();

            .authorizeRequests()
                .antMatchers(HttpMethod.POST, "/api/login").anonymous()
                .antMatchers(HttpMethod.POST, "/api/users/").anonymous()
                .antMatchers(HttpMethod.PATCH, "/api/users").anonymous()
                .antMatchers("/api/users/**").hasAnyRole("USER", "ADMIN")
                .antMatchers("/api/tasks/**").hasAnyRole("USER", "ADMIN")
                .antMatchers("/**").permitAll()
                .and()
            .exceptionHandling()
                .authenticationEntryPoint(restAuthenticationEntryPoint)
                .accessDeniedHandler(restAccessDeniedHandler)
                .and()
            .formLogin()
                .loginProcessingUrl("/api/login")
                .successHandler(restAuthenticationSuccessHandler)
                .failureHandler(restAuthenticationFailureHandler)
                .usernameParameter("email")
                .passwordParameter("password")
                .and()
            .logout()
                .logoutRequestMatcher(new AntPathRequestMatcher("/api/logout","POST"))
                .logoutSuccessHandler(new HttpStatusReturningLogoutSuccessHandler())
                .deleteCookies("JSESSIONID");
    }
}
```