Чорноморський національний університет імені Петра Могили

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Вебзастосунок керування проєктами

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.22010815

Здобувач _____ Т. О. Нюргечев

підпис

«__» _____ 2024 р.

Керівник канд. техн. наук, доцент _____ Г. В. Горбань

підпис

«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«_22_» ____грудня___2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано здобувачу групи _408_ факультету комп'ютерних наук

_____Нюргечеву Тимуру Олексійовичу_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

_____Вебзастосунок керування проєктами_____

Затверджена наказом по ЧНУ від «22» _грудня_ 20_23_ р. №___269____

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

_____Вхідні дані – функціональні та нефункціональні вимоги до програмного забезпечення системи керування проєктами. Результат – функціонуючий вебзастосунок керування проєктами._____

4. Перелік питань, що підлягають розробці

– аналіз предметної області;

− огляд вебзастосунків із аналогічним функціоналом;

− формування вимог до системи;

− проєктування архітектури вебзастосунку;

− побудова бази даних проєкту;

− програмна реалізація, тестування та відлагодження вебзастосунку.

5. Перелік графічних матеріалів:

Презентація.

6. Завдання до спеціальної частини

Дослідження питань охорони праці, які безпосередньо пов’язані з діяльністю розробника програмного забезпечення.

7. Консультанти:

Консультант	Кафедра(організація)	Частина роботи
Алєксєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи канд. техн. наук, доцент Горбань Гліб Валентинович.

(посада, прізвище, ім’я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

Нюргечев Тимур Олексійович

(прізвище, ім’я, по батькові)

_____

(підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема:_____Вебзастосунок керування проєктами_____
_____
_____

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	04.12.2023	07.12.2023	Виконано
2.	Огляд навчальних матеріалів за темою роботи	15.01.2024	25.01.2024	Виконано
3.	Складання календарного плану КРБ	26.01.2024	28.01.2024	Виконано
4.	Аналіз предметної області	29.01.2024	10.02.2024	Виконано
5.	Моделювання та ПЗ	11.02.2024	22.02.2024	Виконано
6.	Розробка, тестування ПЗ, формування посібнику користувача	23.02.2024	03.04.2024	Виконано
7.	Опис розділу з охорони праці	04.04.2024	24.04.2024	Виконано
8.	Відгук керівника КРБ	25.04.2024	27.04.2024	Виконано
9.	Оформлення КРБ та презентації	28.04.2024	10.05.2024	Виконано
10.	Попередній захист	03.06.2024	05.06.2024	Виконано
11.	Рецензування	15.06.2024	18.06.2024	Виконано
12.	Завершення оформлення КРБ та презентації	19.06.2024	20.06.2024	Виконано
13.	Захист кваліфікаційної роботи	24.06.2024	27.06.2024	Виконано

Розробив здобувач _____Нюргечев Т. О._____ _____
(прізвище, ім’я, по батькові) (підпис)
«28» _січня__ 2024 р.

Керівник роботи _____канд. техн. наук, доцент Горбань Г. В. _____
(посада, прізвище, ім’я, по батькові) (підпис)

«28» _січня__ 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Вебзастосунок керування проєктами»

Здобувач 408 гр.: Нюргечев Тимур Олексійович

Керівник: викладач кафедри ІПЗ, канд. техн. наук, доцент Горбань Г. В.

Тема кваліфікаційної роботи є актуальною, оскільки вебзастосунки керування проєктами є невід'ємною складовою при розробці програмного забезпечення, над створенням або модернізацією якого працює команда. Для повної реалізації потенціалу вебзастосунку важливо інтегрувати в нього інструменти для аналізу діяльності команд та ефективність рішень, що застосовуються на рівень нижче, аніж власне проєкт.

Об'єкт кваліфікаційної роботи – процес моделювання та створення вебзастосунку керування проєктами.

Предмет кваліфікаційної роботи – вебзастосунок керування проєктами.

Мета кваліфікаційної роботи – створення вебзастосунку керування проєктами із відповідними інструментами для збору та аналізу статистики для вдосконалення можливостей використання систем керування проєктами для саморефлексії.

У першому розділі сформовано вимоги до системи шляхом аналізу предметної області та аналогічних систем, у другому здійснено проєктування та моделювання системи, сформовано сценарії використання, у третьому оглянуто стек задіяних технологій та здійснено програмну реалізацію проєкту, а у четвертому оглянуто інтерфейси вебзастосунку, їх дизайн та функціонал.

В результаті виконання кваліфікаційної роботи бакалавра було реалізовано вебзастосунок керування проєктами. КРБ викладена на 65 сторінки, вона містить 4 розділи, 38 ілюстрацій, 11 таблиць, 20 джерел в переліку посилань.

Ключові слова: *вебзастосунок, система керування проєктами, Spring Framework, аналітика.*

ABSTRACT

of the Bachelor’s Thesis

"Project management web application"

Student of group 408: Niurhechev Tymur Oleksiiovych

Supervisor: Lecturer of the Department of SE, Ph.D. Horban H. V.

The thesis’s topic is relevant because web-based project management applications are an integral part of software development, creation or modernization of this team's work. To fully implement the potential of a web application, it is important to integrate tools for analyzing team's activities and the effectiveness of decisions that are applied at a level lower than the project itself.

The object of the thesis is the process of designing a project management web application.

Subject of the thesis is a project management web application.

The purpose of the thesis is creating a project management web application with appropriate tools for collecting and analyzing statistics to improve the possibilities of using project management systems for self-reflection, increasing productivity and selecting effective work practices.

The first section presents an analysis of system requirements, which includes a description of the subject area, analysis of similar systems, problem statement and use case scenarios. The second section is dedicated to designing and modeling the system. In the third section, the stack of involved technologies is reviewed and the software implementation of the project is carried out. The fourth secion reviews web application interfaces, their design and functionality.

As a result of the bachelor's qualification work, a project management web application was developed. The BT is set out 65 on pages, it contains 4 sections, 38 illustrations, 11 tables, 20 sources.

Keywords: *web application, project management system, Spring Framework, analytics.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

СКБД – система керування базами даних

СУП – система управління проєктами

API – Application Programming Interface

CSS – Cascading Style Sheets

HTML – Hypertext Markup Language

JPA – Java Persistence API

JS – JavaScript

MVC – Model-View-Controller

ORM – Object-Relational Mapping

POJO – Plain old Java object

SQL – Structured Query Language

UI – User interface

UML – Unified Modeling Language

ВСТУП

Комунікація, напевно, є найважливішою складовою успіху у командній роботі. Важко переоцінити, який вплив вона має на ефективність, якість та дієвість розроблюваних рішень та наскільки проблематичною може бути її відсутність. Швидкий обмін інформацією, чітка делегація завдань, звітність та можливість швидко підлаштовуватися під будь-яку зміну планів – кожна з цих практик сьогодні застосовується у групових проєктах, зокрема і при розробці програмного забезпечення.

Сфера IT, будучи технологічно прогресивною та часто залучаючи до роботи робітників, що працюють віддалено, однією з перших почала масово застосовувати інструменти, що дозволяли б усім учасникам швидко та зрозуміло комунікувати про свій прогрес, проблеми або питання. На сьогоднішній день найбільш популярним з них є канбан – методологія розробки ПЗ, для візуалізації якої найчастіше використовують інтерактивні дошки, що відображають прогрес при роботі над певним проєктом. Ця методологія зарекомендувала себе як ефективний засіб для відслідковування виконання поставлених задач і уникнення перенавантаження окремих членів команди. Проте чи слід системам управління проєктами цим обмежуватися?

Не дивлячись на те, наскільки популярними є подібні системи, більшість з них представляє з себе доволі прості і невибагливі інструменти, що зазвичай не пропонують нічого, окрім безпосередньо відслідковування прогресу проєктів. З одного боку, це не перевантажує інтерфейс та дає змогу швидко освоїти фунціонал вебзастосунків. З іншого, це не дозволяє командам ефективно аналізувати виконані проєкти та оптимізувати свою роботи на рівні завдань.

Тема кваліфікаційної роботи є **актуальною**, оскільки вебзастосунки керування проєктами є невід’ємною складовою при розробці програмного забезпечення, над створенням або модернізацією якого працює команда. Для повної реалізації потенціалу вебзастосунку важливо інтегрувати в нього

інструменти для аналізу діяльності команд та ефективність рішень, що застосовуються на рівень нижче, аніж власне проєкт.

Об’єкт кваліфікаційної роботи – процес моделювання та створення вебзастосунку керування проєктами.

Предмет кваліфікаційної роботи – вебзастосунок керування проєктами.

Метою роботи є створення вебзастосунку керування проєктами із відповідними інструментами для збору та аналізу статистики для вдосконалення можливостей використання систем керування проєктами для саморефлексії, підвищення продуктивності та підбору ефективних робочих практик.

Для досягнення поставленої мети необхідно виконати **наступні завдання:**

– аналіз предметної області;

– розробка вимог до застосунку на основі проведеного аналізу

– проєктування архітектури вебзастосунку керування проєктами

– проєктування бази даних

– розробка, тестування та розгортання вебзастосунку

Для виконання поставлених завдань використано системи контролю базами даних MariaDB, а також фреймворк Spring Framework мови програмування Java.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМ КЕРУВАННЯ ПРОЄКТАМИ

1.1 Опис предметної області

Системи управління проєктами – це програмні рішення, які допомагають командам ефективно планувати, виконувати та контролювати проєкти [9]. Вони забезпечують інструменти для організації завдань, управління ресурсами, визначення термінів та моніторингу прогресу. Основні функції систем управління проєктами включають:

1) планування проєктів: системи управління проєктами дозволяють створювати докладні плани, визначати завдання, ресурси, терміни виконання та залежності між ними [11];

2) організація завдань: вони дозволяють створювати список завдань, призначати їх відповідальним особам, встановлювати пріоритети та визначати строки виконання;

3) керування ресурсами: системи управління проєктами допомагають розподілити ресурси (людські, фінансові, матеріальні) між різними завданнями та визначити їх доступність у часі [13];

4) моніторинг і контроль: вони надають засоби для відстеження прогресу проєкту, виявлення затримок та вирішення проблемних ситуацій;

5) звітність і аналіз: системи управління проєктами забезпечують можливість створення звітів про прогрес, витрати, ризики та інші показники для аналізу ефективності проєкту;

6) спільна робота та комунікація: Багато СУП мають функції спільної роботи, що дозволяють командам спілкуватися, обмінюватися інформацією та спільно працювати над завданнями;

7) управління ризиками і змінами: вони допомагають ідентифікувати потенційні ризики проєкту та керувати змінами в ході його виконання [6];

8) інтеграція з іншими системами: окремі застосунки керування проєктами мають можливість інтеграції з іншими інструментами та платформами, що полегшує обмін даними та співпрацю з іншими програмами.

Серед недоліків таких систем часто зустрічаються:

1) чимала кількість функціонала є платною;
2) відсутність аналітичних інструментів;
3) невелика масштабованість проєктів;
4) застарілий інтерфейс.

1.2 Аналіз аналогів

Для формування вимог до системи слід проаналізувати аналогічні продукти. Це допоможе досягти одразу декількох цілей:

1) Визначити, які потреби мають користувачі та за яким принципом вони обирають інструменти для управління проєктами. Це допоможе зрозуміти, який вже існуючий функціонал точно необхідно буде імплементувати у вебзастосунку.

2) Виявити сильні та слабкі сторони наявних аналогів, аби уявити їх вагомість та потреби у внесенні змін до власного застосунку. Це дозволить сформувати вимоги, що визначатимуть переваги вебзастосунку над конкурентами.

3) Визначити, за яким призначенням найчастіше використовуються аналоги, аби зробити застосунок зручним для використання і не наповнювати його зайвим функціоналом.

Перейдемо до конкретних прикладів вебзастосунків, що містять функціонал для управління проєктами:

1) trello.com [1]

Вебзастосунок trello.com є найпопулярнішою системою керування проєктами. Створений 2011 року, він миттєво привернув до себе увагу ефективний інструмент для організації роботи у команді. За час свого

існування він суттєво розширив свій функціонал з дошки канбан до можливості трекінгу та аналізу проєктів, хоча подібні розширення не є вбудованими і підходять скоріше для сортування та планування. Окрім цього, trello має дуже поверхневу реалізацію звітності. Має дуже зручний, невибагливий інтерфейс. Підтримує інтеграцію із іншими системами керування проєктами.

Таблиця 1.1 – Характеристики Trello

Назва	Trello
Виробник	Atlassian
Архітектура	Вебзастосунок та мобільний додаток
Мова реалізації	JavaScript(jQuery/node.js)
Основні функції	Створення канбан-дошок для керування проєктами, планування, трекінг проєктів, статистичний аналіз, базова звітність
Переваги	Популярність, зручний інтерфейс, велика кількість розширень
Недоліки	Велика кількість преміум-функцій, відсутність поглибленого аналізу та звітності
Джерело інформації	trello.com

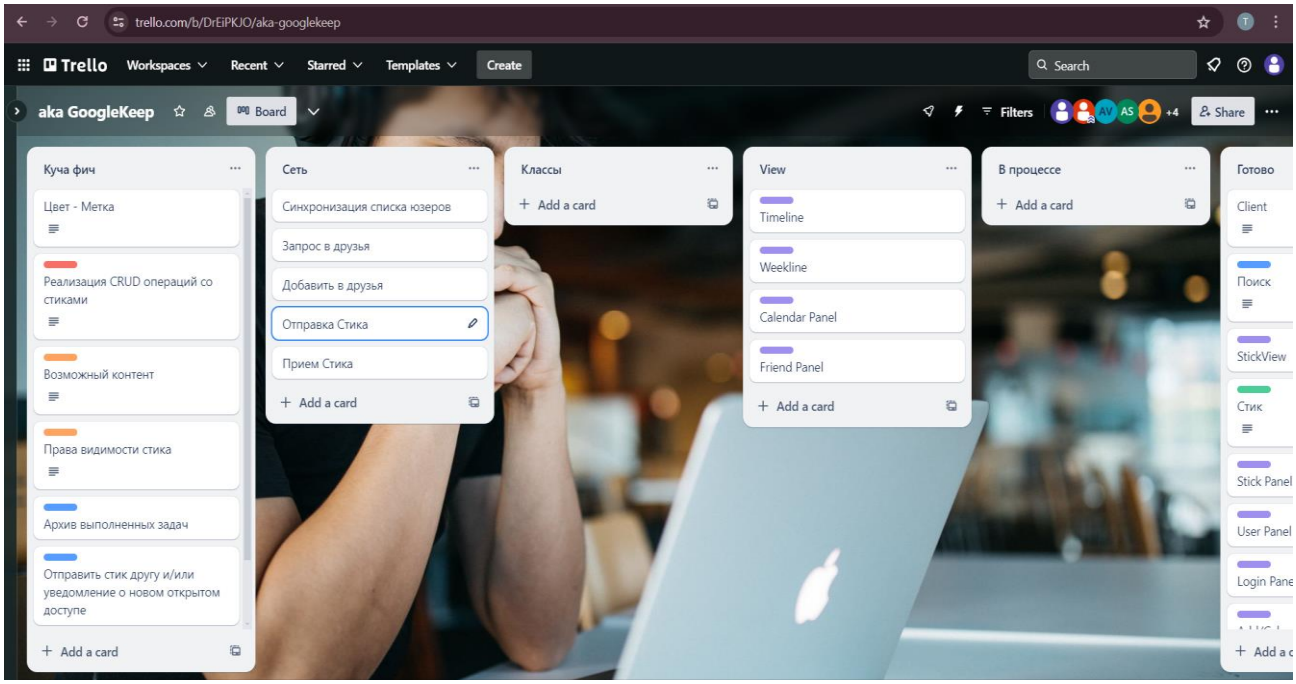

Рисунок 1.1 – Приклад активної дошки у Trello

2) Monday.com [2]

Monday.com – це веб-платформа для управління проєктами та спільної роботи, яка надає широкий спектр інструментів для планування, організації завдань, комунікації та моніторингу прогресу проєктів. На відміну від trello, він пропонує ширшу спеціалізацію, зокрема, керування проєктами у сферах маркетингу, HR тощо. Окрім цього, як і Trello, Monday також має мобільну версію у вигляді додатку.

Таблиця 1.2 – Характеристики Monday

Назва	Monday
Виробник	Monday.com Ltd.
Архітектура	Вебзастосунок та мобільний додаток
Мова реалізації	JavaScript, React
Основні функції	Створення todo-списків, планування, організація завдань, моніторинг прогресу, розширені засоби комунікації
Переваги	Зручний інтерфейс, покращені засоби для аналітики, розширена спеціалізація(маркетинг, HR тощо)
Недоліки	Незручний інтерфейс, погана оптимізація
Джерело інформації	Monday.com

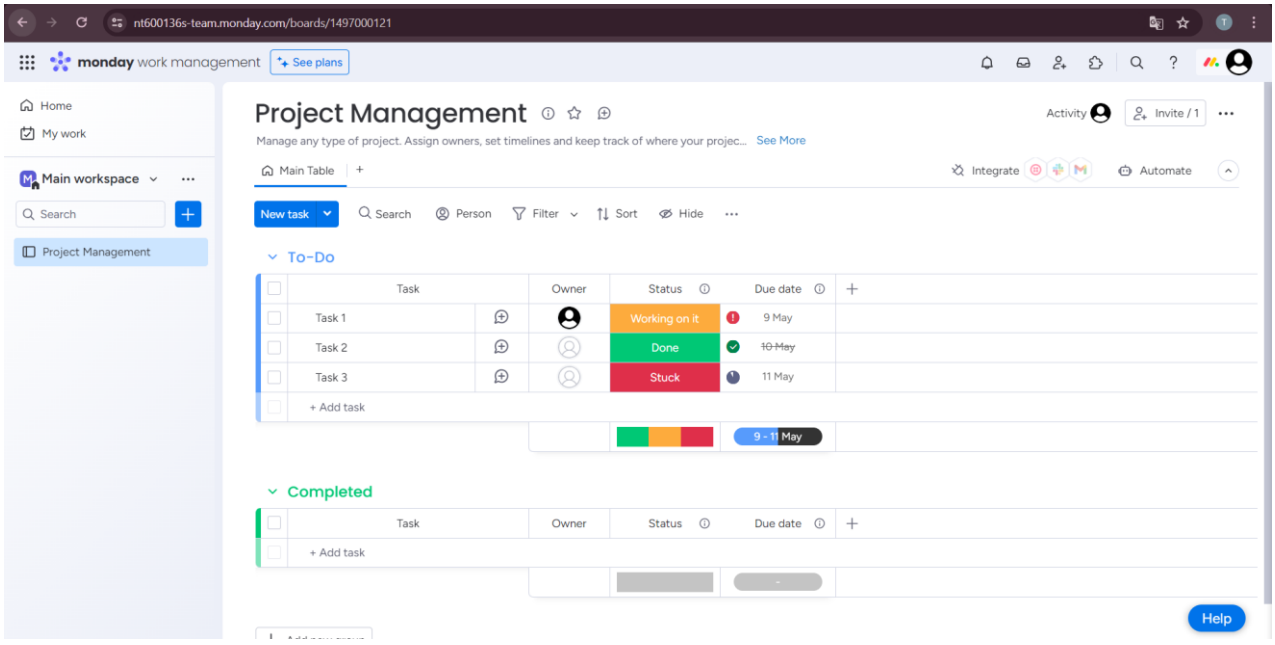

Рисунок 1.2 – Приклад новоствореного проєкту у Monday

3) Clickup.com [3]

Clickup.com – вебзастосунок керування проєктами, що відрізняється розширеним функціоналом операцій на дошці. Має функції імпорту дошок з Trello та Jira і найчастіше застосовується тоді, коли їх функціоналу недостатньо. Являє собою дуже спеціалізований інструмент, доволі важкий для швидкого опанування. Містить чимало засобів для аналітики.

Таблиця 1.3 – Характеристики ClickUp

Назва	ClickUp
Виробник	Приватна компанія ClickUp
Архітектура	Вебзастосунок та мобільний додаток
Мова реалізації	PHP, jQuery
Основні функції	Створення todo-списків для керування проєктами, планування, тренінг проєктів, поглиблений статистичний аналіз, розширена звітність
Переваги	Дуже екстенсивні інструменти аналітики, можливість експорту дошок з trello, велика кількість розширень
Недоліки	Велика кількість преміум-функцій, складний для осягнення інтерфейс
Джерело інформації	https://help.clickup.com/hc/en-us

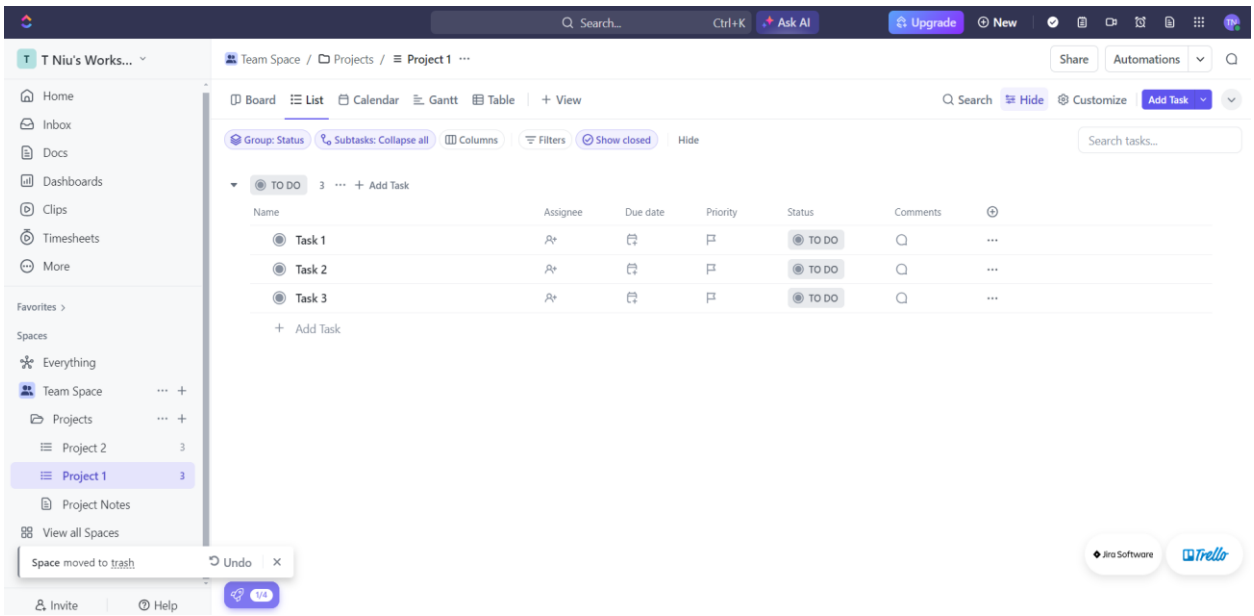

Рисунок 1.3 – Приклад новоствореного проєкту у Clickup

Загалом, можна відзначити, що усі аналоги мають інструментарій для відслідковування прогресу виконання командної роботи. Найчастіше бракує аналітичних функцій, що були б напрямлені на визначення ефективності роботи та порівняння із минулими проєктами. Також слід відзначити використання у окремих аналогів елементів інтерфейсу, що є малоефективними та суттєво уповільнюють роботу із проєктом (наприклад, використання кнопок для зміни стану завдання або ж розподілення стовпців у окремі таблиці). В робочих набагато зручнішим був простий, але ефективний інтерфейс, де можна швидко змінювати стан дошки.

1.3 Специфікація вимог

Спираючись на проведений аналіз предметної області та огляд аналогічних вебзастосунків, можна сформувати вимоги до ПЗ. У майбутньому, за допомогою цих вимог можна конкретизувати потрібний функціонал, що має реалізовувати вебзастосунок та характеристики, яким він має відповідати.

Характеристики користувачів

У системі повинні бути передбаченні наступні ролі користувачів з різними правами доступу до функцій системи:

1) користувач – роль з обмеженими правами доступу до функцій системи;

2) адміністратор – роль з усіма правами доступу у системі. Аби убезпечити вебзастосунок від можливого несанкціонованого внесення змін, можливість створення нових облікових записів із правами адміністратора мають лише власне адміністратори;

3) незареєстрований користувач – не має доступу до функціоналу сайту.

Загальна структура і склад системи

Система має представляти з себе вебзастосунок, що матиме два окремі інтерфейси для користувача та адміністратора із розподіленим функціоналом.

Архітектура застосунку має складатися із серверної частини, бази даних та користувацького інтерфейсу.

Загальні обмеження

Обмеженням функціонування вебзастосунку є наявність Інтернет-з’єднання та створений обліковий запис. Оскільки головною частиною функціоналу є взаємодія між користувачами на одній дошці із прив’язкою до їх облікових записів, потреби реалізації частини функціоналу для незареєстрованих користувачів немає.

Функції системи

Конкретизуємо функціонал системи шляхом створення функціональних вимог:

1) створення, редагування та видалення облікових записів;
2) створення, редагування та видалення дошок;
3) додавання облікових записів декількох користувачів до однієї дошки;
4) авторизація користувачів;
5) створення звітів про активність;
6) аналітика проєктів;
7) планування виконання завдань;
8) розрахунок бюджету;
9) комунікація між розробниками;
10) встановлення пріоритетності завдань.

Властивості програмного забезпечення

Доступність

Система має бути доступною цілодобово на будь-яких пристроях, що мають підтримуваний системою браузер, що використовує протокол http.

Продуктивність

Час від надходження запиту обслуговування клієнта до віддачі контенту клієнту сервісом (тобто. загальний час обробки запиту на серверної частини) має становити не більше 5 секунд при нормальному навантаженні обладнання.

Надійність

Система має зберігати поточний стан облікових записів та дошок у разі екстреного вимкнення, а також попереджати користувачів про несправності у разі їх виникнення та запобігати небажаним у таких ситуаціях діям.

Безпека

Користувацькі дані слід зберігати у зашифрованому вигляді. Інтерфейс для взаємодії із системою та його функціонал мають відрізнятися для користувача та адміністратора.

Висновки до розділу 1

У розділі 1 було виконано аналіз предметної області. З’ясовано, що таке система керування проєктами, її призначення та загальний функціонал. Визначено вебзастосунок як найкращий варіант імплементації системи, виходячи із наявних потреб та ресурсів.

Оглянуто наявні аналоги, їх функціонал, виокремлено їх переваги та недоліки. Проаналізовано користувацький інтерфейс аналогів та технології, задіяні у їх розробці. На основі проведеного аналізу сформовано технічне завдання. Зокрема, потреба у розробці системи із ширшим набором інструментів для аналізу та звітності, зручнішим інтерфейсом та можливістю інтеграції з іншими сервісами.

Сформовано функціональні вимоги до системи. Визначено властивості програмного забезпечення. Описано загальну структуру та склад системи, сформовано обмеження.

У результаті проведеної роботи створено ТЗ та розроблено специфікацію вимог.

2 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ СИСТЕМИ

Для конкретизації функцій ПЗ слід почати із проєктування програмного забезпечення. Це дозволить візуалізувати структуру вебзастосунку і полегшити подальший процес його створення.

2.1 Створення сценаріїв використання

Сценарій №1 (коротка форма): Авторизація

Користувач заходить на головну сторінку сайту. У випадку відсутності облікового запису проходить реєстрацію. Система зберігає новий обліковий запис із правами користувача до бази даних. Користувач повертається га головну сторінку та входить до свого облікового запису.

Сценарій №2 (поверхнева форма): Створення дошки

Головний сценарій:

− користувач обирає створення нової дошки;

− користувач редагує ім’я та тип дошки;

− користувач додає до дошки користувачів (за потреби);

− користувач створює картки (за потреби);

– користувач обирає інструменти аналітики (за потреби);

− користувач підтверджує створення дошки;

− дошка проходить автоматичну модерацію;

− система створює запис у БД та зберігає поточний стан дошки.

Альтернативні сценарії:

– збій у роботі системи, користувач отримає відповідне повідомлення, створення дошки неможливе;

– певні атрибути дошки (ім’я, назви карток тощо) не пройшли модерацію, створення дошки неможливе.

Сценарій №3 (повна форма): створення облікового запису адміністратора

Scope: система керування проєктами

Level: user-goal

Primary Actor: адміністратор

Stakeholders and interests:

1) чинний адміністратор: зацікавлений в зменшенні навантаження на роботі за рахунок збільшення штату адміністраторів

2) новий адміністратор: зацікавлений в якнайшвидшому отриманні інструментів адміністрування системи для початку роботи.

3) користувач: зацікавлений у пришвидшенні обробки запитів до служби підтримки.

Preconditions: адміністратор увійшов до облікового запису з правами адміністратора.

Main Success Secnario:

1) адміністратор обирає опцію створення нового облікового запису адміністратора;

2) відкривається форма заповнення даних;

3) адміністратор вводить узгоджену із новим працівником адресу його електронної скриньки;

4) адміністратор додає пароль для акаунту;

5) адміністратор повторює пароль у полі підтвердження;

6) адміністратори звіряють дані;

7) адміністратор відправляє дані на сервер;

8) система звіряє паролі та чи вже існує електронна пошта в записах;

9) система додає новий акаунт;

10) адміністратор переходить на сторінку реєстрації.

Extensions:

3a) У нового працівника немає електронної скриньки:

1) працівник створює електронну скриньку;

2) працівник отримує у своє користування існуючий акаунт компанії;

3b) Працівник забув свою адресу:

1) працівник створює електронну скриньку;

2) працівник отримує у своє користування існуючий акаунт компанії.

8a) На електронну адресу вже зареєстровано акаунт:

1) новий адміністратор надає іншу електронну адресу;

2) працівник видаляє свій минулий акаунт;

3) працівник створює електронну скриньку;

4) працівник отримує у своє користування існуючий акаунт компанії.

8b) Паролі не збігаються:

1) форма заповнюється повторно.

9a) Стався збій системи

1) форма заповнюється повторно.

2) система перезавантажується.

Special requirements:

1) пристрій, що має браузер підтримуваний системою.

2) система у разі перевантаження повинна повідомляти про тимчасові труднощі протягом 30 секунд.

Technology and Data Variations List:

1) адміністратор користується інтерфейсом адміністратора для використання відповідних функцій;

2) система може використовувати декілька методів валідації (jQuery, Spring Data, JPA, серверний код);

3) інформація про помилку повертається за допомогою jQuery або повернення виключення з сервера.

Frequency of occurrence

Щоразу, коли треба надати інструменти адміністрування новому адміністраторові.

Miscellaneous (open issues):

1) створити можливість генерації робочої поштової адреси під час реєстрації.

2) вивчити можливість відновлення відправлених даних після збою.

3) як узгодити акаунти користувача та адміністратора при наявності у робітника лише однієї скриньки?

4) чи може адміністратор створювати безліч облікових записів з розширеними правами?

Візуалізуємо сценарії використання за допомогою діаграм прецедентів.

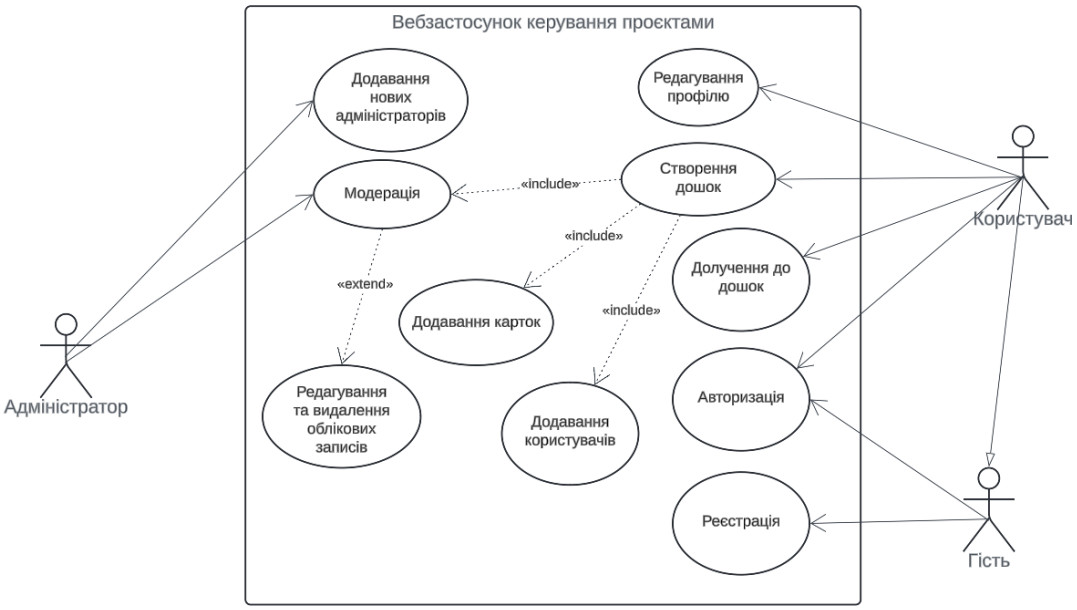

Рисунок 2.1 – Загальна діаграма прецедентів

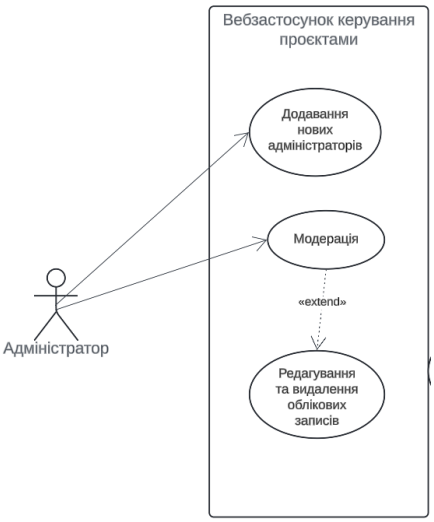

Рисунок 2.2 – Діаграма прецедентів для адміністратора

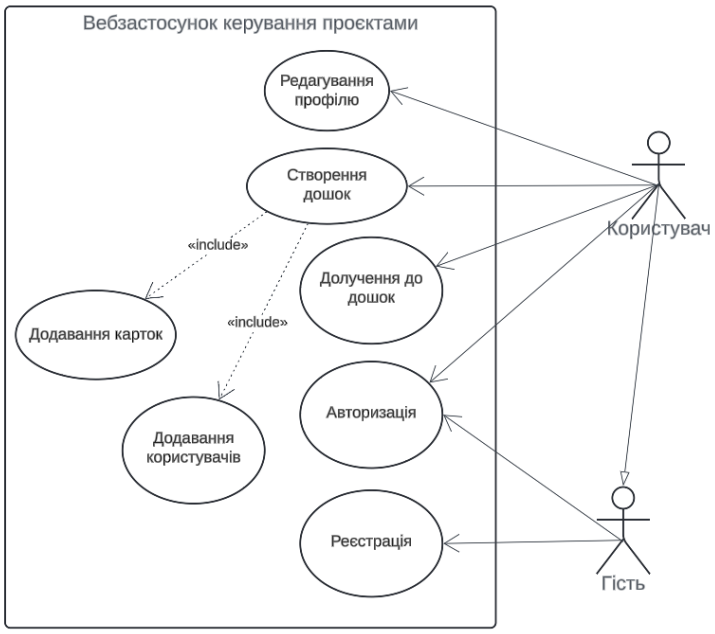

Рисунок 2.3 – Діаграма прецедентів для користувача

На діаграмах зображено взаємодію різних акторів із системою. За допомогою діаграми прецедентів також можна сформувати логічні зв’язки між операціями, що дозволить побудувати краще структурований проєкт у майбутньому.

2.2 Побудова діаграм взаємодії

Для відображення взаємодії логічних елементів системи між собою скористаємось діаграмами відображень. Це дозволить правильно розподілити функціонал ПЗ та делегувати частинам вебзастосунку лише задачі, відповідні їх спеціалізації, а також упевнитися у правильності виконання сценаріїв використання та наданих акторам прав, візуалізуючи взаємодію між компонентами системи через передачу даних, виклик методів та інші процеси взаємодії.

Почнемо з діаграми створення дошки. На ній вкажемо усі можливі сценарії при операції створення, зокрема, використання некоректного імені або відправка у якості нього пустого рядка.

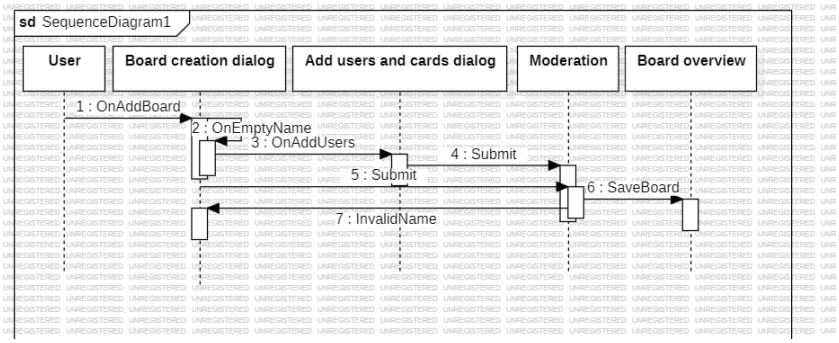

Рисунок 2.4 – Діаграма взаємодії створення дошки

Далі додамо діаграму взаємодії для операції авторизації. Аналогічно вкажемо усі можливі сценарії.

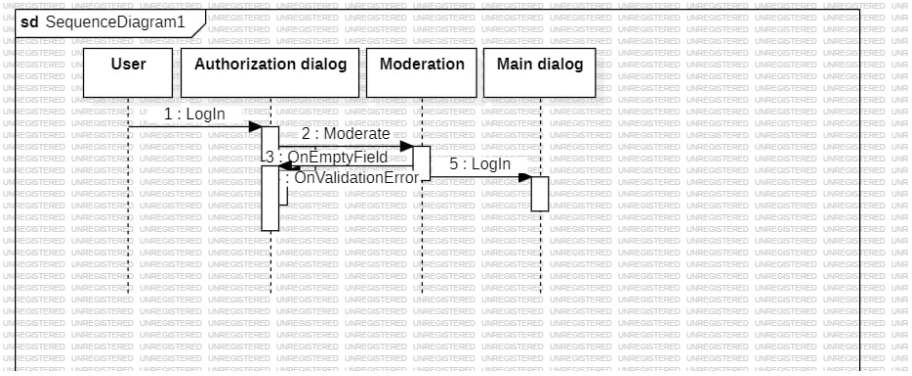

Рисунок 2.5 – Діаграма взаємодії авторизації

Наостанок, створимо діаграму для опису процесу додавання завдання. Розподілимо його на декілька етапів для кращого розуміння, як обробляти усі можливі сценарії.

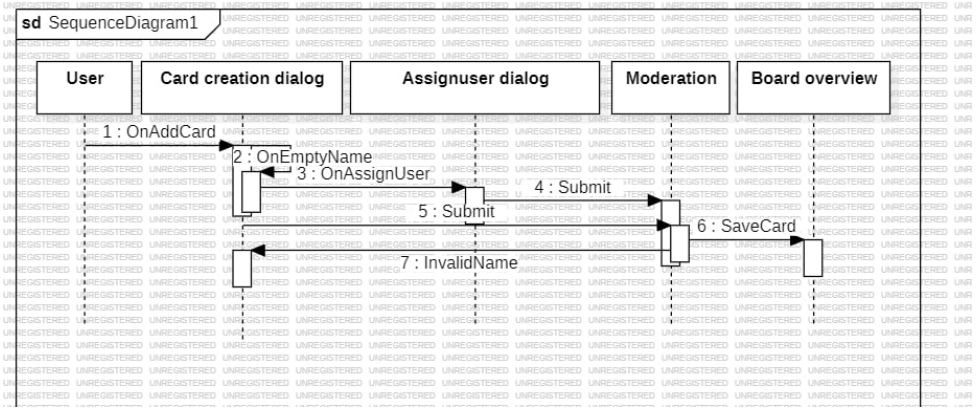

Рисунок 2.6 – Діаграма взаємодії додавання завдання

У результаті створення діаграм взаємодії, маємо зручну візуалізацію логічних компонентів програми для окремих завдань. Це дозволить краще та лаконічно описати програмну реалізацію, не вдаючись до зайвих дій.

2.3 Побудова діаграм станів

Для відображення зміни станів програми створимо діаграми станів. Вони демонструють стадії роботи вебзастосунку. На цих діаграмах зображується послідовність станів, через які проходить система, події, що викликають ці стани, а також переходи між станами. Діаграми станів допомагають візуалізувати поведінку вебзастосунку під час його взаємодії з користувачем та внутрішніми компонентами, що полегшує розуміння та налагодження логіки роботи програми.

Почнемо з діаграми створення нової дошки. Діаграма станів стане в нагоді при описі цього процесу, оскільки він має багато подій та умов, що їх викликають. Слід визначити, які з них є обов’язковими, а які вибірковими. За допомогою діаграми станів це легко відслідкувати перевіривши, чи можна не виконуючи певну умови перейти у потрібний стан. На діаграмі показано початковий стан, який переходить у стан "waiting for name". Якщо ім'я невірне, воно повертається до цього стану. Інакше, подія onAddUsers переводить до "add users". Якщо виникає помилка бази даних, система переходить до "saving board". В іншому випадку подія onAddCard переводить до "add default cards". Далі, після успішної модерації (onModerationSuccess), система переходить до "submit board". Подія onSubmit переводить до стану "moderation". Стан "saving board" є кінцевим станом.

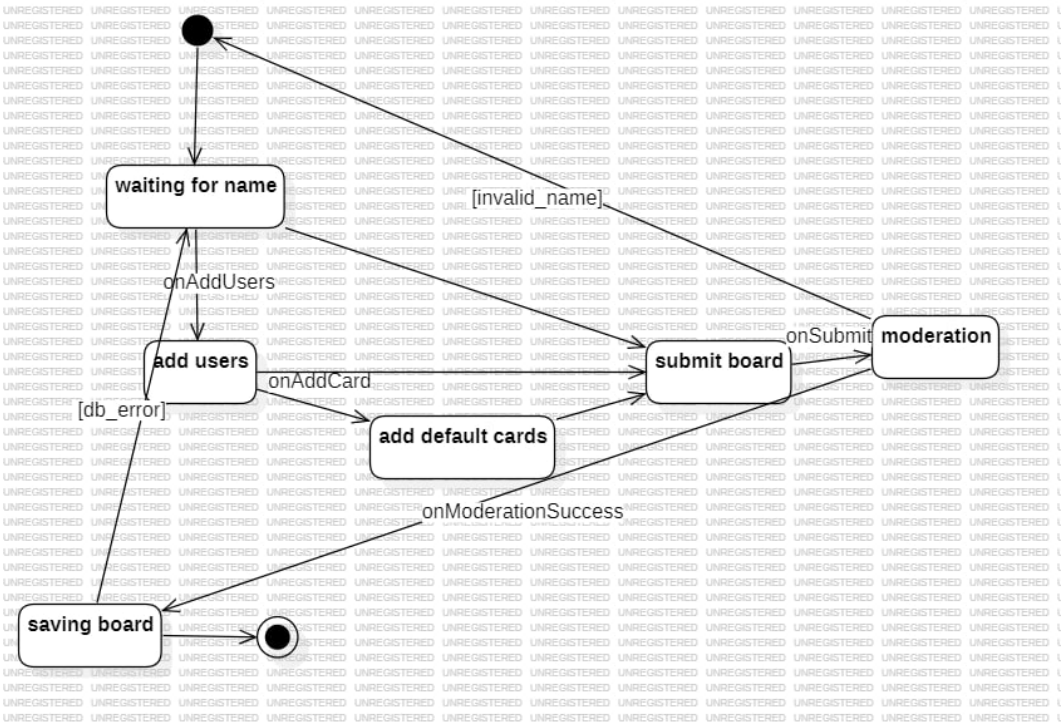

Рисунок 2.7 – Діаграма станів для створення канбан-дошки

Також опишемо процес додавання карток на дошку. Це дозволить правильно в майбутньому визначити послідовність кроків при виконанні даної операції.

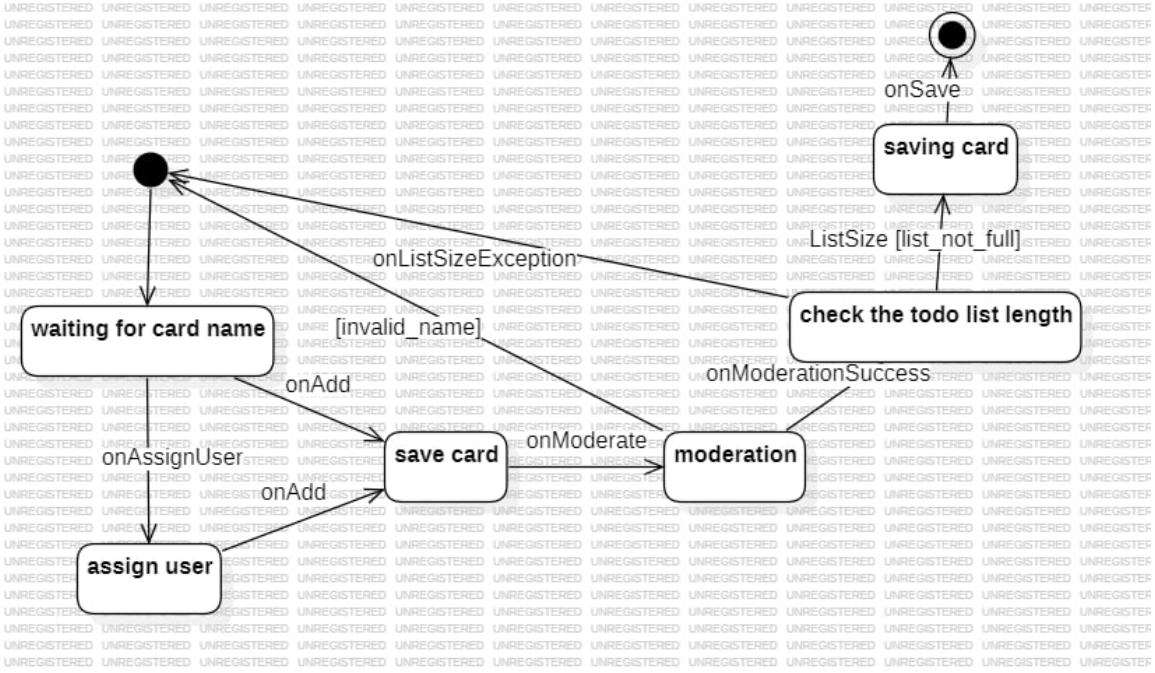

Рисунок 2.8 – Діаграма станів для додавання карток на дошку

Нарешті, додамо загальну діаграму станів для проєкту. На ній можна побачити чітке розділення функціоналу між користувачами із різними ролями та інструмент, що для цього застосовується (користувацький інтерфейс).

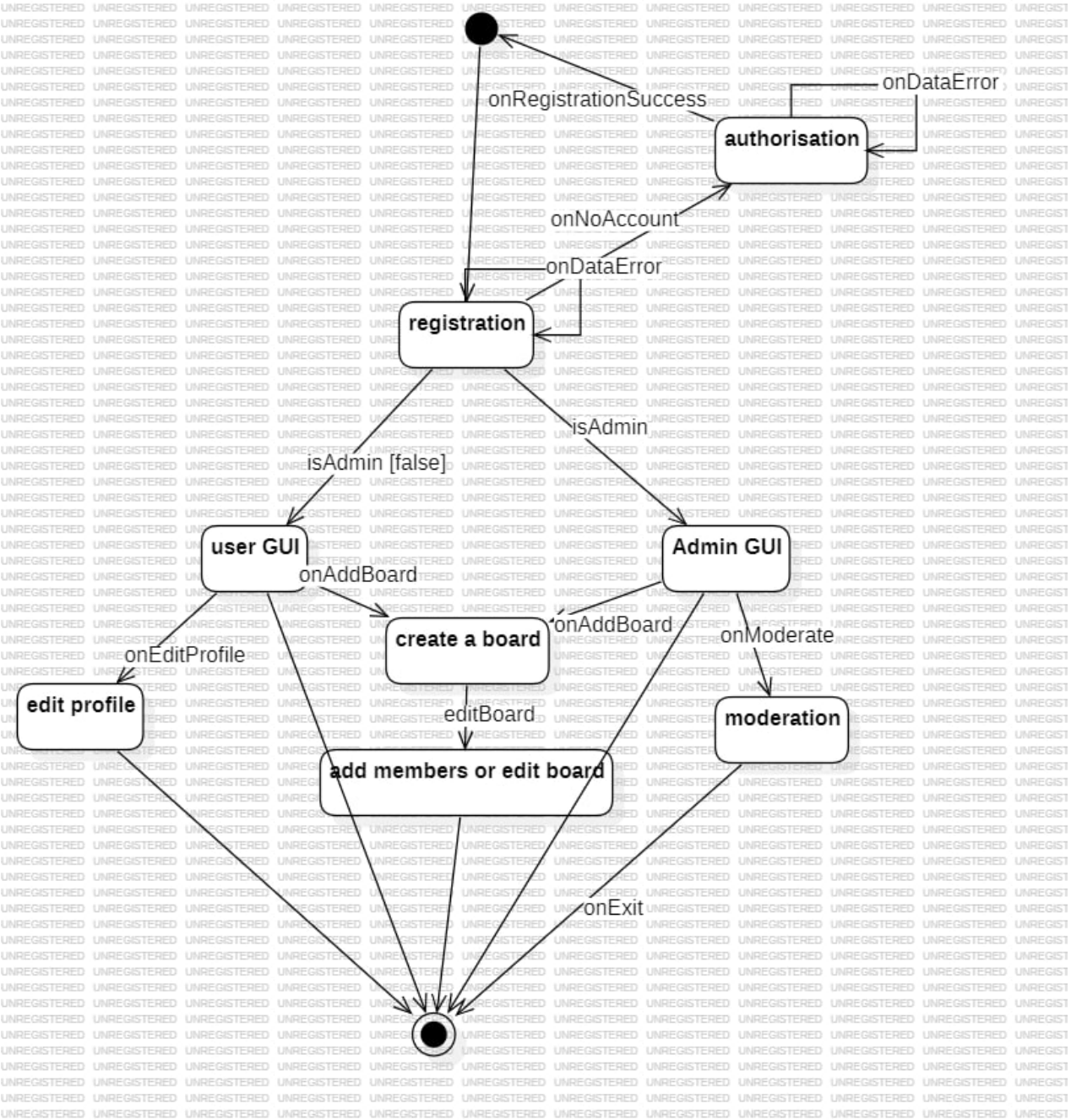

Рисунок 2.9 – Діаграма станів проєкту

У результаті побудови діаграм станів виділено події та умови, що сприяють зміні станів як програми, так і окремих її частин.

2.4 Створення діаграм діяльності

Для представлення стадій виконання програми додано діаграми діяльності, які ілюструють послідовність дій і процесів, що виконуються в системі, включаючи потоки управління між різними етапами та рішеннями. На цих діаграмах зображуються початкові та кінцеві точки процесів, дії, що виконуються, умови переходів між діями, а також можливі розгалуження та паралельні потоки. Це дозволяє візуально представити логіку виконання та взаємодію компонентів програми, спрощуючи аналіз і оптимізацію процесів.

Почнемо зі створення діаграми, що описує процес створення дошки. На ній можна побачити, як система взаємодіє із різними акторами та яку відповідальність покладено в межах процесу на адміністратора.

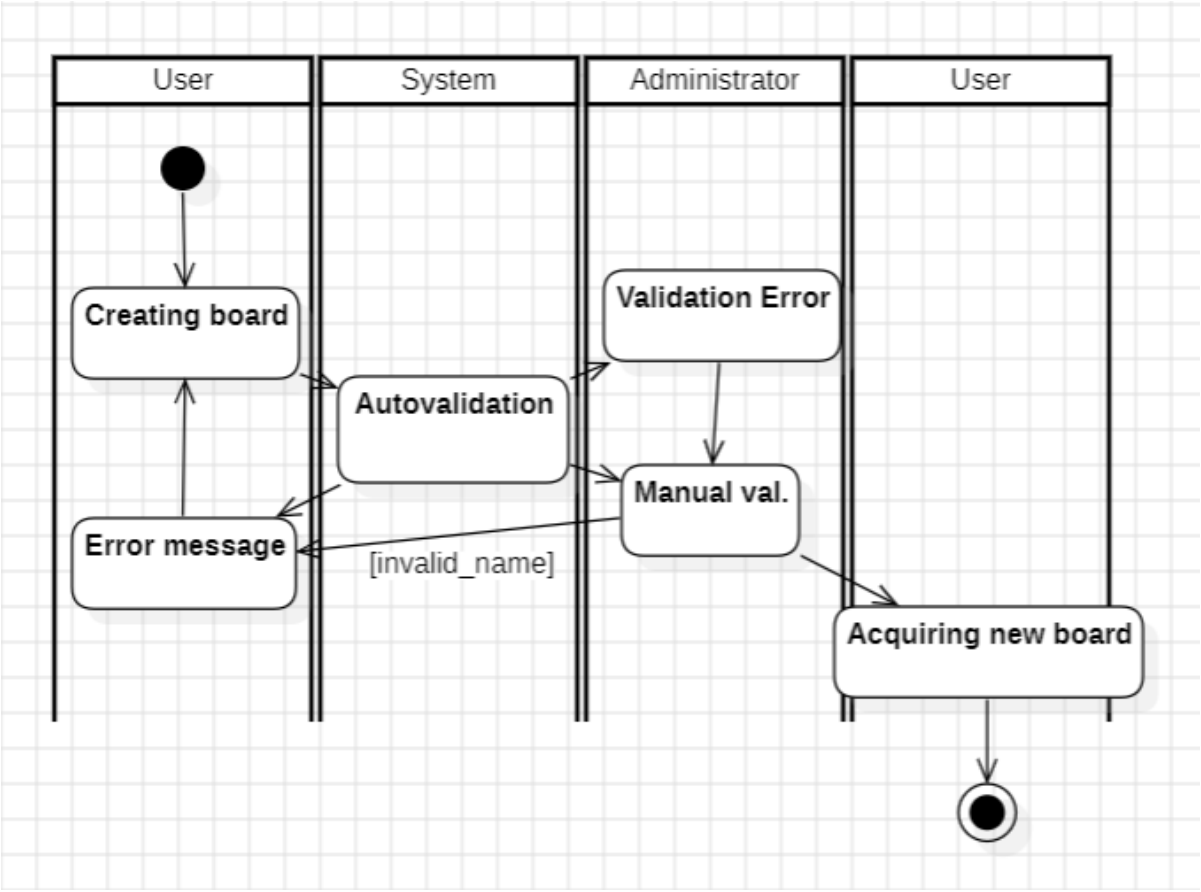

Рисунок 2.10 – Діаграма діяльності створення дошки

Тепер додамо діаграму діяльності тестування системи. Важливо прорахувати усі можливі сценарії при тестуванні для запобігання потенційним помилкам у роботі системи.

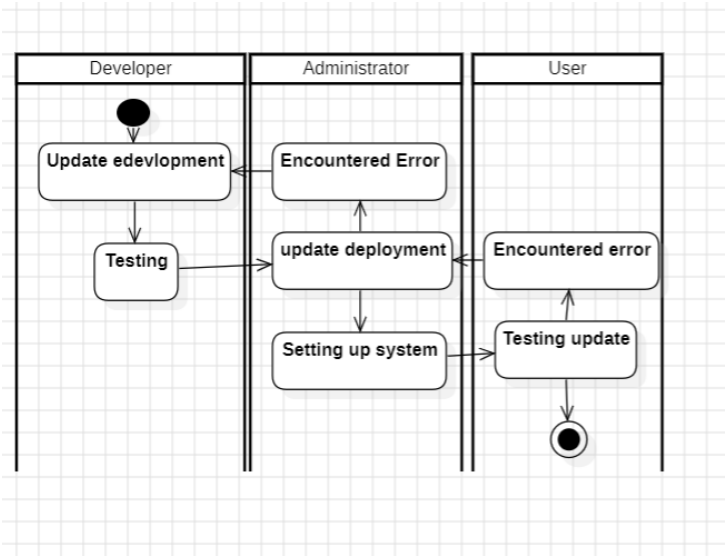

Рисунок 2.11 – Діаграма діяльності тестування системи

Також за допомогою діаграми діяльності опишемо обробку помилок у разі, якщо вони все ж виникатимуть.

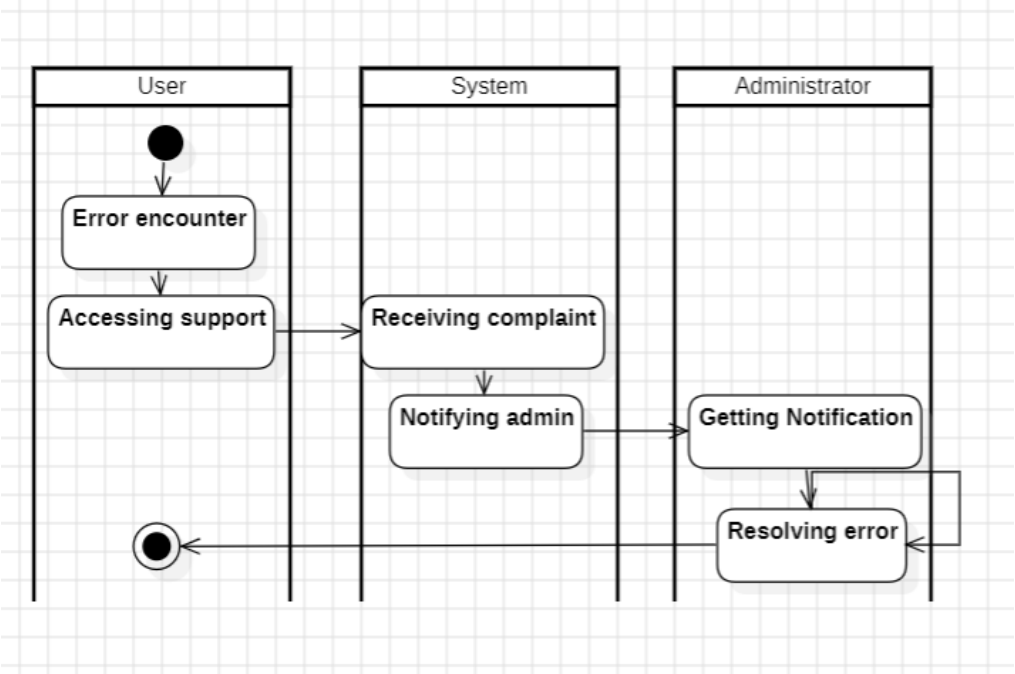

Рисунок 2.12 – Діаграма діяльності обробки помилок

За допомогою побудованих можна оптимізувати програмну реалізацію окремих завдань, краще розподіливши їх між акторами та перешкоджаючи зайвим переходам між ними.

2.5 Проєктування розгортання вебзастосунку

Після створення діаграм для опису роботи власне системи керування проєктами, слід також окреслити план розгортання ПЗ та його модулі, що використовуватимуться для цього. Спочатку додамо діаграму пакетів, що дозволить розділити функціонал застосунку і виокремити його частини, що застосовуватимуться при розгортанні.

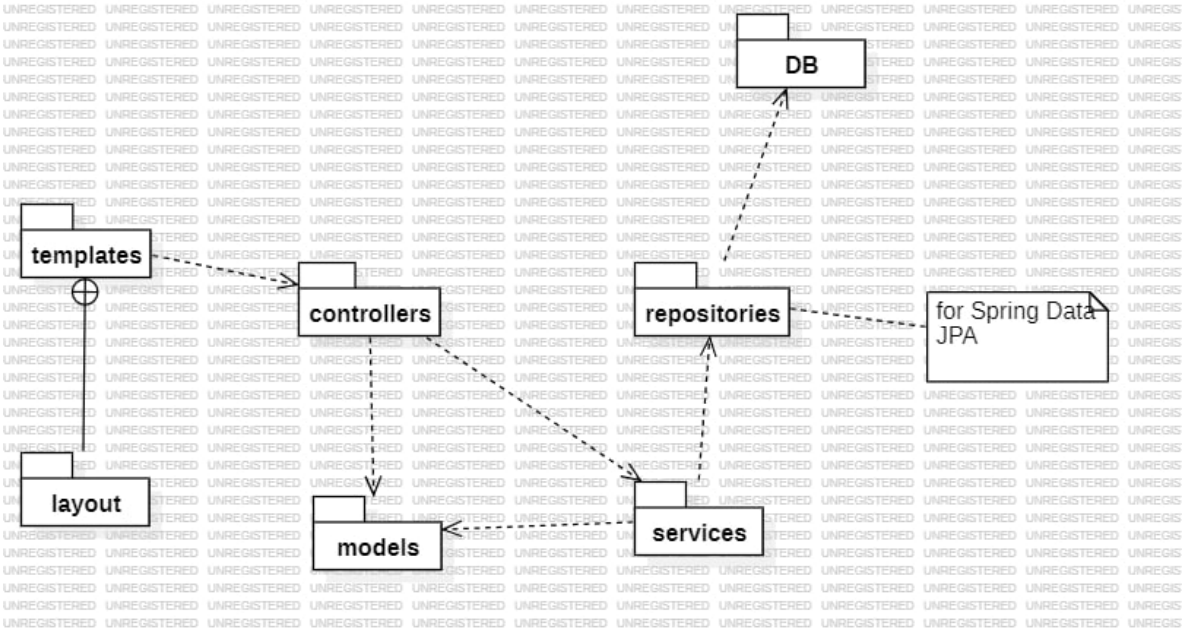

Рисунок 2.13 – Діаграма пакетів вебзастосунку

Спираючись на вищевказану діаграму можна буде будувати архітектуру вебзастосунку. Наразі ж застосуємо її для опису модулів ПЗ при розгортанні. Для демонстрації цього створимо діаграму розгортання. Вона демонструватиме вузли системи, що взаємодіятимуть між собою. Діаграми розгортання надають візуальне подання фізичної архітектури системи, включаючи сервери, бази даних та інші компоненти інфраструктури, а також

їхні взаємозв'язки та комунікаційні шляхи. Вони відображають, як програмні модулі та компоненти розміщуються на апаратних вузлах, ілюструючи, як інформація та процеси передаються між різними частинами системи у реальному часі.

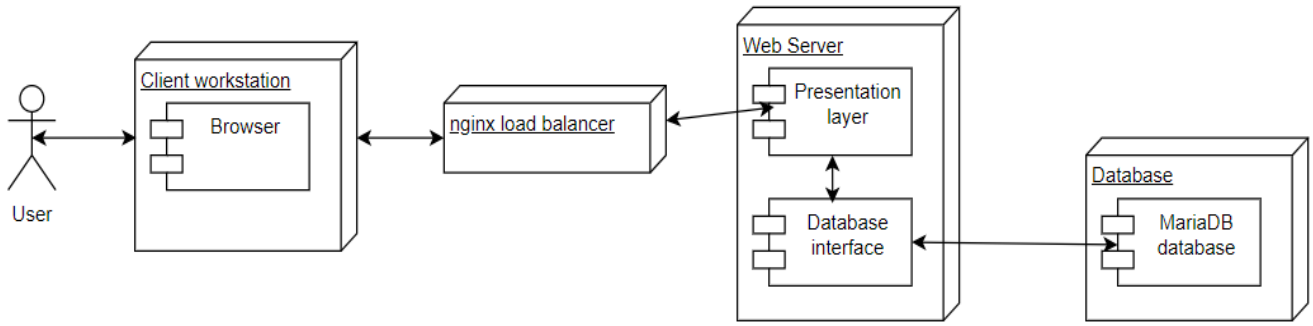

Рисунок 2.14 – Діаграма розгортання системи

Діаграми розгортання та пакетів стануть у нагоді при створенні директорій та пакетів на етапі побудови проєкту. Це дозволить логічно розподілити програмну реалізацію для полегшення процесу розробки вебзастосунку.

Висновки до розділу 2

У другому розділі було наведено сценарії використання вебзастосунку керування проєктами, створено діаграму використання для візуалізації можливостей акторів. Побудовано діаграми прецедентів для зображення акторів та їх взаємодії із системою керування проєктами. Додано діаграми діяльності, станів та відображень для конкретизації функціоналу застосунку та його частин, а також коректної делегації завдань лише у межах тих елементів системи, що мають безпосередньо за них відповідати. Розроблено діаграми пакетів та розгортання для демонстрації розгортання вебзастосунку та структурованої побудови проєкту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ КЕРУВАННЯ ПРОЄКТАМИ

3.1 Опис технологій розробки

Для створення вебзастосунку з базою даних обрано Spring – найпопулярніший фреймворк мови Java, що застосовується для розробки проєктів з архітектурою MVC. Його популярність ґрунтується, перш за все, на широкому спектрі доступних технологій, що суттєво полегшують розробку програм такого типу, зокрема, модульні тести, Spring Data JPA, конфігураційні файли тощо. Окрім цього, широке використання анотацій роблять код більш зрозумілим та структурованим. Для підвищення читабельності коду також застосовано і бібліотеку Lombok.

В основу вебзастосунку покладено патерн Model-View-Controller, що має на меті розподілення бізнес-логіки, інтерфейсу та даних.

Модель є ядром додатку і відповідає за управління даними, бізнес-логікою та правилами. Вона представляє об'єкти та взаємодіє з базою даних або іншими джерелами даних для зберігання та отримання інформації. Модель також здійснює основні обчислення та обробку даних, надаючи чистий і абстрактний інтерфейс для роботи з інформацією.

Представлення відповідає за відображення інформації користувачеві. Воно формує інтерфейс користувача та реагує на зміни в моделі. У контексті веб-додатків, представлення зазвичай включає HTML, CSS та JavaScript, які створюють веб-сторінки, що користувач бачить і з якими взаємодіє. Представлення забезпечує інтуїтивний та зручний інтерфейс, дозволяючи користувачам взаємодіяти з додатком.

Контролер є посередником між моделлю та представленням. Він обробляє вхідні запити користувача, взаємодіє з моделлю для виконання необхідних операцій і оновлює представлення відповідно до результатів цих операцій. Контролер інтерпретує дії користувача, такі як натискання кнопок

чи введення даних, і координує оновлення моделі та представлення. Контролер відповідає за маршрутизацію запитів, управління потоком даних та забезпечення належної взаємодії між компонентами.

Патерн MVC сприяє чіткому розподілу функцій між компонентами, що полегшує розробку та підтримку додатку. Кожен компонент зосереджується на своїх завданнях: модель обробляє дані, представлення відповідає за інтерфейс, а контролер управляє взаємодією між ними. Це забезпечує модульність та підвищує зручність внесення змін або додавання нових функцій.

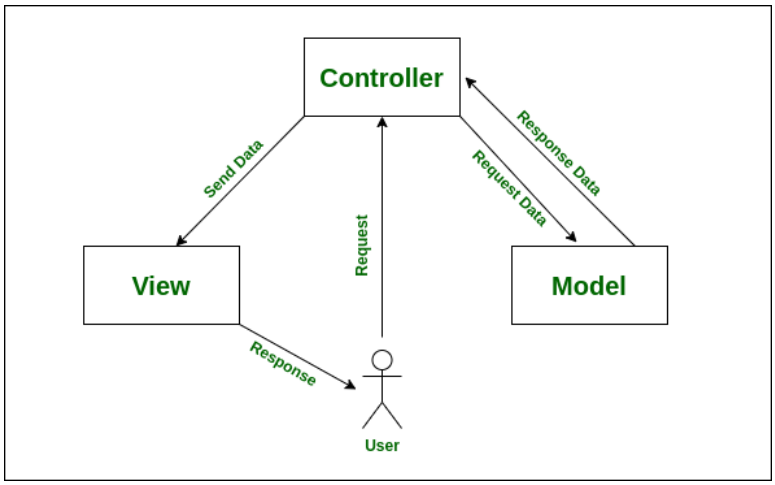

Рисунок 3.1 – Структура патерну MVC

У якості сховища даних використано СКБД MariaDB – розгалуження MySQL, що відрізняється зручнішим інтерфейсом для роботи із СКБД та покращеним фунціоналом SQL-ін’єкцій [10]. Вона також має розширені функції безпеки та масштабованості, що дозволяють ефективно керувати великими обсягами даних та забезпечувати їхню цілісність і захищеність.

Для автоматизації запитів до БД використано технологію Spring Data JPA, а для відслідковування сесій – Spring Session. Spring Data JPA дозволяє пов’язати класи-сутності із таблицями бази даних шляхом використання відповідних анотацій, тобто підтримує ORM. Таким чином можна автоматизувати створення таблиць та зв’язків між ними. Окрім цього, Spring Data JPA також суттєво полегшує формування запитів до БД. Замість

написання безпосередньо SQL-запитів використовуються класи-репозиторії, що пов’язуються із відповідною моделлю та дочірні класи-сервіси на яких, власне, і делегується завдання роботи зі сховищем даних. Вони автоматично генерують набір потрібних методів для виконання CRUD-операцій. У випадку потреби в ширшому функціоналі (наприклад, сортуванні), методи можна розширити всередині сервісу за допомогою методів іменування, не виносячи опис обробки даних у методи, не призначені для цього. Таким чином, дана технологія дозволяє роботи запити до БД шляхом виклику відповідних методів, що суттєво покращує читабельність коду та полегшує процес розробки серверної частини вебзастосунку [19, 20].

Spring Session — це технологія, розроблена для управління сесіями в Java веб-додатках. Вона забезпечує потужний і гнучкий спосіб роботи з сесіями, включаючи підтримку розподілених сесій, що може стати у нагоді при розробці вебзастосунку із розподіленим функціоналом. Spring Session дозволяє легко реалізувати складні вимоги до сесій у сучасних веб-додатках, забезпечуючи їхню стабільність і масштабованість [15].

Для обробки інформації всередині представлень застосовано двигун шаблонів Thymeleaf. Він дозволяє динамічно відображувати дані на представленнях, а також легко пов’язувати представлення із відповідними операціями. Thymeleaf підтримує різні типи шаблонів, включаючи HTML, XML, текстові файли та інші [7]. Він також пропонує потужний набір функцій, таких як цикли, умовні конструкції, маніпуляції з рядками та об'єктами, що дозволяє створювати складні і гнучкі шаблони та легко оброблювати інформацію у представленнях. Крім того, Thymeleaf легко інтегрується з Spring MVC, забезпечуючи безперешкодну роботу з контролерами і моделями Spring [8].

Окрім цього, використано фреймворк Bootstrap для опису дизайну клієнтської частини застосунку. Цей фреймворк надає широкий набір готових компонентів та стилів, що дозволяє швидко створювати сучасні і привабливі

інтерфейси. Bootstrap містить бібліотеки CSS і JavaScript для створення форм, кнопок, навігації, модальних вікон та інших елементів інтерфейсу. Окрім цього, він забезпечує підтримку адаптивного дизайну, тобто автоматично підлаштовує зовнішній вигляд сайту під різні розміри екранів та пристроїв. Використання Bootstrap значно скорочує час розробки і дозволяє зосередитися на функціональності та користувацькому досвіді.

3.2 Проєктування діаграми класів

Для відображення структури серверної частини застосунку, скористаємося діаграмою класів. Вона дозволяє візуалізувати структури класів, включаючи поля та їх типи даних, зв’язки між класами, а також методи, потрібні для реалізації вимог системи.

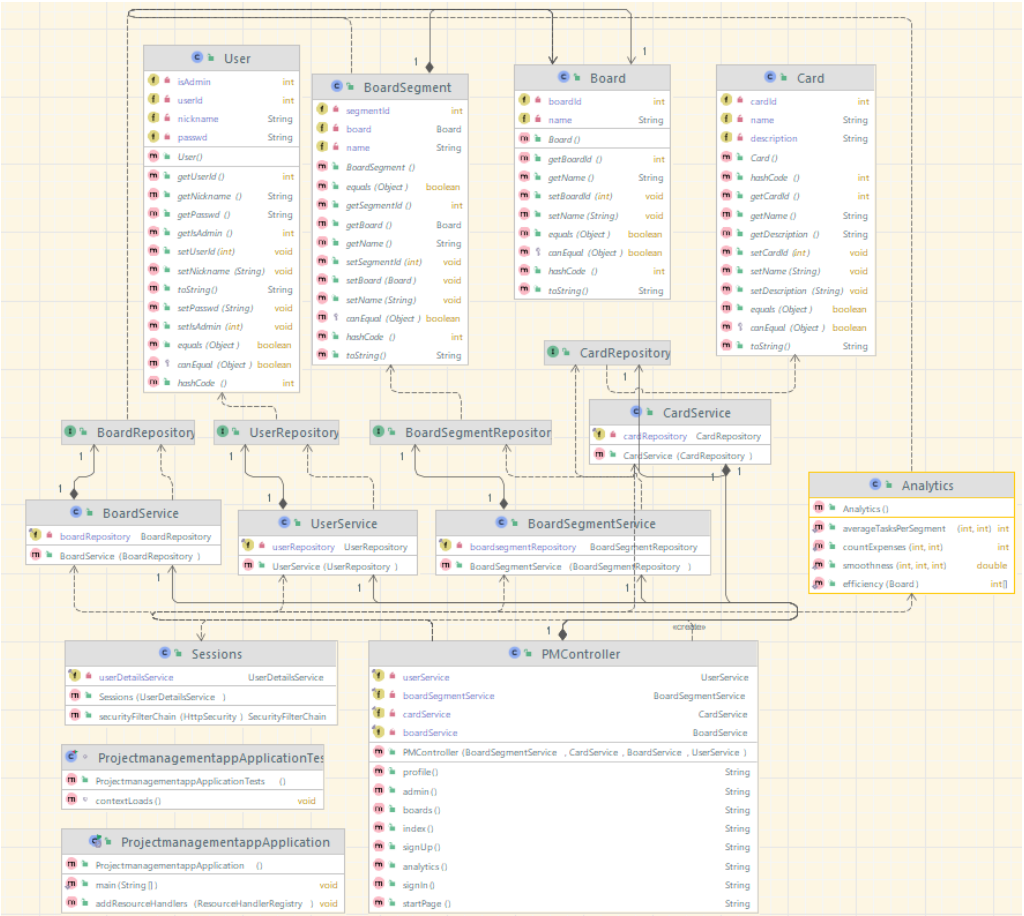

Рисунок 3.2 – Діаграма класів проєкту

У структурі серверної частини використовується архітектурний шаблон модель-представлення-контролер. Контролер застосовано для взаємодії між СКБД та користувацькими запитами. Для представлення користувачів, дошок та завдань використовуються моделі у вигляді POJO-класів. Окрім цього, застосовано класи-репозиторії та класи-сервіси для роботи із технологією Spring Data JPA [17]. У окремі класи виділено логіку керування сесіями та методи підрахунку метрик для аналітики.

3.3 Проєктування бази даних

Для візуалізації структури бази даних, створимо схему БД. Це дозволить визначити потрібні сутності, а також зв’язки між ними для наступних кроків зі створення сховища даних.

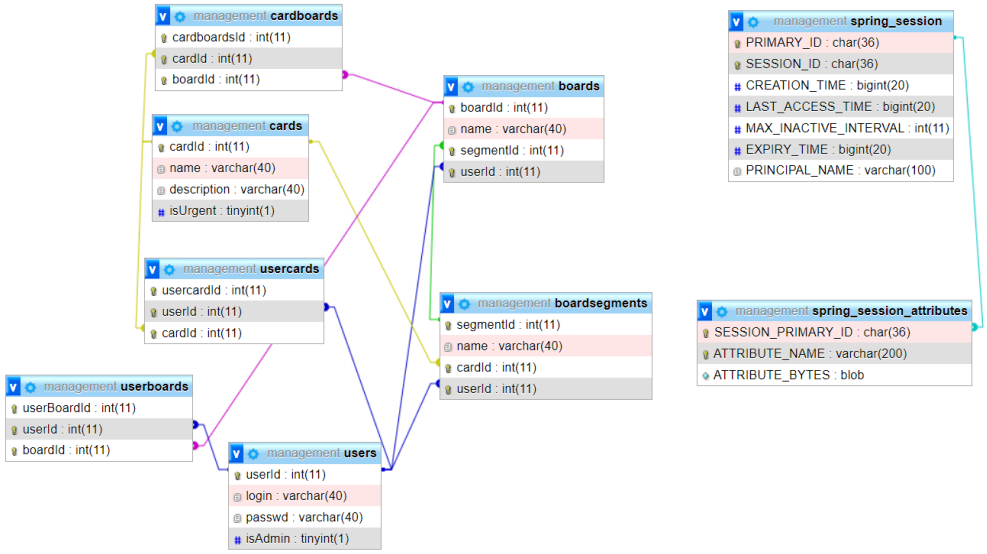

Рисунок 3.3 – Схема бази даних

На схему додамо таблиці для CRUD-операцій, а також для збереження сесій користувачів. На її основі здійснимо проєктування таблиць. Для кожної з них позначимо поля та їх типи.

Таблиця 3.1 – Таблиця бази даних users

Поле	Тип
userId	Integer, primary key auto_increment

Кінець таблиці 3.1

login	Varchar(40)
passwd	Varchar(40)
isAdmin	Tinyint

Таблиця 3.2 – таблиця бази даних boards

Поле	Тип
boardId	Integer, primary key auto_increment
name	Varchar(40)

Таблиця 3.3 – таблиця бази даних boardsegments

Поле	Тип
segmentId	Integer, primary key auto_increment
name	Varchar(40)
boardId	Integer, foreign key

Таблиця 3.4 – таблиця бази даних cardboards

Поле	Тип
cardboardsId	Integer, primary key auto_increment
cardId	Integer, foreign key
boardId	Integer, foreign key

Таблиця 3.5 – таблиця бази даних userboards

Поле	Тип
userBoardId	Integer, primary key auto_increment
userId	Integer, foreign key
boardId	Integer, foreign key

Таблиця 3.6 – таблиця бази даних spring_session

Поле	Тип
PRIMARY_ID	Char
SESSION_ID	Char
CREATION_TIME	Bigint
LAST_ACCESS_TIME	Bigint
MAX_INACTIVE_INTERVAL	Int
EXPIRY_TIME	Bigint
PRINCIPAL_NAME	Varchar(100)

Таблиця 3.7 – таблиця бази даних spring_session_attributes

Поле	Тип
SESSION_PRIMARY_ID	char
ATTRIBUTE_NAME	Varchar(100)
ATTRIBUTE_BYTES	blob

Таблиця 3.8 – таблиця бази даних cards

Поле	Тип
cardId	Integer, primary key auto_increment
name	Varchar(40)
description	Varchar(40)
isUrgent	Tinyint

Додамо відповідні таблиці до бази даних. Для цього скористаємося SQL-запитами. Окремо опишемо створення таблиць та зв’язки між ними за допомогою зовнішніх ключів. SQL-запит створення таблиць наведено у Додатку А.

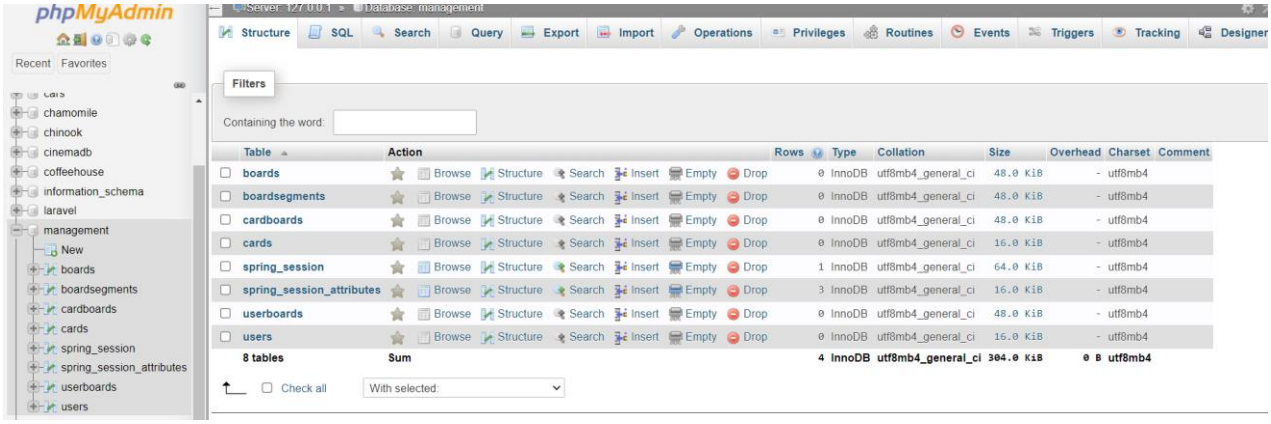

Рисунок 3.4 – Структура бази даних

Після створення таблиць можна переходити до опису бізнес-логіки застосунку, насамперед, взаємодії із новоствореною базою даних.

3.4 Програмна реалізація вебзастосунку

Після створення бази даних розпочнемо процес безпосередньої програмної реалізації вебзастосунку. Для створення проєкту із усіма потрібними залежностями, скористаємось Spring Initializr. Це інструмент та

API, який дозволяє швидко створювати проєкти на основі Spring Boot. Він значно полегшує початкову конфігурацію проєкту, автоматично генеруючи структуру проєкту з необхідними залежностями та налаштуваннями [4].

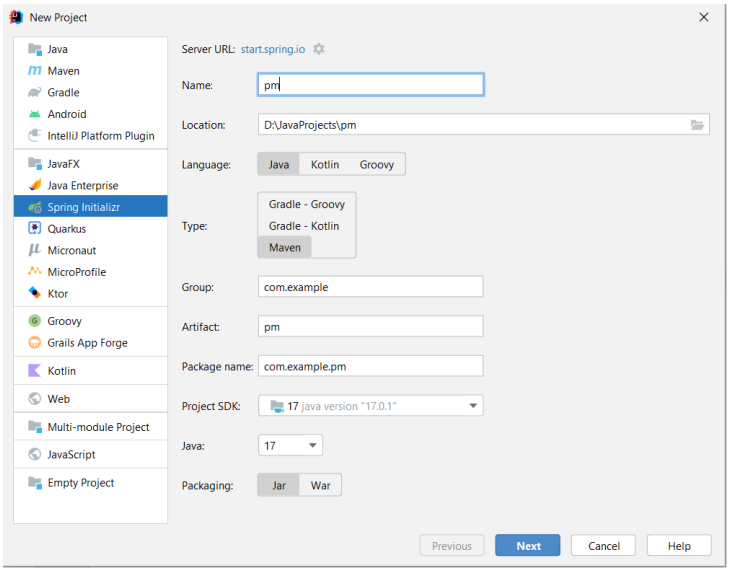

Рисунок 3.5 – Створення проєкту

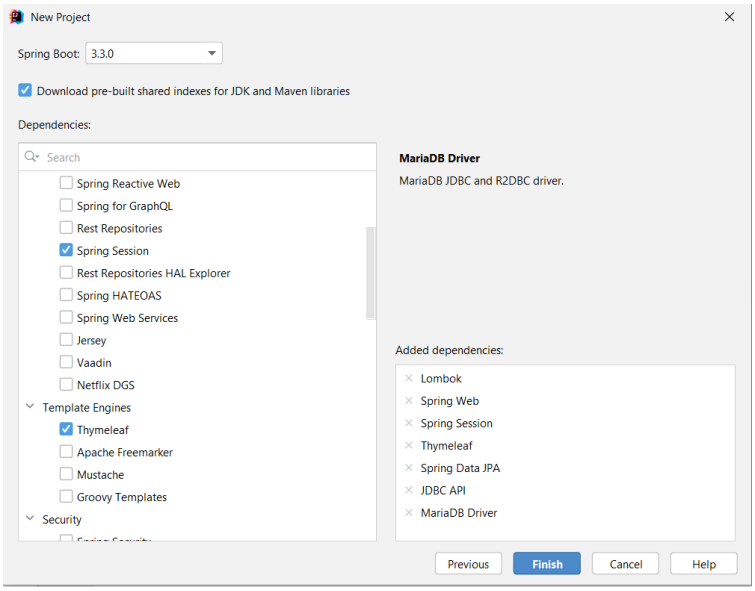

Рисунок 3.6 – Налаштування залежностей

Далі налаштуємо конфігураційні файли проєкту. Пізніше вони застосовуватимуться для роботи із різними заледностями, насамперед, JPA [5].

Код файлу application.properties:

```
spring.application.name=projectmanagementapp
spring.datasource.url=jdbc:mariadb://localhost:3306/pm
```

```
spring.datasource.username=root
spring.datasource.password=12345678
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.hibernate.naming.physical-
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardI
mpl
```

Код файлу system.properties:

```
java.runtime.version=17
```

Також додамо перевантажений метод addResourceHandlers для роботи із файлами з розширеннями html та css. Успадкуємо цей метод із класу WebMvcConfigurationSupport [16].

Код файлу ProjectmanagementappApplication.java:

```
package com.example.projectmanagementapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistr
y;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurationSup
port;
@SpringBootApplication
public class ProjectmanagementappApplication extends
WebMvcConfigurationSupport {
    public static void main(String[] args) {
        SpringApplication.run(ProjectmanagementappApplication.class,
args);
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        if (!registry.hasMappingForPattern("/assets/**")) {
            registry.addResourceHandler("/assets/**")
                    .addResourceLocations("classpath:/assets/");
        }
    }
}
```

Тепер упевнимося у правильності вказаних налаштувань та створимо підключення до бази даних. Для цього використаємо вбудований інструментарій обраного середовища розробки.

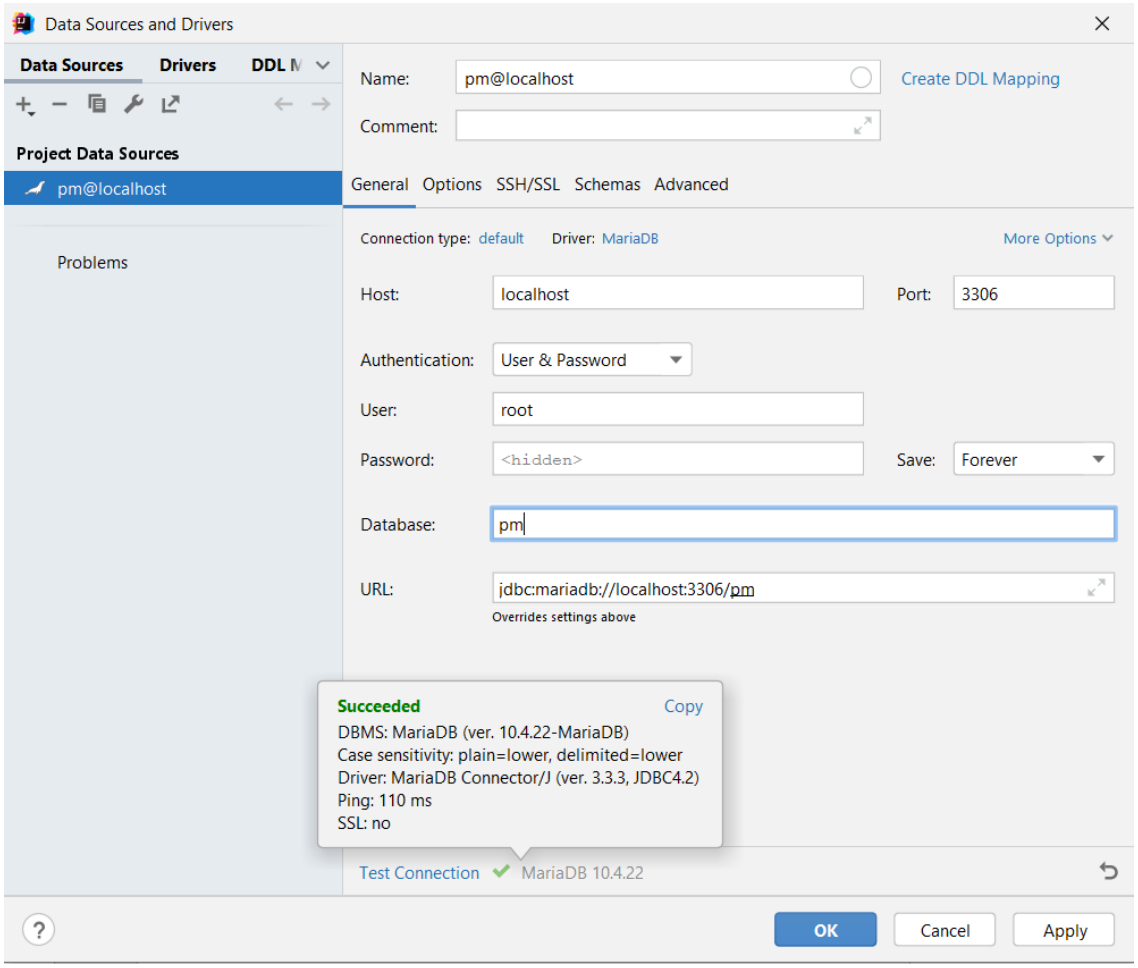

Рисунок 3.7 – Перевірка підключення

Після цього можна розпочати процес кодування серверної та клієнтської частин вебзастосунку.

3.4.1 Створення моделей

Розпочнемо із опису моделей, що застосовуватимуться для ORM. Для таких класів використаємо анотацію @Data бібліотеки Lombok для автоматичного створення гетерів, сетерів та конструкторів. Окрім цього, додамо відповідні JPA-анотації для пов’язування класу із таблицею бази даних. Нижче наведено код одного з класів-сутностей:

Код моделі User.java:

```
package com.example.krb.models;
import lombok.Data;
import org.hibernate.annotations.Type;
import jakarta.persistence.*;
@Entity
@Data
@Table(name = "users")
public class User {
    @Id
    @Column(name = "userId")
    private int userId;
    @Column(name = "nickname")
    private String nickname;
    @Column(name = "passwd")
    private String passwd;
    @Column(name = "isAdmin")
    private int isAdmin;
}
```

Для роботи із базою даних використаємо клас-репозиторії та класи-сервіси. Далі наведено приклад структури подібних класів.

Код класу-репозиторію UserRepository.java:

```
package com.example.krb.repositories;
import com.example.krb.models.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
}
```

Код класу-сервісу UserService.java:

```
package com.example.krb.services;

import com.example.krb.repositories.BoardSegmentRepository;
import com.example.krb.repositories.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
```

```
public class UserService {
    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository){
        this.userRepository = userRepository;
    }
    public User findByLogin(String login);
}
```

3.4.2 Опис бізнес-логіки

Оскільки застосунок має структуру модель-представлення-контролер, то логіка роботи вебзастосунку міститься у контролері та допоміжних класах. У випадку розроблюваної системи, реалізація взаємодії представлень та сховища даних знаходиться у контролері, у той час як інструменти аналітики винесено у окремий допоміжний клас.

Контролер у якості полів містить сервіси для роботи із базою даних. Методи, у залежності від тип запиту, позначаються анотаціями @GetMapping, @PatchMapping або @PostMapping. Параметром передається шлях, оброблюваний методом. У створено контролері застосовано Get- та Post-методи.

Код методів Get та Post, що оброблюють представлення сторінки реєстрації:

```
@GetMapping("sign-up")
public String signUpGet(Model model){
    model.addAttribute("signUpModel", new SignUpModel());
    return "sign-up";
}

@PostMapping("sign-up")
public String signUpPost(@ModelAttribute SignUpModel signUpModel, Model
model) {
    List<User> users = userService.getAll();
    if(userService.getByName(signUpModel.getNickname())!=null) {
        model.addAttribute("error", "This nickname is taken!");
```

```
            return "sign-up";
        }
        if(!signUpModel.getPasswd().equals(signUpModel.getConfirm())) {
            model.addAttribute("error", "Passwords aren't matching!");
            return "sign-up";
        }
        if(signUpModel.getPasswd().length()<8) {
            model.addAttribute("error", "Password must contain at least 8
characters!");
            return "sign-up";
        }
        User user = new User();
        user.setNickname(signUpModel.getNickname());
        user.setPasswd(signUpModel.getPasswd());
        user.setAdmin(false);
        userService.save(user);
        return "redirect:/start-page";
    }
```

У наведеному прикладі використано об’єкт класу Model для виведення повідомлень модерації. Анотацією @ModelAttribute позначається безпосередньо модель [18]. Окрім цього, для повернення до кореневого шляху застосовано redirect у повертаємому значенні. Оскільки контролер має передусім відповідати своїй ролі у межам архітектури MVC, реалізацію інструментів аналітики винесено у окремий клас. Цей клас міститиме статичні методи для підрахунку обраних метрик аналізу ефективності керування проєктами. Наприклад, реалізуємо метод, що визначатиме на основі даних проєкту та cookies рівномірнсть швидкості виконання завдань.

Код методу оцінки плавності виконання завдань:

```
public static double smoothness(int days, int tasks, int segments) {
         return
((double)tasks/(double)days)/(double)averageTasksPerSegment(tasks,
segments);
}
public static int averageTasksPerSegment(int tasks, int segments){
    return tasks/segments;
}
```

Також реалізуємо Spring Session для відстеження сесій користувачів із метою як перешкоджання доступу до інтерфейсу із розширеними правами доступу, так і збереження активних сесій для полегшення регулярного користування вебзастосунком. Код класу, що займається обробкою сесій, наведено у Додатку А.

3.4.3 Створення представлень

Насамкінець, створимо інтерфейс користувача. Оскільки вебзастосунок побудований на базі Spring Framework, зручно використовувати шаблони для відображення та роботи із даними на уявленнях [12]. Для цього застосуємо двигун шаблонів Thymeleaf.

Перш за все, виокреслимо спільну частину інтерфейсу для декількох представлень. Нею є панель керування (меню у верхній частині сторінок) та футер. Thymeleaf дозволяє виокремлювати такі частини у окремі файли, а потім використовувати у інших, використовуючи тег th:fragment. Скористаємося цим і виділимо користувацьке меню у окремий файл.

Код файлу header.html:

```
  <style>
            .fixed-button {
                  position: fixed;
          bottom: 20px;
          right: 20px;
           }
    .             btn-sm {
          padding: 0.25rem 0.5rem;
          font-size: 0.875rem;
          }

          .fa-sign-out-alt {
              font-size: 0.875rem;
           }
</style>
  <div th:fragment="sharedheader" class="fixed-top">
    <nav class="navbar navbar-expand-sm navbar-light bg-white border-
```

```
bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" style="font-family: 'Courier New',
monospace; font-size: 24px"><b>PM App</b></a>
            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link" th:href="="@{~/boards-
view}">Boards</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/analytics-
view}">Analytics</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/profile-view}">My
profile</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/privacy-
policy}">Privacy policy</a>
                </li>
            </ul>
            <a class="btn btn-sm btn-primary" th:href="@{~/start-page}">
                <i class="fas fa-sign-out-alt mr-2"></i>Logout
            </a>
        </div>
    </nav>
</div>
```

Теги th:href застосовано для навігації застосунком. Аналогічно, відокремимо футер.

Код файлу footer.html:

```
  <div th:fragment="sharedfooter" class="bg-dark text-white py-2">
    <div class="container text-center">
        <p class="mb-1">
            <b class="text-info">Got any problems? Contact support at <a
href="mailto:nt600136@gmail.com" class="text-
info">nt600136@gmail.com</a></b>
        </p>
        <div>&copy;2024 - TNiurhechev</div>
```

```
    </div>
</div>
```

Аналогічно, розробимо окремі фрагменти для інтерфейсу адміністратора.

Код файлу adminheader.html:

```
<style>
           .fixed-button {
                 position: fixed;
         bottom: 20px;
         right: 20px;
          }
    .            btn-sm {
         padding: 0.25rem 0.5rem;
         font-size: 0.875rem;
         }

         .fa-sign-out-alt {
             font-size: 0.875rem;
          }
</style>
  <div th:fragment="adminheader" class="fixed-top">
    <nav class="navbar navbar-expand-sm navbar-light bg-white border-
bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" style="font-family: 'Courier New',
monospace; font-size: 24px"><b>PM App</b></a>
            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/boards-view-
admin}">All boards</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/admin-add}">Add
admin</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/profile-view}">My
profile</a>
                </li>
```

```
                <li class="nav-item">
                    <a class="nav-link" th:href="@{~/privacy-
policy}">Privacy policy</a>
                </li>
            </ul>
            <a class="btn btn-sm btn-primary" th:href="@{~/start-page}">
                <i class="fas fa-sign-out-alt mr-2"></i>Logout
            </a>
        </div>
    </nav>
</div>
```

Код файлу adminfooter.html:

```
<div th:fragment="adminfooter" class="bg-dark text-white py-2">
         <div class="container text-center">
          <p class="mb-1">
            <b class="text-success">Admin mode</a></b>
        </p>
        <div>&copy;2024 - TNiurhechev</div>
    </div>
</div>
```

Після опису загальної частини, перейдемо до кодування окремих представлень. За допомогою тегу th:insert у них можна включати створені фрагменти. Нижче наведено приклад використання тегу у одному з представлень.

Код файлу privacy-policy.html:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>PM App - Privacy policy</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.mi
n.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
</head>
<body>
<div th:insert="header :: header"></div>
<div class="text-center" style="margin-top: 200px">
```

```
  <h2 class="display-3">Privacy policy</h2>
  <p class="font-weight-light text-muted" style="margin: auto; width:
50%; font-size: 24px">
    PM Inc. is committed to our users' privacy. We <b>DO NOT</b> collect
or store any customer information (e.g. names,
    geolocation, users' browser etc.). We also do not share any
information with third parties or store any
    Cookies/session data. By using this website, you recognize and
confirm that you are 7 years old or older.
  </p>
  <br />
  <br />
  <div>
    Policy is due 31.12.2024 ©2024 TNiurhechev
  </div>
</div>
<div th:insert="footer :: footer"></div>
</body>
</html>
```

Ще одним часто застосовуваним тегом є th:action. Цей тег використовується для того, щоб пов’язати форму із post-методом контролера. Таким чином, передавши у якості аргументу шлях, вказаний у анотації @PostMapping, можна автоматизувати виклик post-метода при заповненні форми. Для прив’язки полів форми до полів моделі використовуються теги th:object та th:field. Перший використовується для позначення, який саме клас використовується у якості моделі, а другий – яке поле класу представляє кожне поле введення на формі. На прикладі сторінки авторизації можна побачити використання цих тегів.

Код файлу sign-in.html:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Sportswear - Sign in</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.mi
n.css" integrity="sha384-
```

```
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
</head>
<body>
    <header class="fixed-top">
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-
light bg-white border-bottom box-shadow mb-3">
            <label class="navbar-brand text-center" style="font-family:
'Courier New', monospace;
                font-size: 24px"><b>PM App</b></label>
        </nav>
    </header>
    <div class="text-center" style="margin-top: 180px">
        <h2 class="display-4">Sign in</h2>
        <form style="width: 20%; margin: auto" action="#"
th:action="@{/sign-in}" th:object="${signInModel}" method="post">
        <div class="form-group">
            <label>Nickname</label>
            <input type="text" class="form-control"
th:field="*{nickname}" />
        </div>
        <div>
            <label>Password</label>
            <input type="password" class="form-control"
th:field="*{passwd}" />
        </div>
        <button type="submit" class="btn btn-lg btn-outline-primary"
style="margin-top: 10px">Sign in</button>
        </form>
        <p class="text-danger" th:text="${error}"></p>
    </div>
    <div th:insert=" footer :: footer"></div>
</body>
</html>
```

У наведеному прикладі також застосовано тег th:text для відображення відповідного тексту. Передане у тег значення відповідає об’єкту класу Model, що вказаний у методах контролеру.

Наостанок, додамо потрібні скрипти. Оскільки частиною застосунку є дошка канбан, важливою є можливість легкої динамічної зміни її структури.

Це суттєво впливає на зручність користувацького інтерфейсу. Додамо скрипт, що дозволяв би переміщувати картки між стовпцями методом drag-and-drop замість використання кнопок або інших незручних реалізацій.

Код файлу drag.js:

```
const draggables = document.querySelectorAll(".task");
const droppables = document.querySelectorAll(".segment");
draggables.forEach((task) => {
task.addEventListener("dragstart", () => {
task.classList.add("is-dragging");
});
task.addEventListener("dragend", () => {
task.classList.remove("is-dragging");
});
});

droppables.forEach((zone) => {
zone.addEventListener("dragover", (e) => {
e.preventDefault();

const bottomTask = insertAboveTask(zone, e.clientY);
const curTask = document.querySelector(".is-dragging");

if (!bottomTask) {
zone.appendChild(curTask);
} else {
zone.insertBefore(curTask, bottomTask);
}
});
});

const insertAboveTask = (zone, mouseY) => {
const els = zone.querySelectorAll(".task:not(.is-dragging)");
let closestTask = null;
let closestOffset = Number.NEGATIVE_INFINITY;
els.forEach((task) => {
const { top } = task.getBoundingClientRect();

const offset = mouseY - top;
if (offset < 0 && offset > closestOffset) {
closestOffset = offset;
```

```
closestTask = task;
}
});
return closestTask;
};
```

Наведений вище код забезпечує можливість перетягування карток між стовпчиками дошки.

Висновки до розділу 3

У розділі 3 було описано стек технологій, що використовуватиметься для розробки вебзастосунку керування проєктами. Зокрема, використану СКБД, фреймворки для розробки клієнтської та серверної частин, API. Створено діаграму класів для відображення структури серверної частини вебзастосунку. Зображено класи, їх методи, поля, внутрішні класи тощо. Відображено зв’язки між класами та їх тип.

Здійснено проєктування бази даних для визначення необхідних сутностей та зв’язків між ними. На основі створеної схеми побудовано базу даних вебзастосунку. Наведено структуру створених таблиць та запити, застосовані для їх створення.

Описано покроковий шлях програмної реалізації вебзастосунку керування проєктами з використанням визначених технологій. Створено вебзастосунок зі структурою MVC. Інтегровано інструменти для об’єктно-реляційного представлення даних та відстеження сесій. Реалізовано розподілення функціоналу шляхом розробки окремих інтерфейсів для користувачів із різними правами доступу.

У результаті проведеної роботи створено вебзастосунок керування проєктами.

4 ОГЛЯД ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

Вебзастосунок містить 11 представлень: сторінки авторизації, реєстрації, перегляду дошки, переліку дошок, аналітики, профілю та інформаційну для користувача, а також переліку усіх дошок, редагування окремої та додавання нових облікових записів із розширеними правами для адміністратора. Загальним є представлення на випадок виникнення помилок при обробці запитів.

4.1 Огляд дизайну вебзастосунку

Задля кращого уявлення про необхідні елементи інтерфейсу та їх коригування у випадку невідповідності вимогам, почнемо зі створення макетів.

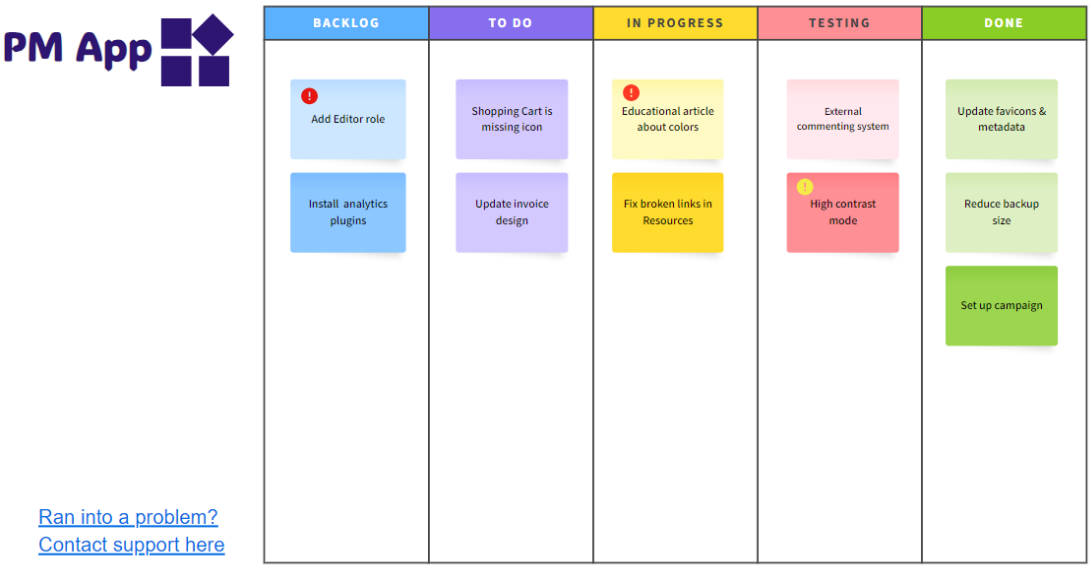

Рисунок 4.1 – Перегляд активної дошки

При перегляді проєкту користувач бачитиме активну канбан дошку зі створеними картками. Кожна картка знаходиться у стовпчику таблиці, що відповідає стану виконання завдання, яке вона позначає та, в залежності від терміновості, може мати відповідну позначку. Зліва розташоване меню, задля

зручності роботи оформлене у вигляді спливаючого списку. У нижньому лівому кутку знаходиться посилання на сторінку підтримки.

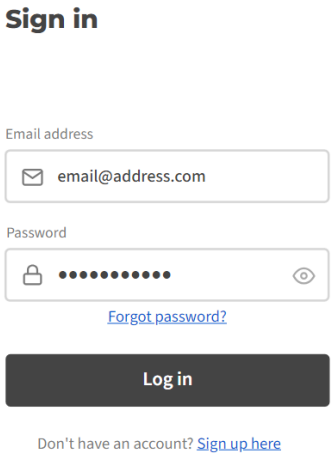

Рисунок 4.2 – Вікно авторизації

При вході на сайт користувач потрапляє на сторінку авторизації, що містить поля для логіну та паролю, а також кнопку підтвердження. Окрім цього, сторінка має посилання на сторінки реєстрації та відновлення паролю.

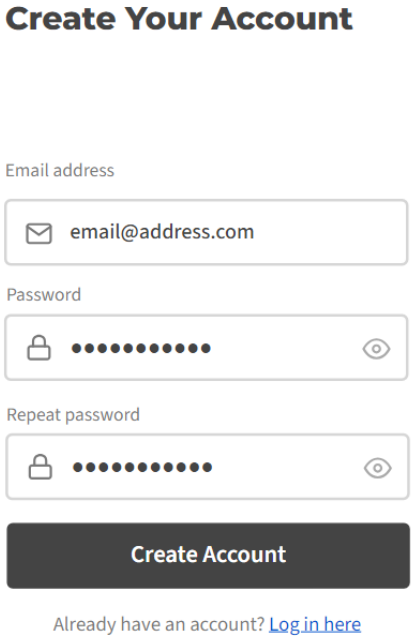

Рисунок 4.3 – Вікно реєстрації

У випадку відсутності облікового запису користувач переходить на сторінку реєстрації, що має додаткове поле для підтвердження паролю, а також посилання назад на сторінку авторизації.

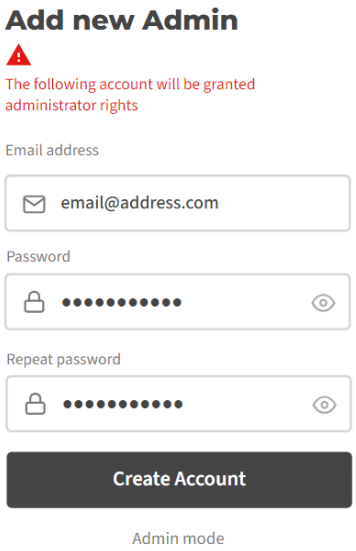

Рисунок 4.4 – Вікно реєстрації облікового запису із правами адміністратора

Оскільки у застосунку розподілено функціонал для різних акторів, для убезпечення даних та функціонування вебзастосунку право на створення облікових записів адміністраторів має лише власне адміністратор. Для цього він користується формою, схожою на користувацьку.

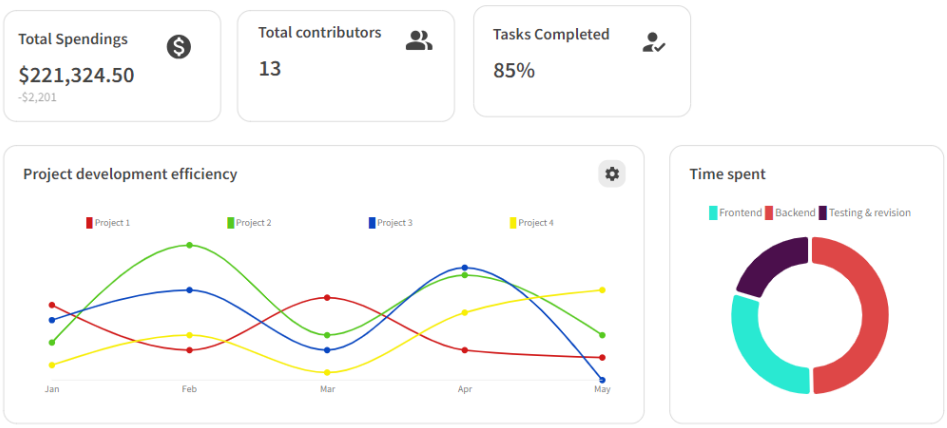

Рисунок 4.5 – Перегляд звіту про виконання проєкту

Однією з головних переваг розроблюваного ПЗ має бути наявність інструментів для збору статистики для саморефлексії та пошуку ефективних стратегій розробки ПЗ. Для перегляду цих даних існує окремо сторінка, де висвітлено рівномірність виконання завдань у порівнянні із попередніми проєктами, час, витрачений на кожен з етапів розробки та загальну статистику.

4.2 Огляд функціоналу вебзастосунку

Оглянемо створений на основі мокапів інтерфейс та його функціонал. Однією з частин інтерфейсу, що збігатиметься для всіх користувачів, є сторінки реєстрації та авторизації. Обидві представляють з себе форми, що мають поля для введення логіну та паролю (сторінка реєстрації має також поле для підтвердження паролю), кнопку підтвердження та приховане поле, в якому з'являтиметься інформація про помилки у разі їх виникнення.

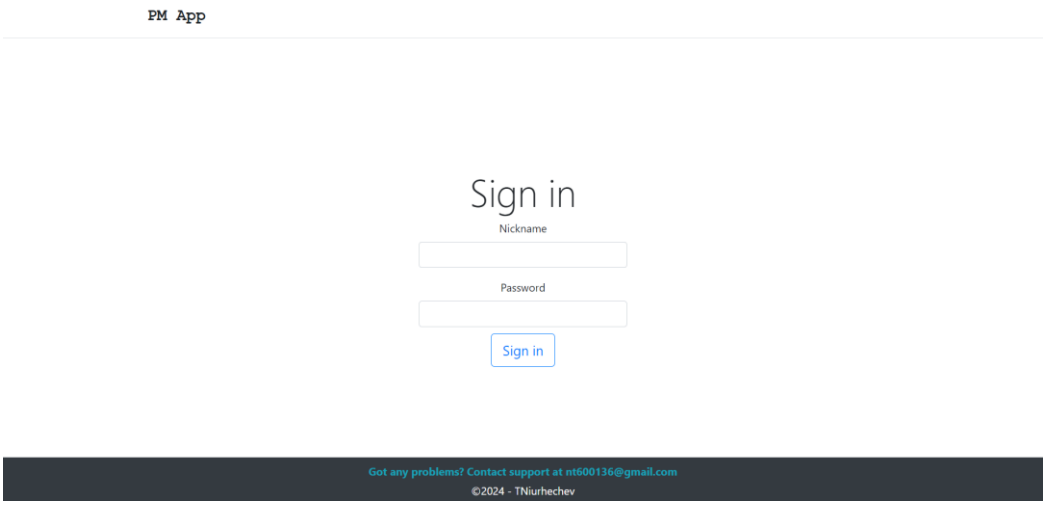

Рисунок 4.6 – Сторінка авторизації

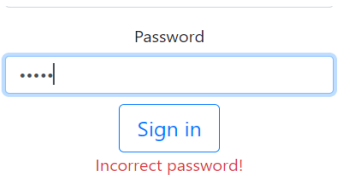

Рисунок 4.7 – Сторінка авторизації, приклад введення неправильних даних

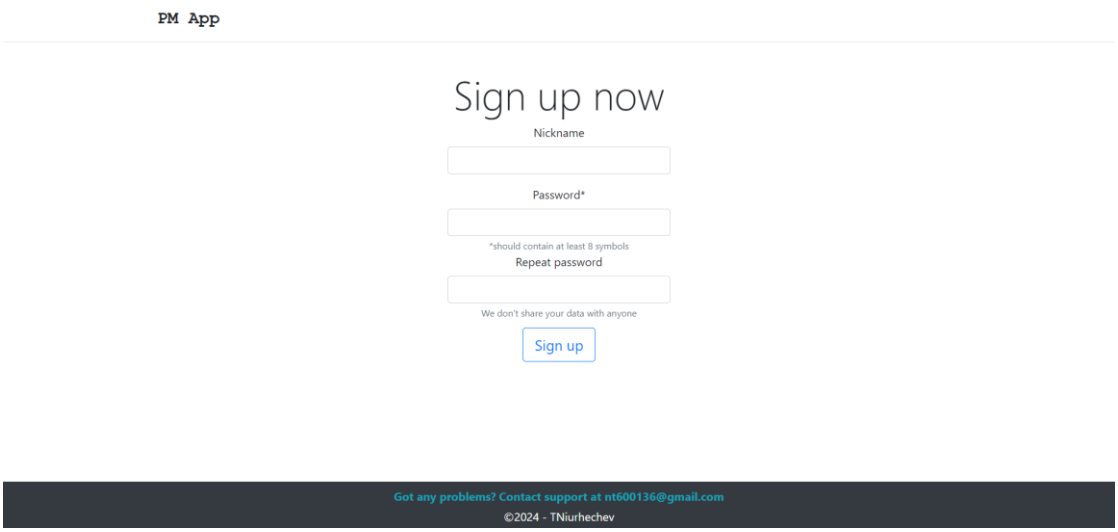

Рисунок 4.8 – Сторінка реєстрації

Для кожної дошки доступна можливість перегляду. На ній відображаються стовпчики та завдання, що можна перетягувати між ними. У першому стовпчику є кнопка для додавання нових карток. Окрім цього, поруч знаходяться кнопки для керування учасниками проєкту та створення нових сегментів. Для карток доступний детальний перегляд, а також встановлення позначки про терміновість завдання.

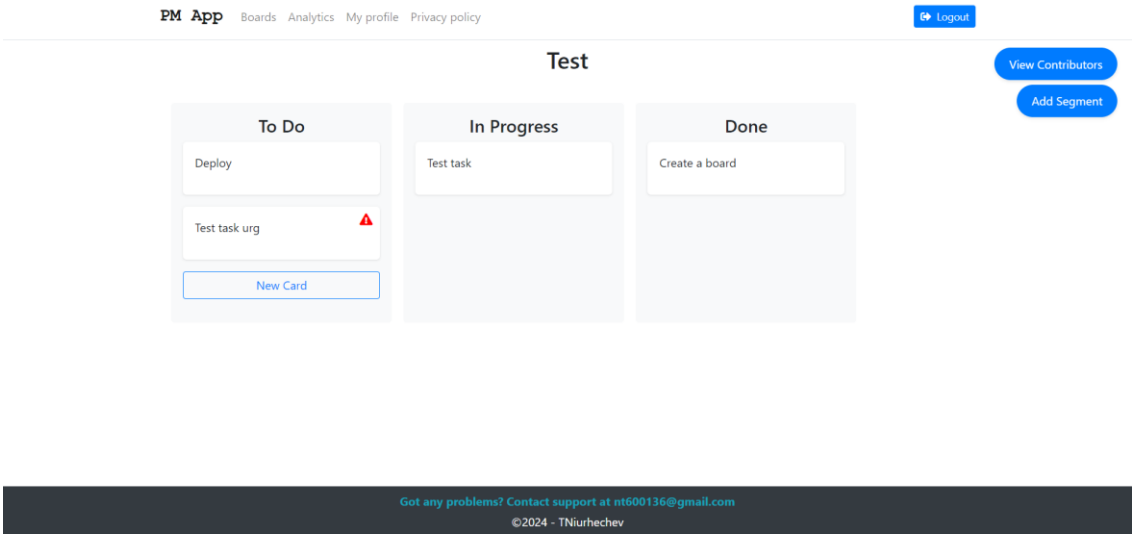

Рисунок 4.9 – Перегляд дошки

На сторінці аналітики доступний перегляд метрик для аналізу ефективності виконання завдань. Зокрема, приблизний підрахунок витрат для роботодавця, кількість залучених користувачів та завдань, що знаходяться на

останньому етапі розробки (у останньому стовпчику). Окрім цього, на сторінці також можна переглянути темп виконання завдань у порівнянні з минулими проєктами задля покращення рефлексії та, у випадку наявності тегів карток, обсяг завдань різних типів.

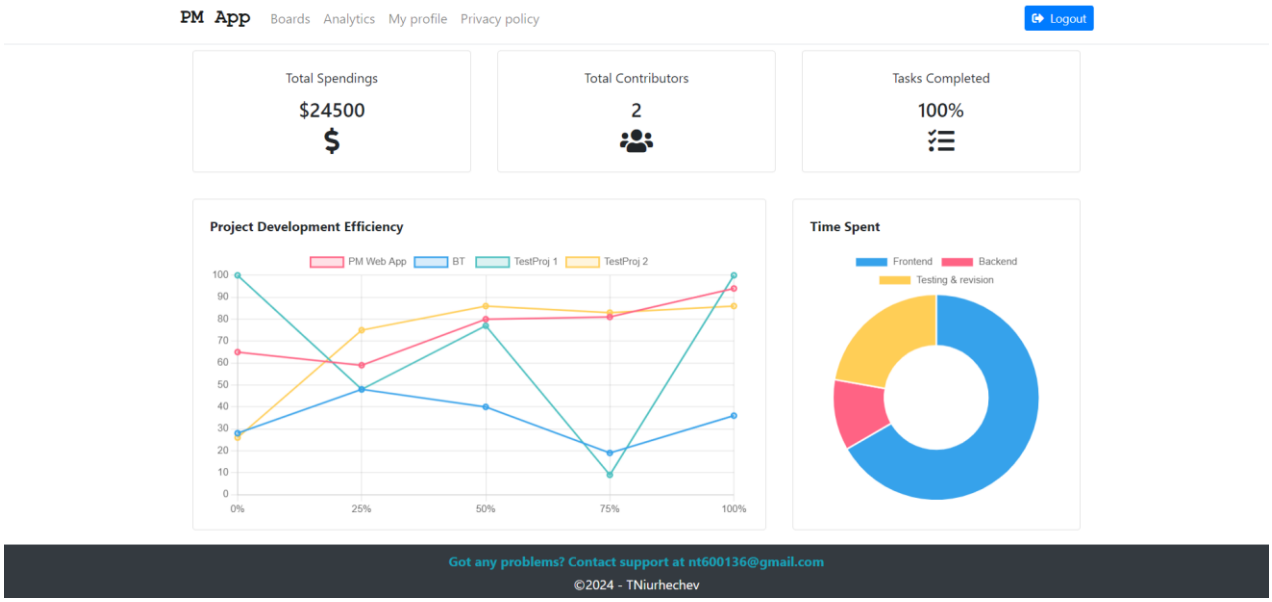

Рисунок 4.10 – Сторінка аналізу

Інформаційна сторінка містить інформацію щодо обробки даних користувачів.

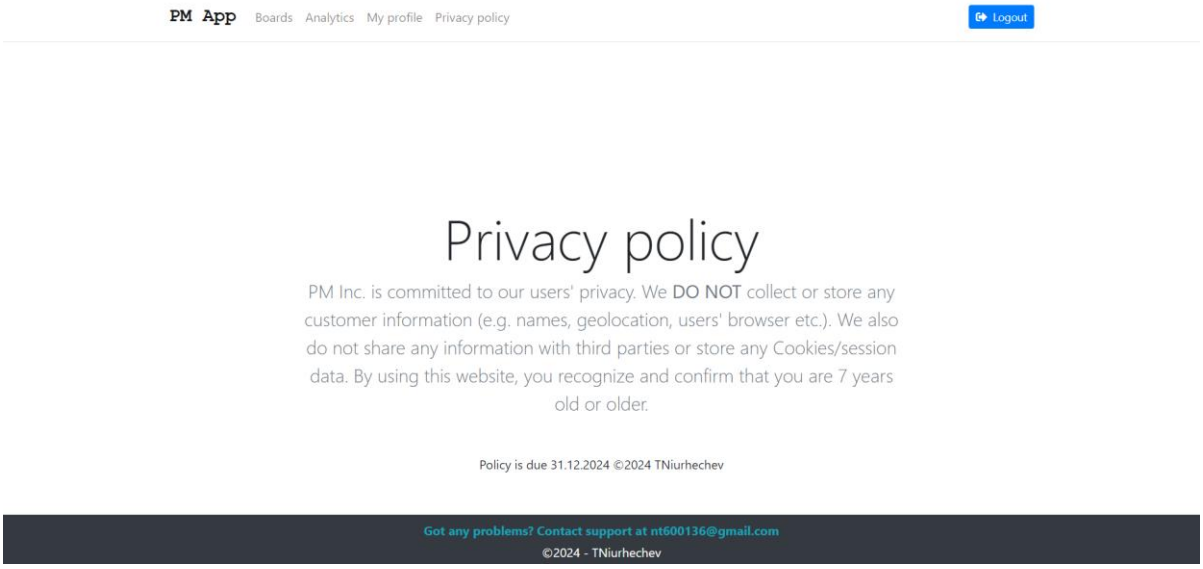

Рисунок 4.11 – Інформаційна сторінка

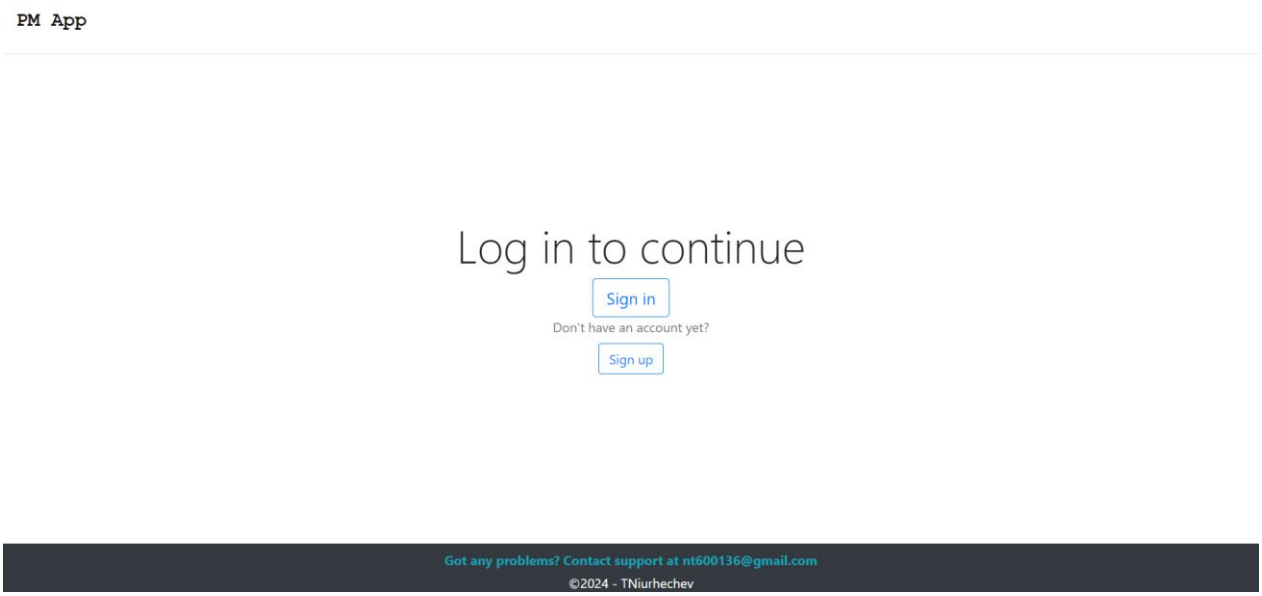

Рисунок 4.12 – Головна

Форма додавання адміністратора є аналогічною формі реєстрації звичайного користувача, за виключенням наявності попереджувального тексту.

Сторінка перегляду інформації про картку містить короткий (до 40 символів) опис завдання, та користувача, що до неї прив’язаний. Також наявні кнопки зміни користувача та видалення картки.

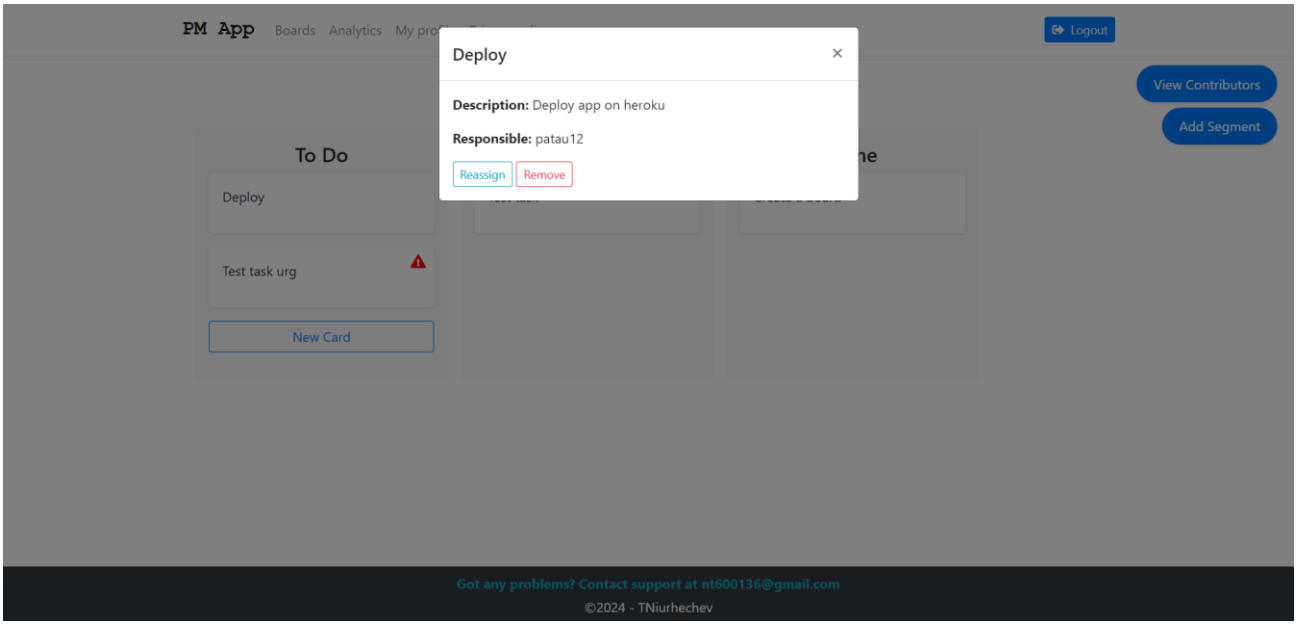

Рисунок 4.13 – Інформація про картку

Форма додавання адміністратора є аналогічною формі реєстрації звичайного користувача, за виключенням наявності попереджувального тексту.

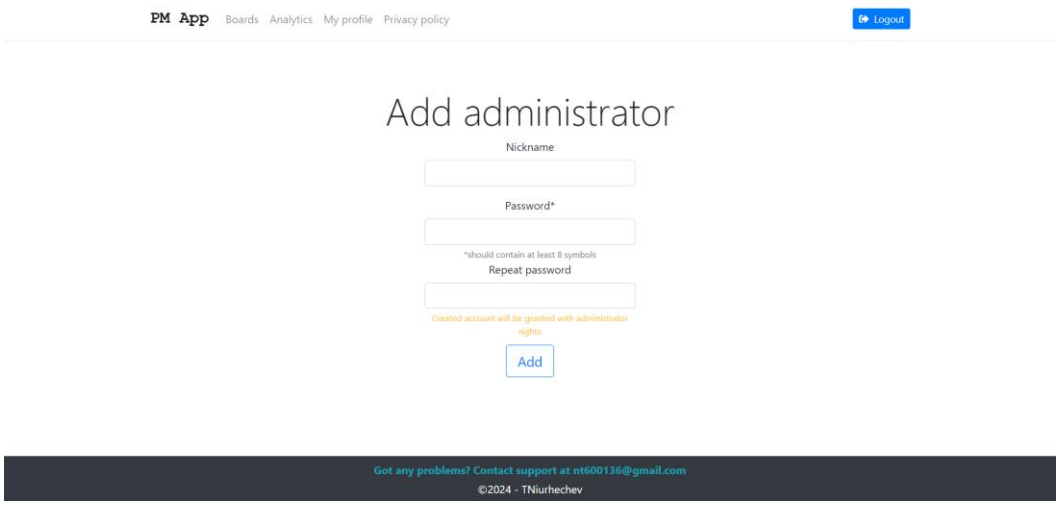

Рисунок 4.14 – Додавання адміністратора

Також на представленні можна побачити змінену для адміністратора панель керування, як приклад розподілення інтерфейсів.

Висновки до розділу 4

У четвертому розділі було описано особливості дизайну вебзастосунку, його функціонал та причини наявності відповідних засобів у звичайного користувача та адміністратора.

Здійснено проєктування бази даних для визначення необхідних сутностей та зв’язків між ними. На основі створеної схеми побудовано базу даних вебзастосунку.

Описано покроковий шлях програмної реалізації вебзастосунку керування проєктами з використанням визначених технологій. Проведено тестування системи.

У результаті оглянуто інтерфейс та функціонал вебзастосунку керування проєктами.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи створено вебзастосунок керування проєктами. Застосунок, окрім загальноприйнятого функціоналу з відслідковування прогресу у командних проєктах із можливістю розподілу завдань та зміни їх статусу, також надає можливість зберігати різні метрики для аналізу ефективності виконання завдань та командної роботи загалом. Було наведено сценарії використання вебзастосунку, а також створено діаграми класів, станів та пакетів для моделювання програмного забезпечення та його функціоналу. Застосовано фреймворк Spring Framework та реляційну СКБД MariaDB для програмної реалізації вебзастосунку.

Для досягнення поставленої мети виконано наступні завдання:

– аналіз предметної області;

– розробка вимог до застосунку на основі проведеного аналізу;

– проєктування архітектури вебзастосунку керування проєктами;

– проєктування бази даних;

– розробка, тестування та розгортання вебзастосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Trello: вебсайт. URL: https://trello.com/ (дата звернення: 16.02.2024).

2. Monday: вебсайт. URL: https://monday.com/ (дата звернення: 17.02.2024).

3. ClickUp: вебсайт. URL: https://clickup.com/ (дата звернення: 19.02.2024).

4. Poul Klausen: Java 18: Algorithms and data structures. Bookboon: 2018. 203 p.

5. Marcello La Rocca. Advanced Algorithms and Data Structures. Manning: 2021. 768 p.

6. Josh Juneau, Tarun Telang. Java EE to Jakarta EE 10 Recipes: A Problem-Solution Approach for Enterprise Java. APress: 2022. 683 p.

7. Spring Controllers. URL: https://www.baeldung.com/spring-controllers (date of access: 05.03.2024).

8. Wolff D. Get Things Done with Trello: Your Quick Access to Productivity and Success includes a Step-by-Step Guide to Set Up and Implement Trello. Dominic Wolff: 2014. 110 p.

9. Scott La C. The Ridiculously Simple Guide to Trello: A Beginners Guide to Project Management with Trello. Scott La Counte: 2020. 85 p.

10. Pierre Mavro. MariaDB High Performance: Familiarize yourself with the MariaDB system and build high-performance applications. Packt: 2014 298 p.

11. Bob Hughes, Brian West, David I. Shepard. Книга Project Management for IT-Related Projects: 3rd edition. Bob Hughes: 2019. 162 p.

12. Annotated Controllers. URL: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller.html (date of access: 18.04.2024).

13. Етапи управління у IT-проєктами. URL: https://iampm.club/ua/blog/etapi-upravlinnya-it-projektami-inicziacziya-projektu-dlya-menedzhera/ (дата звернення: 2.05.2024)

14. Paul Deck. Spring MVC: a tutorial (Second edition). BrainySoftware: 2012. 162p.

15. Dr. Tim Downey. Guide to Web development with Java. Springer: 2021. 196p.

16. Shameer Kunjumohamed, Hamidreza Sattari, Alex Bretet, Geoffroy Warin. Spring MVC: Designing Real-World Web Applications. Packt: 2016. 766p.

17. Marten Deinum, Koen Serneels, Colin Yates, Seth Ladd, Erwin Vervaet, Christophe Vanfleteren. Pro Spring MVC: With Web Flow. Apress: 2012. 290p.

18. Emilien Kenler, Federico Razzoli. MariaDB Essentials. Packt: 2015. 30p.

19. Daniel Bartholomew. MariaDB vs MySQL. Monty Program: 2012. 6p.

20. Russell J.T. Dyer Learning MySQL and MariaDB. Russell J.T. Dyer: 2015. 188p.

ДОДАТОК А

Запит створення таблиць

```
CREATE TABLE `boards` (
  `boardId` int(11) NOT NULL,
  `name` varchar(40) DEFAULT NULL,
  `segmentId` int(11) DEFAULT NULL,
  `userId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE `boardsegments` (
  `segmentId` int(11) NOT NULL,
  `name` varchar(40) DEFAULT NULL,
  `cardId` int(11) DEFAULT NULL,
  `userId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE `cardboards` (
  `cardboardsId` int(11) NOT NULL,
  `cardId` int(11) DEFAULT NULL,
  `boardId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE `cards` (
  `cardId` int(11) NOT NULL,
  `name` varchar(40) NOT NULL,
  `description` varchar(40) NOT NULL,
  `isUrgent` tinyint(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE `spring_session` (
  `PRIMARY_ID` char(36) NOT NULL,
  `SESSION_ID` char(36) NOT NULL,
  `CREATION_TIME` bigint(20) NOT NULL,
  `LAST_ACCESS_TIME` bigint(20) NOT NULL,
  `MAX_INACTIVE_INTERVAL` int(11) NOT NULL,
  `EXPIRY_TIME` bigint(20) NOT NULL,
  `PRINCIPAL_NAME` varchar(100) DEFAULT NULL)
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC;
```

```
CREATE TABLE `spring_session_attributes` (
  `SESSION_PRIMARY_ID` char(36) NOT NULL,
  `ATTRIBUTE_NAME` varchar(200) NOT NULL,
  `ATTRIBUTE_BYTES` blob NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC;
CREATE TABLE `userboards` (
  `userBoardId` int(11) NOT NULL,
  `userId` int(11) DEFAULT NULL,
  `boardId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE `users` (
  `userId` int(11) NOT NULL,
  `login` varchar(40) NOT NULL,
  `passwd` varchar(40) NOT NULL,
  `isAdmin` tinyint(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `boards`
  ADD PRIMARY KEY (`boardId`),
  ADD UNIQUE KEY `segmentId` (`segmentId`),
  ADD UNIQUE KEY `userId` (`userId`);
ALTER TABLE `boardsegments`
  ADD PRIMARY KEY (`segmentId`),
  ADD UNIQUE KEY `cardId` (`cardId`),
  ADD UNIQUE KEY `userId` (`userId`);
ALTER TABLE `cardboards`
  ADD PRIMARY KEY (`cardboardsId`),
  ADD UNIQUE KEY `cardId` (`cardId`,`boardId`),
  ADD KEY `boardId` (`boardId`);
ALTER TABLE `cards`
  ADD PRIMARY KEY (`cardId`);
ALTER TABLE `spring_session`
  ADD PRIMARY KEY (`PRIMARY_ID`),
  ADD UNIQUE KEY `SPRING_SESSION_IX1` (`SESSION_ID`),
  ADD KEY `SPRING_SESSION_IX2` (`EXPIRY_TIME`),
```

```
  ADD KEY `SPRING_SESSION_IX3` (`PRINCIPAL_NAME`);
ALTER TABLE `spring_session_attributes`
  ADD PRIMARY KEY (`SESSION_PRIMARY_ID`,`ATTRIBUTE_NAME`);
ALTER TABLE `userboards`
  ADD PRIMARY KEY (`userBoardId`),
  ADD UNIQUE KEY `userId` (`userId`,`boardId`),
  ADD KEY `boardId` (`boardId`);
ALTER TABLE `users`
  ADD PRIMARY KEY (`userId`);
ALTER TABLE `cardboards`
  MODIFY `cardboardsId` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `cards`
  MODIFY `cardId` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `userboards`
  MODIFY `userBoardId` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `users`
  MODIFY `userId` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `boards`
  ADD CONSTRAINT `boards_ibfk_1` FOREIGN KEY (`segmentId`) REFERENCES
`boardsegments` (`segmentId`),
  ADD CONSTRAINT `boards_ibfk_2` FOREIGN KEY (`userId`) REFERENCES `users`
(`userId`);
ALTER TABLE `boardsegments`
  ADD CONSTRAINT `boardsegments_ibfk_1` FOREIGN KEY (`cardId`) REFERENCES
`cards` (`cardId`),
  ADD CONSTRAINT `boardsegments_ibfk_2` FOREIGN KEY (`userId`) REFERENCES
`users` (`userId`);
ALTER TABLE `cardboards`
  ADD CONSTRAINT `cardboards_ibfk_1` FOREIGN KEY (`cardId`) REFERENCES `cards`
(`cardId`),
  ADD CONSTRAINT `cardboards_ibfk_2` FOREIGN KEY (`boardId`) REFERENCES
`boards` (`boardId`);
ALTER TABLE `spring_session_attributes`
```

```
  ADD CONSTRAINT `SPRING_SESSION_ATTRIBUTES_FK` FOREIGN KEY
(`SESSION_PRIMARY_ID`) REFERENCES `spring_session` (`PRIMARY_ID`) ON DELETE
CASCADE;
ALTER TABLE `userboards`
  ADD CONSTRAINT `userboards_ibfk_1` FOREIGN KEY (`userId`) REFERENCES `users`
(`userId`),
  ADD CONSTRAINT `userboards_ibfk_2` FOREIGN KEY (`boardId`) REFERENCES
`boards` (`boardId`);
COMMIT;
```

ДОДАТОК Б

Застосування Spring Session

```
package com.example.projectmanagementapp.sessions;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableMet
hodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpCo
nfigurer;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class Sessions {

    private final UserDetailsService userDetailsService;

    @Autowired
    public Sessions(@Qualifier("userDetailsServiceImpl") UserDetailsService
userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity)
throws Exception {
        httpSecurity
                .csrf(AbstractHttpConfigurer::disable)
```

```
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/css/**", "/img/**",
"/js/**").permitAll()
                        .requestMatchers("/index/**", "/auth/**",
"/error").permitAll()
                        .anyRequest().authenticated()
                )
                .formLogin(login -> login
                        .loginPage("/auth/login")
                        .defaultSuccessUrl("/")
                        .failureUrl("/auth/login?error=true"))
                .logout(logout -> logout
                        .logoutRequestMatcher(new
AntPathRequestMatcher("/auth/logout", "GET"))
                        .clearAuthentication(true)
                        .invalidateHttpSession(true)
                        .logoutSuccessUrl("/auth/login"))
                .rememberMe(me -> me
                        .key("uniqueAndSecret")
                        .tokenValiditySeconds(172800));

        return httpSecurity.build();
    }
}
```