Міністерство освіти і науки України
«Київський Національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____»_____2023 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технології»

на тему:

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ "ОБЛІК ЧАСУ ТА ВИТРАТ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ"

Виконав: студент групи МІТ-41
Богдан ХОМІК _____

Керівник: доцент кафедри мережевих та інтернет технологій
Наталія ДАХНО _____

Київ 2023

Міністерство освіти і науки України
«Київський Національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2023 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Хомік Богдан Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи:
Розробка інформаційної системи «Облік часу та витрат на розробку програмного забезпечення»

затверджена на засіданні кафедри МІТ, протокол «_7_» грудня 2022 р. протокол № 5
2. Термін здачі закінченої роботи «30» травня 2023 р.
3. Вихідні дані до проєкту (роботи)

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-40 стор.)

Вступ

1. Аналіз існуючих інформаційних систем обліку часу та витрат на розробку програмного забезпечення. Постановка задачі.

1.1 Огляд існуючих інформаційних систем обліку часу та витрат на розробку програмного забезпечення.

1.2 Аналіз переваг та недоліків розглянутих систем обліку часу та витрат на розробку програмного забезпечення.

1.3 Постановка задачі

2. Вибір архітектури, платформи розробки, технологій та мови програмування.

2.1 Аналіз існуючих архітектурних рішень.

2.2 Аналіз протоколів обміну інформації.

2.3 Аналіз хмарних засобів, що плануються до використання.

2.4 Вибір платформи реалізації, мов програмування, фреймворків, СУБД, бібліотек.

3. Реалізація інформаційної системи "Облік часу та витрат на розробку програмного забезпечення".

3.1. Аналіз вхідної інформації та розробка моделі запропонованої системи.

3.2. Проєктування моделі даних.

3.3. Розробка серверної частини.

3.4. Створення інтерфейсу користувача.

3.5. Аналіз функціонування інформаційної системи.

Висновки

4. Перелік графічного матеріалу 8-10 слайдів

Дата видачі завдання
Керівник роботи _____ доц. Дахно Н.Б.
(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Хомік Б. С.

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	23.03.2023	
2	Розділ 1	03.04.2023	
3	Розділ 2	12.04.2023	
4	Розділ 3	26.04.2023	
5	Доповідь та слайди	25.05.2023	
6	Пояснювальна записка	30.05.2023	

Здобувач вищої освіти _____ Б.С. Хомік
(підпис)

Керівник _____ Н.Б. Дахно
(підпис)

РЕФЕРАТ

Пояснювальна записка: 60 с., 39 рис., 2 додатки, 30 джерел.

Об’єкт дослідження: управління часом та витратами на розробку програмного забезпеченя.

Предмет дослідження: інформаційна система для обліку часу та витрат на розробку програмного забезпечення.

Мета роботи (проєкту): створення інформаційної системи, яка допоможе розробникам та проєктним менеджерам ефективно відстежувати та аналізувати час та фактичні його витрати під час процесу розробки програмного забезпечення.

В роботі проведено аналіз уже існуючих інформаційних систем для обліку часу та витрат на розробку програмного забезпечення, проаналізовано їх головні переваги та недоліки. Також було проведено аналіз існуючих архітектурних рішень, протоколів обміну інформацією, хмарних сервісів, які можна використати в роботі. Окрім цього було проаналізовано та обрано найбільш підходящі сучасні технології для реалізації ІС.

Розроблено інформаційну систему для обліку часу та витрат на розробку програмного забезпечення.

Практичне значення роботи полягає у тому, що її застосування позитивно впливає на процес розробки та спрощує управління та дозволяє контролювати витрати часу на розробку ПЗ.

Результати здійснених у дипломному проєкті досліджень можуть бути використані під час розробки ПЗ як компаніями так і поодинокими особами.

Прогноз розвитку даної роботи: дану роботу можна вдосконалити шляхом введення адаптивного доступу користувачів, налаштуванням безпечної аутентифікації. Можна додати більше аналітичних функцій та розглянути інтеграцію даної системи з уже існуючими системами менеджменту проєктами.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК ЧАСУ ТА ВИТРАТ, БАЗА ДАНИХ, ПРОЦЕС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СТАТИСТИКА, УПРАВЛІННЯ РОЗРОБКОЮ.

ABSTRACT

Explanatory note: 60 pp., 39 figures, 2 appendices, 30 sources.

The object of development is time and cost management for software development.

The subject of development is an information system for accounting of time and costs for software development.

The purpose of the work (project): development of an information system that will help developers and project managers effectively track and analyze time and actual costs during the software development process.

The article conducts analysis of existing information systems to take into account time and costs of software development, analyzes their main advantages and disadvantages. Also the analysis of existing architectural solutions, protocols of exchange of information, cloud services that can be used in the work. In addition, the most appropriate state-of-the-art technologies for IP implementation were analyzed and selected.

An information system has been developed to account for the time and cost of software development.

The practical value of the work is that its application has a positive impact on the development process and simplifies the management and allows to control the time spent on software development.

The results of the research conducted in the diploma project can be used in the development of software both by companies and individuals.

Prediction of the development of this work: this work can be improved by entering adaptive user access, configuring secure authentication. More analytical functions can be added and integration of the system with existing project management systems considered.

ЗМІСТ

ВСТУП

У сучасному світі, в якому використання та розповсюдженість різного програмного забезпечення є на дуже високому рівні, швидка та якісна його розробка посідає центральне місце в багатьох сферах. В умовах жорсткої конкуренції кожен день є важливим, тому варто звести зайві витрати часу та ресурсів на мінімум, а для цього дуже зручно використовувати різні інформаційні системи для управління процесом розробки ПЗ.

Дана дипломна робота присвячена розробці інформаційної системи, яка буде відповідати за облік часу та витрат на розробку програмного забезпечення. Головна мета роботи полягає в розробці інструменту, який надасть змогу розробникам, проєктним менеджерам і просто будь-якій команді розробників, швидко та зручно управляти процесом розробки ПЗ, ефективно відстежувати та аналізувати витрати часу, який був використаний на розробку. В даній системі також буде реалізовано KANBAN дошку, на ній можна буде створювати та розміщувати завдання, також буде реалізовано зміну статусів створених завдань. Проєкт та завдання можна буде розподіляти на спринти, та фільтрувати залежно від їх приналежності.

Також в цій роботі буде представлено загальний огляд сучасних інформаційних систем для управління проєктами, а також описано процес розробки інформаційної системи для обліку часу та витрат на розробку ПЗ. Детально розглянуті аспекти реалізації системи, включаючи архітектуру, функціональні можливості, та технології, які будуть використані при розробці.

У цій роботі будуть розглянуті основні принципи, методи та інструменти, що використовуються для розробки інформаційної системи обліку часу та витрат на розробку програмного забезпечення. Буде проведений аналіз існуючих рішень та вибір оптимальних підходів для досягнення мети роботи. Крім того, буде розроблено безпосередньо інформаційна система, яка буде перевірена на практиці та проаналізована з використанням реальних даних.

Інформаційна система для обліку часу та витрат на розробку ПЗ є актуальною темою та важливою у контексті постійного розвитку сфери

програмного забезпечення та її потреби в ефективному управлінні проєктами та ресурсами.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ІНФОРМАЦІЙНИХ СИСТЕМ ОБЛІКУ ЧАСУ ТА ВИТРАТ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. ПОСТАНОВКА ЗАВДАННЯ

1.1 Огляд існуючих інформаційних систем обліку часу та витрат на розробку програмного забезпечення

На даний момент використання ІС для обліку витрат та часу на розробку ПЗ є просто невід'ємною складовою розробки. Використання Agile методологій розробки таких як, такі як Scrum, Kanban та Extreme Programming (XP) просто не можуть обійтися без розподілення обов’язків для всіх учасників проєкту та відстеження статусів уже створених завдань дуже зручно використовувати ІС. Вони дозволяють розрахувати плановий час та витрати на розробку тих чи інших функцій, розподіляти та призначати завдання конкретним співробітникам і в цілому вести віддалене управління цілим проєктом. Є можливість формувати звіти по використанню часу та витрат за певні проміжки часу по цілому проєкту або можна сформувати їх лише за конкретними спринтами. Це все робить процес управління розробкою програмного забезпечення набагато зручнішим та менш ресурсозатратним, як зі сторони керівництва проєкту, так і співробітників.

Розглянемо найбільш популярні інформаційні системи.

Безсумнівно, одним із найбільш популярних та потужних ІС для обліку часу та витрат на розробку ПЗ є Jira[1]. Jira [2] – програмне забезпечення, розроблене компанією Atlassian, яке використовується для управління проєктами, відстеження проблем та помилок. Jira дозволяє командам створювати та відстежувати завдання, призначати їх членам команди та контролювати їх прогрес протягом усього життєвого циклу проєкту.

Jira особливо популярна в групах розробників програмного забезпечення, але вона може бути використана в будь-якому проєкті, де потрібно відстежувати завдання та керувати ними. Вона надає такі функції, як гнучкі

дошки, налаштовувані робочі процеси, керування проєктами та портфоліо і безпосередньо звітування.

Jira це веб-застосунок, і до неї можна отримати доступ з будь-якого пристрою з підключенням до Інтернету. Дана ІС також має мобільні додатки для iOS та Android пристроїв.

Іншими популярними альтернативами Jira є такі ІС [3] як : Trello, Asana, Basecamp, ClickUp, Monday.com, Binfire, Pivotal Tracker, Backlog та ін.

Розглянемо деякі із них.

Binfire [4] — це веб-програмне забезпечення для керування проєктами, яке пропонує низку функцій, що допомагають командам ефективніше керувати своїми проєктами. Деякі з ключових цілей Binfire це: керування завданнями, інструменти для співпраці, обмін файлами, відстеження часу та аналітикуа за проєктуом. Основні функції: керування робочим простором, персональна панель, календар робочого простору, інтерактивна дошка, шаблон проєкту, звіт про стан проєкту, інтерактивна діаграма Ганта, потік активності та діаграма вигораяння.

Basecamp [5]: Basecamp — популярний інструмент керування проєктами, який існує вже понад 20 років. Він вирішує низку задач, зокрема відстеження завдань, командну співпрацю та планування проєктів, і популярний серед малих і середніх команд. Основні функції: платформа обміну повідомленнями, універсальний пошук, один центр для обміну документами та файлами, просте керування завданнями (чудово підходить для списків справ), питання про автоматичну реєстрацію для стендапів, портал для клієнтів і замовників для перегляду завдань і спілкування, груповий чат і прямі повідомлення, звіт по завданнях.

Trello [6]: Trello – це візуальний інструмент управління проєктами, який використовує дошки, списки та карти для організації та пріоритезації завдань. Це інтуїтивно зрозумілий та простий у використанні, що робить його популярним вибором для невеликих команд та окремих осіб. Основні функції: теги, мітки та категорії, drag and drop картки, діаграми прогресу, встановлення

нагадувань, відображення Kanban дошки, призначення завдань, багато інтеграцій.

Asana [7]: Asana – це універсальний інструмент управління проєктами, який пропонує широкий спектр можливостей, включаючи відстеження завдань, співпрацю команд та планування проєктів. Він є популярним серед команд усіх розмірів, від стартапів до організацій на рівні підприємства. Основні функції: декілька робочих областей, стрічка активності високого рівня, додатки для зручного ознайомлення, відображення календаря, чат у реальному часі та співпраця для кожного завдання, дозволи проєкту, спеціальні поля, відображення дошки для гнучкого керування проєктами.

Monday.com [8]: Monday.com — це інструмент для керування командою та співпраці, який пропонує гнучку та кастомізовану платформу для відстеження проєктів і робочих процесів. Він відомий своїм інтуїтивно зрозумілим інтерфейсом і зручним дизайном. Основні функції: гнучкі кастомізовані дошки, спільна робоча область, автоматизація, відстеження часу, звітування та аналітика, інтеграція з іншими додатками, кастомізована робоча область, декілька представлень (календар, діаграма Ганта, Канбан відображення та інші), високий рівень безпеки.

ClickUp [9]: ClickUp — це комплексний інструмент управління проєктами, який пропонує вирішення широкого спектру задач, включаючи керування завданнями, командну співпрацю та планування проєктів. Він відомий своїм кастомізованим інтерфейсом і широкою інтеграцією. Основні функції: кастомізовані завдання, Agile дошки, відстеження часу, інтеграція з іншими додатками, декілька відображень (календар, діаграма Ганта, Канбан відображення та інші), інструменти для командної роботи, кастомні поля та шаблони, відстежування цілей та прогресу, інформаційні панелі та звітність, високий рівень безпеки.

В даному підрозділі було проаналізовано такі популярні ІС для обліку часу та витрат на розробку ПЗ як Jira, Trello, Asana, Monday.com, Binfire, Basecamp та ClickUp. Також було розглянуто їх основні функції.

1.2 Аналіз переваг та недоліків розглянутих інформаційних систем обліку часу та витрат на розробку програмного забезпечення

В попередньому підрозділі ми розглянули деякі ІС обліку часу та витрат на розробку програмного забезпечення, тепер проаналізуємо їх переваги та недоліки.

Jira. Переваги:

- Кастомізація: Jira має широкі можливості налаштування, що дозволяє командам налаштовувати інструмент відповідно до своїх конкретних потреб і робочих процесів.
- Орієнтація на Agile: Jira розроблено з урахуванням методології Agile, що робить його ідеальним інструментом для команд, які практикують Agile-розробку.
- Інтеграція: Jira добре інтегрується з іншими інструментами та платформами, такими як Git, Jenkins, Confluence тощо.
- Співпраця: Jira сприяє співпраці між членами команди та дозволяє їм ефективно спілкуватися за допомогою коментарів, сповіщень і нагадувань.
- Звітність: Jira надає ряд варіантів звітів, які допомагають командам відстежувати свій прогрес і визначати сфери, які потрібно вдосконалити.

Недоліки:

- Складність: Jira може бути складною, що ускладнює освоєння новими користувачами і ефективне використання всіх її функцій.
- Вартість: використання хмарного рішення Jira може бути дорогим, особливо для невеликих команд або організацій з обмеженим бюджетом, в той же час підтримка серверного рішення для самостійного використання поступово згортається компанією Atlassian.
- Забирає багато часу: встановлення та конфігурація Jira може займати багато часу, що може вплинути на продуктивність команди.

- Кастомізація: кастомізація водночас як і перевага так і недолік, оскільки вона вимагає значної кількості часу та зусиль для налаштування та обслуговування.
- Навчання: для ефективного використання Jira потрібне навчання, яке може стати тягарем для команд, яким і так бракує часу.

Binfire. Переваги:

- Колаборація: Binfire надає низку інструментів для співпраці, таких як чат у реальному часі, відеоконференції та спільне використання екрана.
- Керування завданнями. Можна створювати, призначати та відстежувати завдання, встановлювати терміни виконання та встановлювати залежності завдань.
- Обмін файлами: Binfire дозволяє командам легко обмінюватися файлами, документами та іншими матеріалами, пов’язаними з проєктом. Він також забезпечує контроль версій, тому члени команди можуть відстежувати зміни та співпрацювати над документами в режимі реального часу.
- Відстеження часу: Binfire пропонує інструменти відстеження часу, які дозволяють членам команди реєструвати час, який вони витрачають на певні завдання, що дуже зручно для управління проєктами.

Недоліки:

- Крута крива навчання: Binfire має широкий набір функцій і може зайняти деякий час для навчання. Він може бути не таким інтуїтивно зрозумілим, як деякі інші інструменти керування проєктами.
- Вища ціна в порівнянні з іншими засобами. Немає безкоштовних планів, доступ до деяких функцій може бути значно дорожчим.
- Обмежена інтеграція: Binfire пропонує інтеграцію з деякими популярними інструментами, такими як Google Drive і Dropbox, але він може не пропонувати стільки інтеграцій, скільки деякі інші інструменти управління проєктами.

BaseCamp. Переваги:

- Проста і зрозуміла система управління проєктами
- Хороші функції спілкування та співпраці

- Пропонує кілька інструментів і функцій на одній платформі
- Хороша підтримка клієнтів і ресурси

Недоліки:

- Не вистачає деяких розширених функцій керування проєктами для великих або складних проєктів
- Обмежені можливості налаштування для окремих користувачів
- Не такий гнучкий, як інші інструменти для різних типів проєктів або робочих процесів

Trello. Переваги:

- Простий у використанні та інтуїтивно зрозумілий інтерфейс.
- Проста і гнучка система управління завданнями.
- Пропонує різні інтеграції та розширення.
- Безкоштовний план доступний для невеликих команд і окремих осіб.

Недоліки:

- Обмежена функціональність для великих та більш складних проєктів.
- Немає розширених функцій звітності та аналітики.
- Робоча область може бути захаращена занадто великою кількістю карток і дощок.

Asana. Переваги:

- Широкий набір функцій для керування проєктами, командної співпраці та відстеження завдань.
- Гнучка та настроювана система для різних типів проєктів.
- Хороші можливості інтеграції з іншими інструментами та платформами.
- Пропонує розширені функції звітності та аналітики.

Недоліки:

- Крута крива навчання для нових користувачів.
- Занадто багато можливостей і опцій, що може бути приголомшливим для нових користувачів.
- Вища ціна в порівнянні з іншими засобами.

Monday.com . Переваги:

- Зручний і візуально привабливий інтерфейс.

- Універсальна та настроювана платформа для управління проєктами та командами.
- Хороші можливості автоматизації для повторюваних завдань.
- Хороша підтримка клієнтів і ресурси.

Недоліки:

- Ціни можуть бути вищими для великих команд або організацій на рівні підприємства.
- Не вистачає деяких додаткових функцій для управління складними проєктами.
- Обмежені можливості налаштування для окремих користувачів.

ClickUp. Переваги:

- Комплексна та настроювана система управління проєктами.
- Широкий набір функцій для різних типів проєктів і команд.
- Хороші можливості інтеграції з іншими інструментами та платформами.
- Хороші можливості автоматизації для повторюваних завдань.

Недоліки:

- Занадто багато можливостей і опцій може бути занадто складним для освоєння новим користувачам.
- Крута крива навчання для нових користувачів.
- Вища ціна порівняно з деякими іншими інформаційними системами.

В даному підрозділі було розглянуто переваги та недоліки одних з найпопулярніших інформаційних систем обліку часу та витрат на розробку програмного забезпечення. Для нашого проєкту варто зосередитися на простоті та зручності функціоналу. Додаток буде легко освоїти та працювати і при цьому він буде виконувати зазначені функції, але також важливо розробити його гнучким, щоб була подальша можливість імплементації нових функцій.

1.3 Постановка завдання

Предметною областю даної кваліфікаційної роботи є автоматизація обліку часу та витрат на розробку програмного забезпечення. Розроблювана інформаційна система повинна також і реалізувати інші функції управління розробкою. Необхідно, щоб була можливість створення проєктів та спринтів і звісно зберігання інформації про них. Обов'язковою є реалізація можливості реєстрації задач, зберігання інформації про них, переведення їх з одного стану до іншого і також необхідно зберігати час попередньо зазначений на розробку та фактичний час розробки для ведення обліку. Також необхідно реалізувати KANBAN, SCRUM або AGAIL дошку, на якій створені завдання будуть зберігатися. Важливо реалізувати створення різної кількості користувацьких колонок для зручності. Також система повинна розраховувати витрачений час на проєкт за певними часовими рамками.

Окрім вищеописаних функцій, можна реалізувати такі як адаптивний доступ до функцій(залежно від ролі працівника), та призначення завдань конкретному працівнику. Це зробить систему більш гнучкою та зручною для використання.

РОЗДІЛ 2. ВИБІР АРХІТЕКТУРИ, ПЛАТФОРМИ РОЗРОБКИ, ТЕХНОЛОГІЙ ТА МОВИ ПРОГРАМУВАННЯ

2.1 Аналіз існуючих архітектурних рішень

MVC(Model-View-Controller) [10] – це шаблон проєктування програмного забезпечення, який зазвичай використовується для розробки інтерфейсів користувача. Даний шаблон розділяє основну логіку програми на три взаємопов'язані елементи. Це робиться для того, щоб відокремити внутрішнє представлення інформації від того, як інформація подається та приймається користувачем.

Модель – представляє дані програми та бізнес-логіку та взаємодіє з базою даних для виконання операцій з даними.

Відображення використовуються для візуалізації деякої частини моделі у вигляді інтерфейсу користувача.

Контролер – клас, який діє як посередник між моделлю та представленням(відображенням), отримуючи дані від користувача через відображення і в залежності від результатів обробки відправляє користувачеві певне виведення, наприклад, у вигляді відображення.

Архітектура MVC дозволяє розподілити проблеми, полегшуючи підтримку та зміну коду. Розробники можуть працювати над окремими компонентами, розробляючи як front-end так і back-end. І якщо ми розробляємо front-end, то ми можемо тестувати відображення незалежно від контролера.

Клієнт-серверна архітектура [11] – це модель обчислень, у якій клієнтська програма, запущена на одному комп'ютері, надсилає запити серверній програмі, запущеній на іншому комп'ютері, зазвичай через комп'ютерну мережу. Серверна програма обробляє запити та відповідає даними або послугами, які може використовувати клієнтська програма.

У клієнт-серверній архітектурі клієнтська програма зазвичай відповідає за представлення інтерфейсу користувачу, а сервер в свою чергу надає певні послуги(наприклад файловий сервер – буде обслуговувати передачу даних).

Клієнт і сервер спілкуються один з одним за допомогою мережевих протоколів. Клієнтом може бути настільна програма, веб-браузер, мобільна програма або будь-яка інша програма, яка може спілкуватися з сервером.

Архітектура клієнт-сервер забезпечує розподілене обчислення, що означає, що обчислювальна потужність і ресурси можуть бути розподілені між кількома комп'ютерами в мережі. Це дозволяє досягти кращих масштабованості, надійності та продуктивності.

Монолітна архітектура [12] – це шаблон архітектури програмного забезпечення, у якому вся програма побудована як єдине неподільне ціле. У цій архітектурі всі компоненти програми, такі як інтерфейс користувача, логіка програми та сховище даних, тісно пов'язані та інтегровані в єдину виконувану кодову базу. Це може спростити розробку та тестування програми, оскільки всі компоненти знаходяться в одному місці. Однак це також може ускладнити масштабування програми, оскільки масштабування вимагає масштабування всієї програми, а не лише компонентів, які потребують більше ресурсів.

Монолітні архітектури часто використовуються для додатків малого та середнього розміру, де простота і легкість розробки переважають проблеми масштабованості. Однак у міру того, як додатки ростуть і стають складнішими, може стати складніше підтримувати та масштабувати монолітні архітектури.

Мікросервіси [13] – це архітектурний шаблон програмного забезпечення, який орієнтований на окремі сервіси. Основна особливість та перевага даного паттерну – написання окремих та повністю незалежних невеликих сервісів. Кожен сервіс відповідає за певну бізнес-логіку та взаємодії з іншими сервісами через конкретні API.

Основними перевагами мікросервісної архітектури є модульність. Це робить програму легшою для розуміння, розробки та тестування оскільки розробники працюють з окремим функціоналом, а не цілим додатком. Масштабованість – кожен мікросервіс розробляється окремо від загального додатку, це в свою чергу дозволяє розробляти все нові і нові сервіси, при цьому не ламаючи основний функціонал. Інтеграція різнорідних систем і застарілих

систем – на даний момент один із можливих способів модернізації старих монолітних додатків, шляхом простого підв’язування мікросервісів.

В кожної з архітектур є свої переваги та недоліки. Монолітна архітектура не є підходящою, оскільки вона не підтримує масштабування як такого, хоча і є зручною та простою для розробки, проте в майбутньому для даного додатку дійсно можна значно розширити функціонал, що робить даний патерн не найкращим вибором. Також даний додаток не потребує розділення функціоналу на клієнтську програму та серверну програму, що робить однойменну архітектуру не підходящою. MVC-архітектура та мікросервісна архітектура підходять для даного проєкту, але у MVC є деякі переваги.

Переваги MVC [14]:

- Простота розробки – розробляти ПЗ з використанням MVC може бути простіше і швидше, оскільки компоненти знаходяться в тій же програмі.
- Менша складність – у мікросервісній архітектурі необхідно налаштовувати взаємодію кожного сервісу один з одним, це доволі складно та потребує додаткового часу на розробку.
- Швидкість розробки – дана перевага витікає із попереднього пункту. Час який витрачається на налаштування взаємодії сервісів в MVC можна витратити на розробку функціоналу.
- Більш просте тестування: тестування ПЗ MVC додатків може бути простіше оскільки всі компоненти знаходяться в одній програмі, а мікросервіси необхідно тестувати окремо.

Отже для даного проєкту найбільш доцільно буде використовувати MVC-архітектуру, вона доволі проста та зручна і є найбільш підходящою.

2.2 Аналіз протоколів обміну інформації

Для передачі даних у відкритому робочому середовищі необхідне використання протоколу обміну інформації — HTTPS. Для використання у локальних мережах підприємства та для тестування допускається використання протоколу HTTP.

HTTP (HyperText Transfer Protocol) [15] – це протокол передачі гіпертексту прикладного рівня в моделі OSI. HTTP є основою передачі даних для всесвітньої павутини, де гіпертекстові документи містять гіперпосилання на інші ресурси, до яких користувач може легко отримати доступ, наприклад, клацнувши мишею або торкнувшись екрана у веб-браузері. HTTP забезпечує засоби для передачі різноманітного контенту, такого як HTML-сторінки, зображення, аудіо та відеофайли, а також для взаємодії між клієнтами та серверами.

HTTPS (HTTP Secure) [16] – це також протокол передачі даних, який використовується в Інтернеті для забезпечення безпеки під час передачі даних між веб-серверами та веб-браузерами шляхом шифрування. HTTPS використовує шифрування SSL або TLS для захисту конфіденційності та цілісності даних. За допомогою HTTPS, інформація, яка передається між веб-браузером та веб-сайтом, шифрується, що ускладнює доступ до неї від сторонніх осіб. Це особливо важливо для передачі конфіденційної інформації, такої як особисті дані, банківські реквізити, паролі та інші чутливі дані.

Основні переваги використання HTTPS порівняно з HTTP включають наступне:

- Безпека: HTTPS забезпечує шифрування даних, що передаються між веб-сайтом та користувачем, тому що дані не можуть бути перехоплені чи змінені посторонніми особами.
- Довіра користувачів: Використання HTTPS показує користувачам, що веб-сайт, на якому вони перебувають, дійсно той, за який себе видає, і що вони можуть довіряти цьому сайту.

- Покращення SEO: Google оголосив, що використання HTTPS є фактором ранжування в пошуковій системі. Це означає, що веб-сайти з HTTPS-захистом можуть мати перевагу над сайтами без захисту, що допомагає в них з'являтися на більш високих позиціях у результатах пошуку.
- Підтримка HTTP/2: HTTP/2, оновлена версія протоколу HTTP, підтримується лише на HTTPS-захищених веб-сайтах. HTTP/2 забезпечує швидшу швидкість завантаження сторінок.

Для налаштування HTTPS необхідно придбати SSL сертифікат після чого його необхідно встановити на веб сервері, для цього можна звернутися до провайдера і уже після цього налаштувати HTTPS на сервері.

Отже, HTPPS є завжди більш пріоритетним ніж HTTP, і для впровадження даного додатку, та його експлуатації є необхідним налаштування HTTPS, але для тестової версії ІС це не є обов'язковим і його налаштуванням можна знехтувати задля збереження часу та ресурсів.

2.3 Аналіз хмарних засобів, що плануються для використання

Для зберігання програмного коду та управління версіями додатку розробленого в ході даної роботи буде використано хмарний сервіс такий як **GitHub**. GitHub – хмарний сервіс для розробки та управління версіями за допомогою Git, який дозволяє розробникам зберігати, керувати та ділитися своїми сховищами коду [17]. Сервіс надає графічний веб-інтерфейс для створення, керування та спільної роботи над проєктами розробки програмного забезпечення. GitHub дозволяє розробникам відстежувати зміни у своєму коді та співпрацювати з іншими розробниками. Він широко використовується окремими особами та компаніями для розміщення проєктів з відкритим кодом, а також для внутрішньої співпраці над власним програмним забезпеченням. GitHub також пропонує різні функції, такі як відстеження помилок, інструменти керування проєктами, перевірка коду та документація. GitHub став однією з

найпопулярніших платформ для розробки програмного забезпечення, якою користуються мільйони розробників у всьому світі.

Ще один хмарний засіб, який планується використати для розгортання додатку та бази даних – **Microsoft Azure**. Це хмарна обчислювальна платформа під керуванням Microsoft, яка пропонує доступ, керування та розробку програмного забезпечення і окремих сервісів через глобальні центри обробки даних [18]. Він надає низку можливостей, зокрема програмне забезпечення як послуга (SaaS) , платформа як послуга (PaaS) та інфраструктура як послуга (IaaS). Azure дозволяє розробникам створювати сайти за допомогою ASP.NET, PHP, Node.js, Java або Python, які згодом можуть бути розгорнуті за допомогою FTP, Git, Mercurial, Team Foundation Server або ж бути завантаженими через портал користувача.

Також у Azure є послуги зберігання, там ми можемо розгорнути базу даних. Сервіс таблиць дозволяє програмам зберігати структурований текст у розділених колекціях сутностей, доступ до яких здійснюється за допомогою ключа розділу та первинного ключа. База даних SQL Azure – це повністю кероване ядро СУБД, що надається за моделлю "платформа як послуга" (PaaS), яке автоматизує більшість функцій управління базами даних, таких як оновлення, виправлення, резервне копіювання та моніторинг [19]. База даних SQL Azure заснована на останній стабільній версії ядра СУБД Microsoft SQL Server (дана база даних буде використана в даній роботі). Тому у користувачів є можливість використовувати розширені функції обробки запитів, наприклад, технології високопродуктивних обчислень у пам'яті та інтелектуальну обробку запитів.

В даному підрозділі було проаналізовано хмарні засоби, які будуть використані в даній кваліфікаційній роботі та було зазначено їх конкретне використання.

2.4 Вибір платформи реалізації, мов програмування, фреймворків, СУБД, бібліотек.

Платформа реалізації. Для побудови заданої інформаційної системи в якості платформи реалізації було прийняте рішення обрати ASP.NET MVC. ASP.NET – це платформа веб-розробки, розроблена Microsoft, яка дозволяє розробникам створювати динамічні веб-додатки та служби за допомогою мов програмування .NET, C# та багато інших [20]. ASP.NET підтримує низку моделей програмування для створення веб-додатків таких як ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web Pages, ASP.NET Web API та інші. Дана платформа дозволяє швидшу розробку ніж на інших скриптових мовах програмування. Також дана платформа має хорошу підтримку зі сторони фреймворків та бібліотек для розробки mvc додатків (в ході розробки для спрощення роботи з даними буде використано Microsoft EntityFramework).

Мови програмування. Для розробки back-end частини ASP.NET підтримує такі мови програмування: C#, Visual Basic .NET, F#, C++ та J#. В нашому додатку буде використано C#. C# – об'єктно-орієнтована мова програмування з доволі простим та зрозумілим синтаксисом. Вона була розроблена компанію Microsoft, тому мова отримує хорошу підтримку та активно розвивається. Також для даної мови є безліч фреймворків та бібліотек, що дозволяє швидко розробляти багатофункціональні та складні веб-додатки. Також варто відмітити, що дана мова програмування є повністю інтегрованою в платформу .NET, що відкриває доступ до всіх засобів розробки .NET.

Для розробки front-end частини будуть використані такі мови програмування:

- HTML – це мова розмітки, яка використовується для створення структури веб-сторінок. Також буде використаний синтаксис Razor (після символу @ відповідно до синтаксису ми можемо використовувати вирази коду на мові C#).
- CSS – це мова стилів, яка використовується для задання зовнішнього вигляду веб-сторінок.

- JavaScript – це мова програмування, яка використовується для створення динамічних інтерфейсів та взаємодії з користувачем.

Фреймворки. Для роботи з даними в ASP.NET Core MVC зачасту використовується EntityFramework. Entity Framework – це структура об'єктно-реляційного відображення (ORM), яка є частиною платформи .NET, розробленої Microsoft [22]. Це інструмент, який використовується для доступу до даних і керування ними в додатках .NET.

Entity Framework дозволяє розробникам працювати з базами даних за допомогою об'єктів .NET, що усуває потребу в коді SQL низького рівня. Він забезпечує операції створення, читання, оновлення та видалення (CRUD) даних, що зберігаються в реляційній базі даних, за допомогою високорівневого об'єктно-орієнтованого API.

Фреймворк відображає схему бази даних у набір класів, які представляють модель даних, і дозволяє розробникам працювати з цими класами замість безпосередньої взаємодії з базою даних. Entity Framework підтримує роботу з різноманітними СУБД, включаючи Microsoft SQL Server, Oracle, MySQL і PostgreSQL та інші.

Таким чином, Entity Framework — це потужний інструмент, який спрощує доступ до даних і керування ними в програмах .NET, забезпечуючи об'єктно-орієнтований рівень абстракції над основною базою даних.

Bootstrap. Bootstrap - це популярний CSS фреймворк з відкритим кодом, який містить готові HTML та CSS шаблони та додаткові розширення для JavaScript, даний фреймворк дозволяє швидко та легко розробити привабливий користувацький інтерфейс [23]. Bootstrap буде доданий до проекту за замовчуванням, оскільки він входить в шаблон додатку ASP.NET MVC версії 6.0. Його стилі будуть використані для стилізації модальних вікон та інших базових елементів створеного шаблону.

СУБД. За моделлю організації даних БД поділяють на три групи: ієрархічна, мережева та реляційна моделі даних. Ієрархічна модель [24] будується за принципом ієрархії даних, тобто один із об'єктів є головним, а решта об'єктів підпорядковуються йому. У кожного об'єкта за виключенням

головного (у нього немає предків) є лише один предок, проте може мати декілька нащадків. Це і є головною характеристикою даної моделі між головним та підпорядкованими об'єктами встановлюється взаємозв'язок "один до багатьох". Перевагою моделі є просто візуалізації та розуміння, схема і правда є дуже легкою.

Недоліки:

- Складно та незручно додавати інформацію про новий об’єкт та видаляти інформацію про старий;
- Неможливо реалізувати зв’язки виду багато до багатьох;
- Повільний доступ до даних з нижніх шарів ієрархії;

Мережева модель [25]. Основною відмінністю між мережевою та ієрархічною моделями є те, що в даній моделі у кожного нащадка може бути будь-яка кількість предків. У мережевій моделі кожен об'єкт може бути зв'язаним з будь-яким іншим. Отже, у даній моделі кожен об'єкт може брати участь у будь-якій кількості відношень. Таким чином мережева модель даних може обробляти зв’язки «багато-до-багатьох», при тому, що ієрархічна модель даних цього не робила – це є головною перевагою даної моделі.

Недоліки:

- Мережева модель є набагато складнішою за ієрархічну, тому її складніше підтримувати та обслуговувати.
- Хоч і мережева модель є значно гнучкішою, все ще залишаються проблеми, які не можна вирішити.
- Ще одним наслідком складністю архітектури є те, що працівник, який буде працювати з нею, дуже добре повинен розуміти структуру, щоб правильно впроваджувати оновлення.

Реляційна модель [26] (RM) представляє базу даних як набір відносин. Відношення – це не що інше, як таблиця значень. Кожен рядок у таблиці представляє набір пов’язаних значень даних. Ці рядки в таблиці позначають реальну сутність або зв’язок. Ім’я таблиці та назви стовпців допомагають інтерпретувати значення значень у кожному рядку. Дані представлені у вигляді

набору відношень. У реляційній моделі дані зберігаються у вигляді таблиць. Однак фізичне зберігання даних не залежить від способу їх логічної організації.

Переваги реляційної моделі:

- простота і доступність для розуміння користувачем. Єдиною використовуваною інформаційною конструкцією є «таблиця»;
- суворі правила проєктування, які базуються на математичному апараті;
- повна незалежність даних. Зміни в прикладній програмі при зміні реляційної БД мінімальні;
- для організації запитів і написання прикладного ПЗ немає необхідності знати конкретну організацію БД у зовнішній пам'яті.

Недоліки реляційної моделі:

- далеко не завжди предметна область може бути представлена у вигляді «таблиць»;
- в результаті логічного проєктування з'являється множина «таблиць». Це призводить до труднощів розуміння структури даних;
- БД займає відносно багато зовнішньої пам'яті;
- відносно низька швидкість доступу до даних.

У реляційних базах даних дані та взаємозв’язки між ними реалізуються за допомогою однорідних таблиць. Проводити вибірку, групування та інші маніпулювання даними, у модулі даного типу набагато швидше та зручніше ніж у мережевій та ієрархічній моделі. Модель двовимірної таблиці дозволяє звертатися до даних як по рядках, так і по стовпцях, що є значною перевагою.

Кожна таблиця має такі властивості:

- кожний елемент таблиці являє собою один елемент даних;
- усі стовпці в таблиці однорідні; це означає, що елементи стовпця мають однакову природу;
- у кожного стовпця таблиці є своє унікальне ім’я;
- у таблицях не можуть зберігатися два однакові записи.

Однією з переваг даної моделі вважаються суворі правила проєктування, які базуються на математичному апараті. Внаслідок простоти організування

системи, єдиного структурного виду елементів – таблиць, в даній моделі доволі легко відбувається групування та сортування елементів, пошук необхідних нам записів та представлень.

Отже проаналізувавши всі недоліки та переваги різних моделей даних, можна стверджувати, що реляційна модель є найбільш оптимальною, завдяки своїй структурі вона дозволить реалізувати весь необхідний нам функціонал та дозволить це зробити з найменшим використанням ресурсів.

Серед реляційних баз даних можна виділити такі як Microsoft SQL Server, MySQL, PostgreSQL, OracleSQL та інші. Однак, якщо йдеться про простоту використання, то найбільш зручним варіантом для розробки ASP.NET MVC проєкту буде Microsoft SQL Server. Microsoft SQL Server є надійною, швидкою та потужною реляційною СУБД, яка працює на платформі Windows, що ідеально підходить для розробки ASP.NET MVC проєктів. Більше того, SQL Server має дуже зручний інтерфейс для розробки та керування базами даних, а також досить простий у використанні. Дана СУБД задовольняє всі умови та чудово підходить для нашої ІС.

Бібліотеки. Для порівняння очікуваних та фактичних витрат на розробку буде використана JS бібліотека Chart.js. Chart.js – безкоштовна бібліотека JavaScript із відкритим вихідним кодом, яка використовується для візуалізації даних [27]. Вона підтримує багато типів діаграм, таких як стовпчаста, лінійна, бульбашкова, точкова, радарна та багато інших. На даний момент вона підтримується активними членами спільноти. Дана бібліотека є однією з найбільш популярних і вважається однією з кращих бібліотек для візуалізації даних.

jQuery. jQuery - це популярна та потужна JavaScript бібліотека з відкритим кодом. Синтаксис даної бібліотеки розроблений для спрощення взаємодії з елементами на сторінці шляхом спрощення взаємодії з об’єктами DOM, обробкою подій, створенню анімацій та розробкою AJAX-застосунків[28]. jQuery за замовчуванням підключена до шаблонного додатку ASP.NET MVC версії 6.0.

В даному підрозділі було проаналізовано та обрано: платформу реалізації, мови програмування, фреймворки, СУБД та бібліотеки, які будуть використані для розробки додатку.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ "ОБЛІК ЧАСУ ТА ВИТРАТ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ"

3.1 Аналіз вхідної інформації та розробка моделі запропонованої системи.

Виходячи з проаналізованої нами в підрозділі 1.3 інформації, можна виявити, що наша система повинна містити інформацію принаймні про 6 сутностей. Проведемо їх детальний аналіз. Перша сутність – користувач системи. У користувача повинні бути такі атрибути як id, email, пароль, ім’я та прізвище. Наступні дві сутності – проєкт та спринт. У них повинні бути такі поля як id, назва та опис. У проєкта також повинен бути код, а у спринта в свою чергу повинні бути атрибути початку та закінчення, оскільки спринт зазвичай обмежений відносно короткими часовими рамками. Наступна – учасник проєкту (розробники, адміністратори та інші, які стосуються лише конкретного проєкту). У нього повинні бути id, id користувача системи та id проєкту, в якому призначений користувач, а також роль. Дана сутність необхідна для налаштування зв’язку багато-до-багатьох між проєктом та користувачем, оскільки як у одного проєкту може бути безліч розробників, так і у користувача може бути багато проєктів. Ще одна важлива сутність це стовпчик (колонка). Дана сутність буде служити як зберігання статусів завдань. За назвою колонки можна буде визначити на якому етапі розробки знаходиться завдання (буде описано наступним). У колонки буде: id, назва та id проєкту. Остання необхідна сутність – завдання. У завдання будуть такі атрибути як: id, id-колонки, id - спринту, заголовок, опис, id призначеного користувача та id користувача, який здав заняття, та атрибут для зберігання дати створення завдання. Також необхідна наявність полів попередньо необхідного та фактичного часу використаного на реалізацію.

Тепер на основі вище сказаного стає можливим побудувати концептуальну модель. Концептуальна модель бази даних є високорівневим представленням даних, яке використовується для опису бізнес-сутностей, їх

взаємозв'язків і правил, що регулюють ці зв'язки [29]. Це абстрактна модель, яка служить основою для подальшого проєктування фізичної бази даних.

Концептуальна модель бази даних зосереджується на сутностях бізнесу і зв'язках між ними, а не на конкретних деталях реалізації бази даних. Представимо нашу концептуальну модель за допомогою діаграми сутність-зв'язок (ERD), де сутності відображаються як прямокутники, атрибути – овалами, зв’язки – ромбами. ER-модель – модель даних, яка дозволяє описувати концептуальні схеми предметної області за допомогою узагальнених конструкцій блоків. За її допомогою можна виділити ключові сутності та позначити зв’язки, які можуть встановлюватися між даними сутностями. Розробимо таку діаграму діаграму (рисунок 3.1.1):

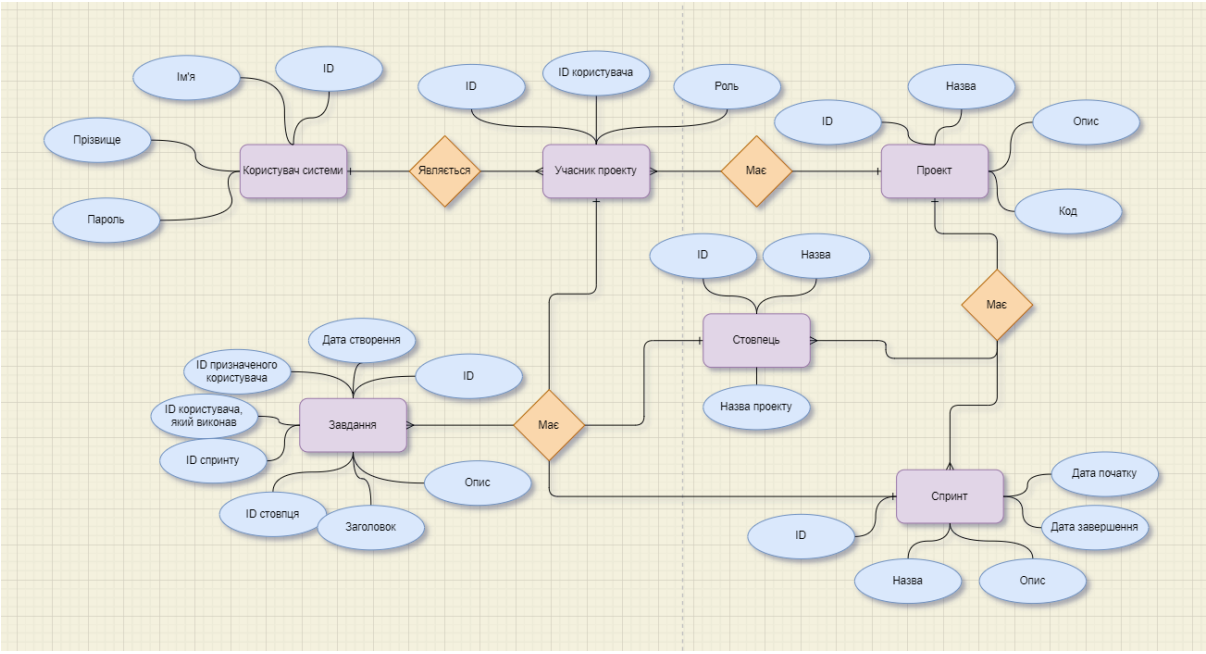


Рисунок 3.1.1 – ER-діаграма інформаційної системи

Також інформаційна система повинна надавати можливість редагування, додавання, видалення та перегляду всіх створених екземплярів сутностей. Повинна бути можливість створення проєктів з бажаною кількістю стовпців одночасно, а не незалежно один від одного, повинна бути реалізована дошка, в якій будуть відображатися завдання, залежно від їх статусів та можливість їх зміни. Можливість призначення завдань користувачам та їх здачу. Також необхідні функції створення, передчасного завершення спринтів та

фільтрування завдань залежно від обраного спринту, та забезпечення зберігання інформації про попередньо-зазначений та фактично витрачений час на виконання завдання для побудови статистики.

3.2 Проєктування моделі даних

У даному підрозділі виконаємо нормалізацію бази даних та створимо логічну та фізичну моделі інформаційної системи на основі ER-діаграми, створеної у попередньому підрозділі. Логічна модель встановлює структуру елементів бази даних зв'язків між ними. На відміну від концептуальної моделі, логічна модель включає в себе фактичні сутності, атрибути, та відношення (зовнішні ключі). Логічна схема бази даних (рисунок 3.2.1) допомагає загальному розумінню елементів бізнес-даних та забезпечує основу і подальший для проєктування бази даних.

Дана схема, як і ER-діаграма з першого розділу, створювалася за допомогою онлайн застосунку app.diagram.net. Він легкий в освоєнні, обширний за функціоналом та безкоштовний.

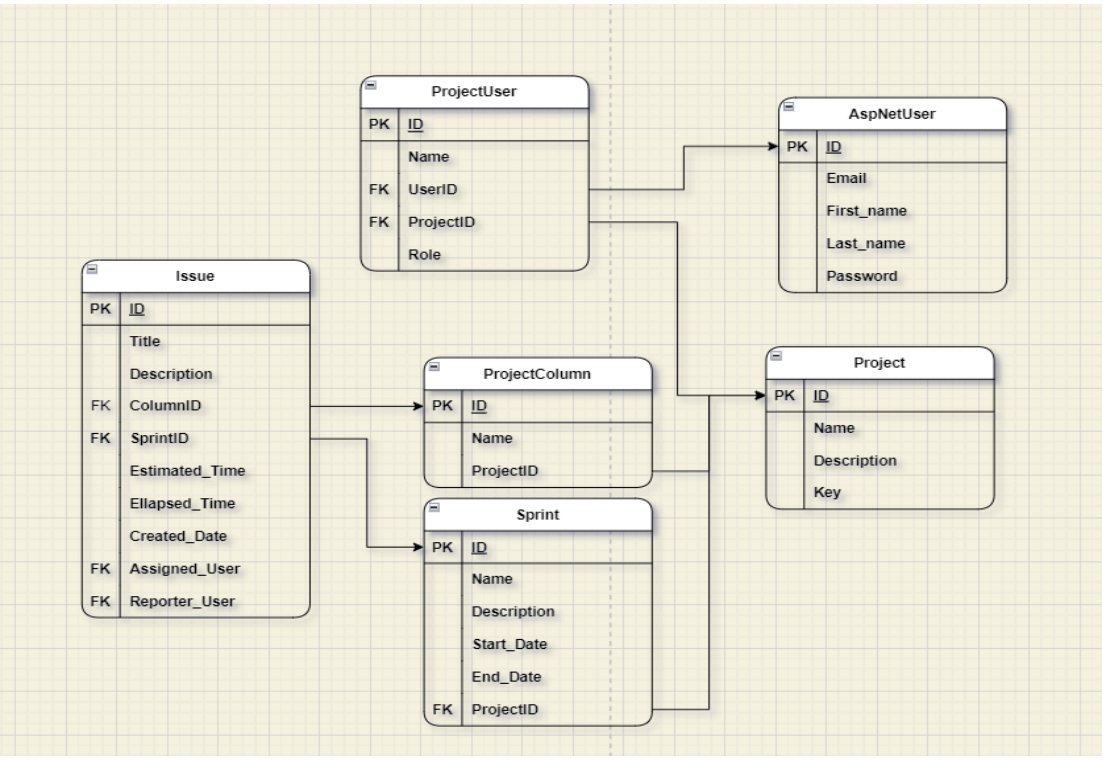


Рисунок 3.2.1 – Логічна модель бази даних

Тепер перейдемо до створення фізичною моделі бази даних [10]. Зазвичай вона походить від логічної бази даних та створюється на її основі. Та на відміну від логічної моделі, фізична модель проєктується з врахуванням конкретних обмежень СУБД. Для побудови фізичної моделі буде використано онлайн застосунок dbdiagram.io, який є також безкоштовним і окрім схеми він формує програмний код для таких СУБД, як PostgreSQL, MySQL і SQL Server.

Також на фізичній моделі нам потрібно вказати типи зв’язків між таблицями бази даних. Їх буває три типи:

- один до одного. Це коли кожній сутності А відповідає лише одна сутності Б та навпаки. В даній моделі не буде використано.
- один до багатьох. Цей тип стосується відношень коли у відповідність до сутності А відповідає декілька елементів Б, але в той час елементу Б відповідає лише 1 елемент А. В даній моделі таке відношення буде між таблицями користувач системи - учасник проєкту, проєкт - учасник проєкту, проєкт - стовпець, проєкт - спринт, спринт - завдання та стовпець - завдання.Усім сутностям, які на першому місці у відповідь може бути кілька сутностей, які на другому місці.
- багато до багатьох. Це коли сутності А може стояти у відповідь декілька сутностей Б, та навпаки. В нашій системі даний зв’язок буде між сутністю проєкт – та користувач системи. Оскільки як користувач може мати багато проєктів, так і проєкт може мати багато користувачів. Важливо, що даний зв’язок буде реалізовано за допомогою додаткової таблички (учасник проєкту) та її двох один-до-багатьох зв’язків.

Так виглядає розроблена нами фізична модель системи(рисунок 3.2.2):

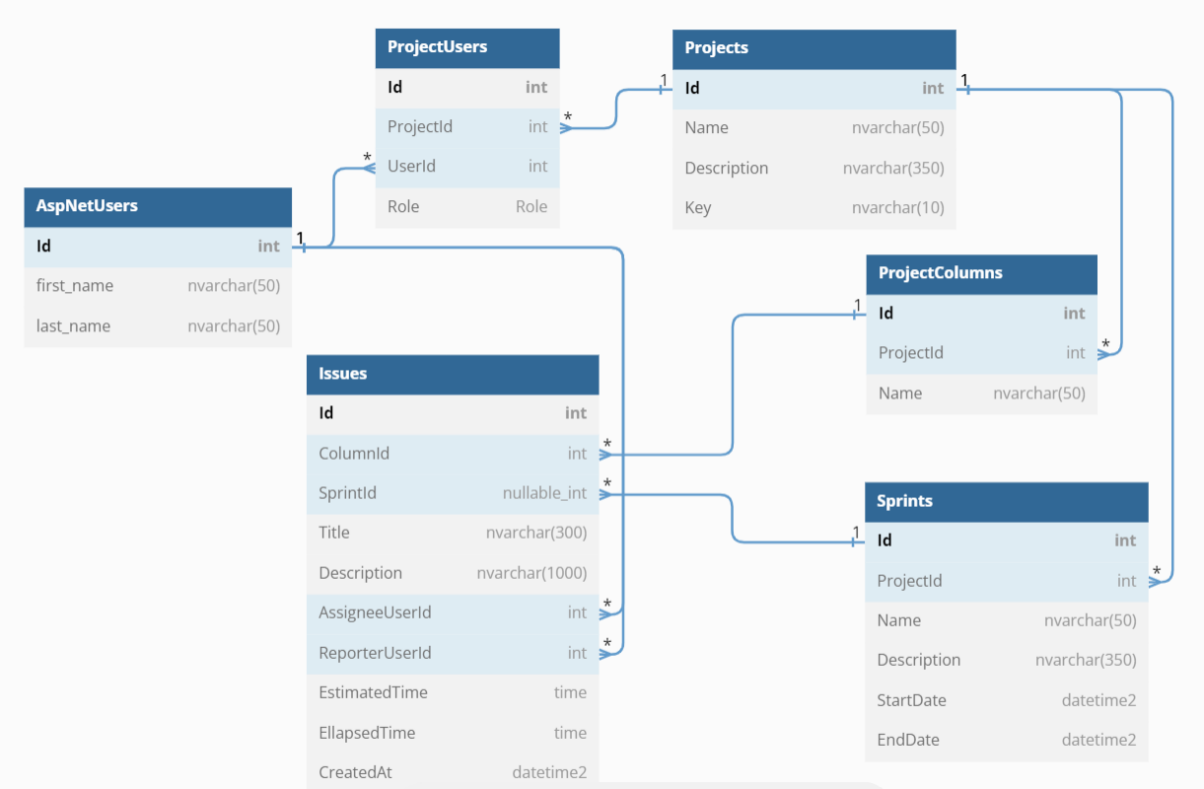


Рисунок 3.2.2 – Фізична модель бази даних.

В даній схемі ми вказали всі потрібні нам таблиці, поля та їх типи даних, ключі таблиць, види відношень між ними. Дана модель надає оптимальне зберігання даних та усієї для цього необхідної додаткової інформації, забезпечує зберігання даних без їх дублювання, що в свою чергу забезпечує кращу продуктивність та швидкодію. Спроєктована схема БД також дозволяє реалізувати усі зазначені нами функції.

3.3 Створення інтерфейсу користувача

У даному підрозділі будуть розроблені усі необхідні веб сторінки для нашого проєкту. Почнемо з розробки головної сторінки нашого проєкту. На ній буде знаходити навігаційне меню (яке буде використано на всіх інших сторінках), в ньому будуть посилання для роботи з нашими моделями (будуть сторінки для реалізації CRUD операцій через користувацький інтерфейс, а не напряму через базу даних) та посилання на сторінки реєстрації і авторизації користувача, вітання користувачу та контейнер для вибору уже існуючих

проєктів або створення нового проєкту. Сторінка буде виглядати приблизно як на рисунку 3.3.1:

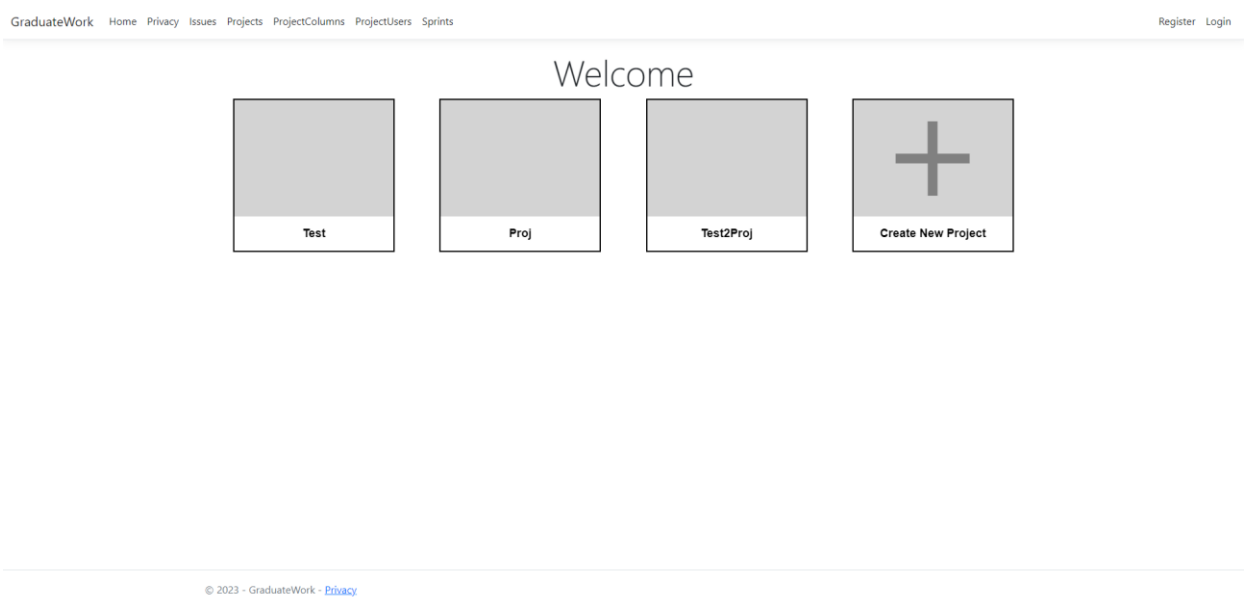


Рисунок 3.3.1 – Головна сторінка

Тепер розробимо сторінку створення проєкту (рисунок 3.3.2). В даному проєкті вона буде реалізована як часткове відображення. Часткове відображення – це файл розмітки Razor.cshtml без @ page директиви, яка відображає вихідні дані HTML у даних, що відображається іншим файлом розмітки, тобто дані відображення можна впроваджувати у інші звичайні відображення. Воно буде з’являтися поверх основної сторінки та матиме такий дизайн:

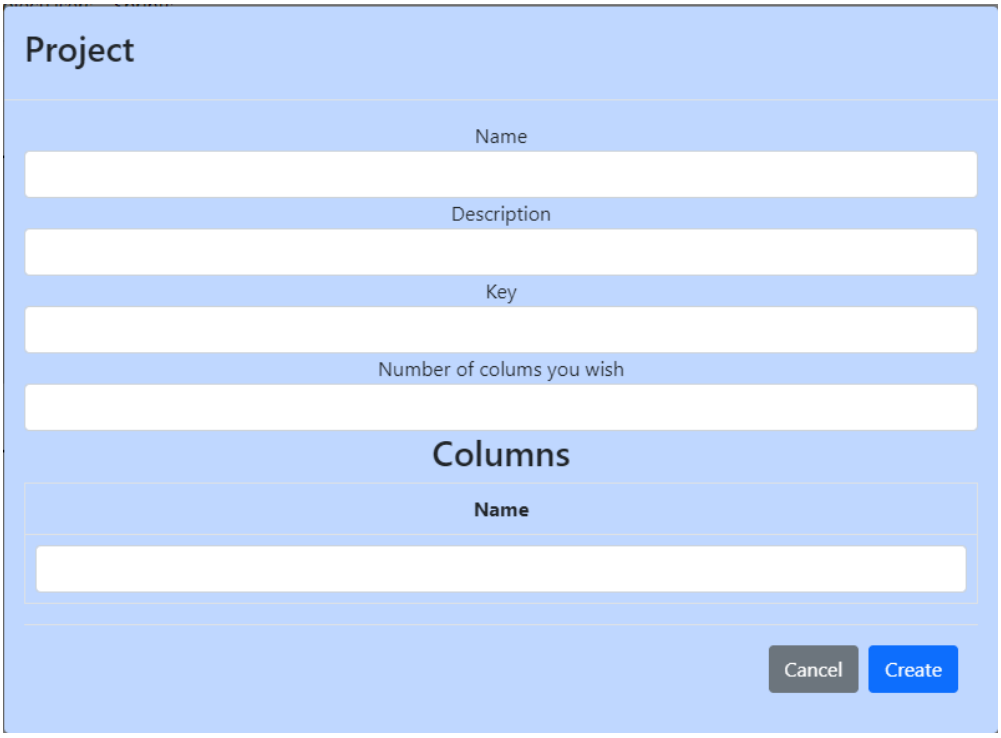


Рисунок 3.3.2 – Сторінка створення проєкту

Дана форма є адаптивною та при зміні кількості бажаних стовпців генерує необхідну кількість полів для їх створення.

Сторінки авторизації та реєстрації зображені на рисунку 3.3.3 (дані сторінки є автогенерованими, але не потребують ніяких змін):

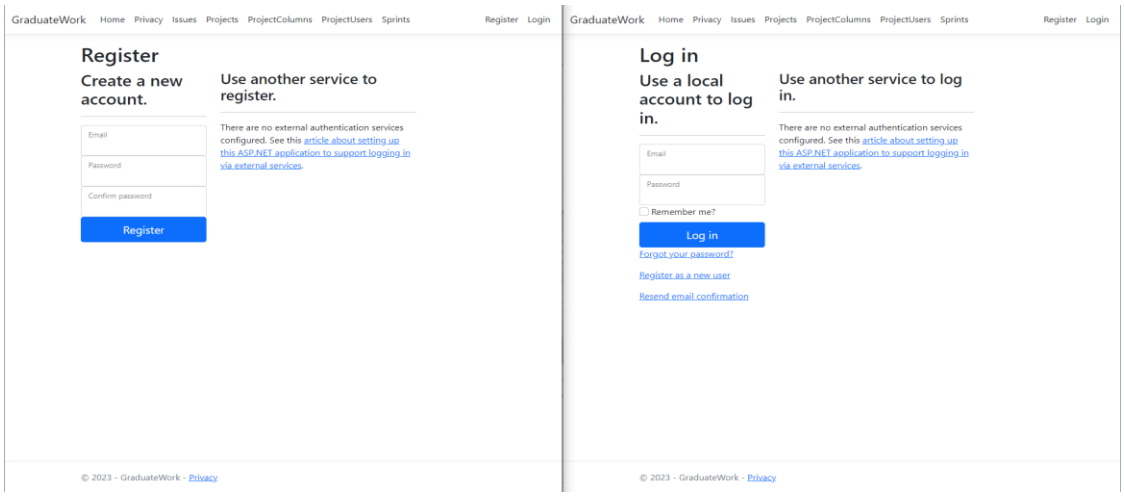


Рисунок 3.3.3 – Сторінки авторизації та реєстрації

Тепер перейдемо до розробки сторінки для дошки завдань (рисунок 3.3.4). Окрім головного меню дана сторінка буде мати іще додаткове меню. Через

нього нам будуть доступні інші функції: почати спринт, закінчити спринт, фільтр завдань за спринтом, створення завдання, управління користувачами та статистика за спринтом та статистика за цілим проєктом. Після меню буде знаходитися дошка з завданнями розподіленими за їх статусами (колонками).

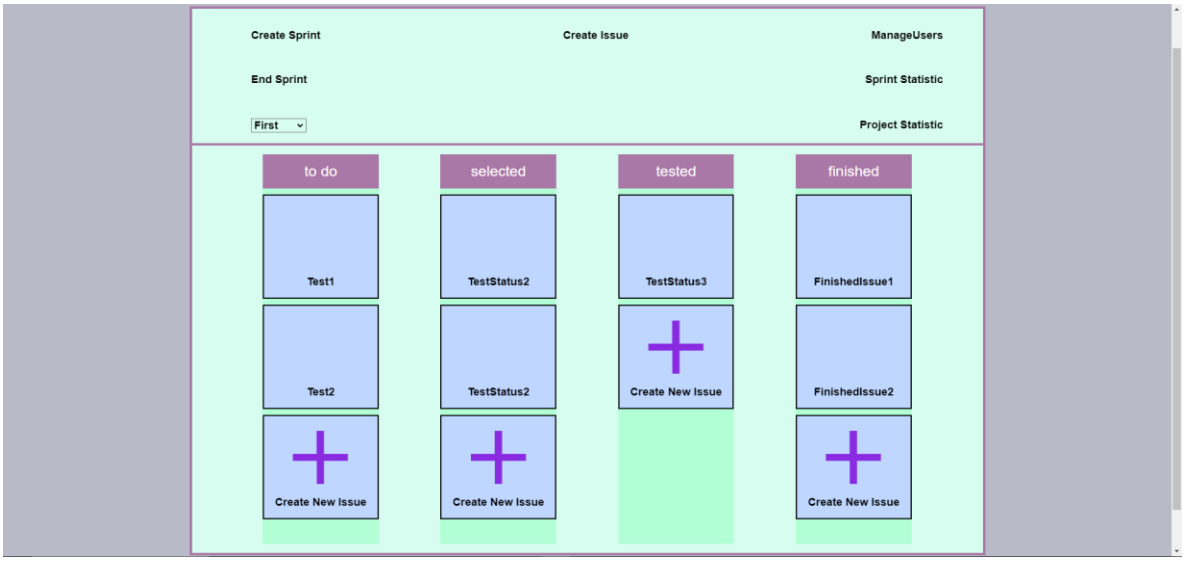


Рисунок 3.3.4 – Сторінка робочої дошки

Для створення та редагування спринтів, завдань та інших сутностей будуть використані часткові відображення аналогічні до форми створення нового проєкту. Сторінки для видалення (рисунок 3.3.5) та перегляду деталей (рисунок 3.3.6) екземплярів моделей буде наведено на наступній сторінці.

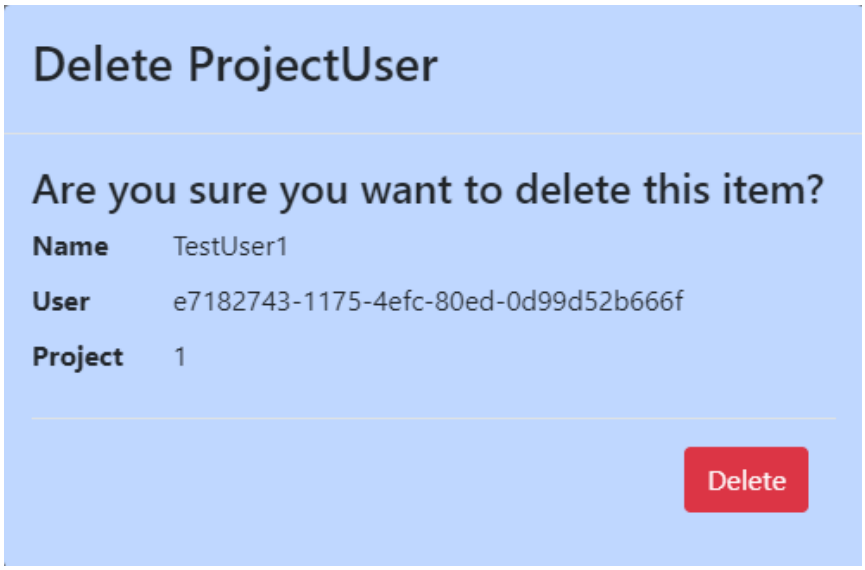


Рисунок 3.3.5 – Сторінка видалення

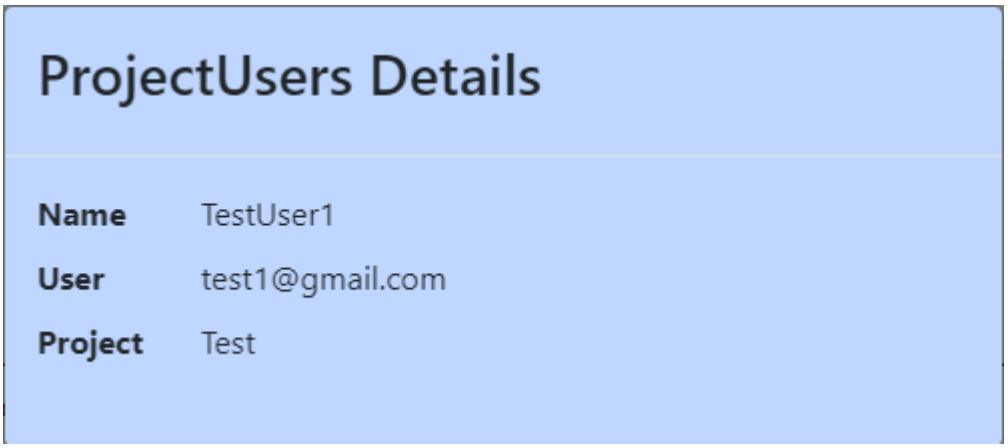


Рисунок 3.3.6 – Сторінка перегляду деталей

Тепер створимо сторінку закінчення спринта (рисунок 3.3.7). На даній сторінці буде таблиця спринтів, які є активними на даний момент, та кнопка для їх завершення.

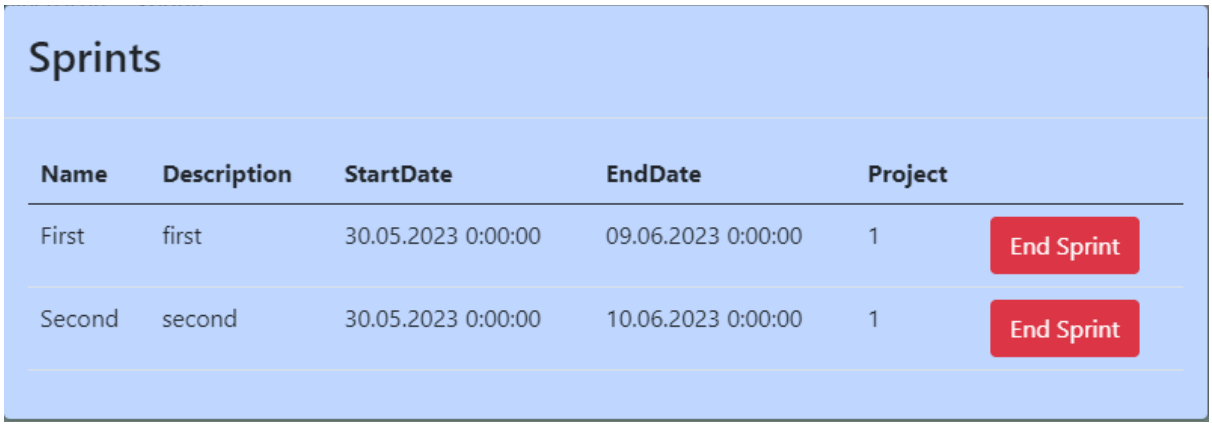


Рисунок 3.3.7 – Сторінка завершення активних спринтів

На сторінці менеджменту учасниками проєкту (рисунок 3.3.8) буде розміщена таблиця, та кнопки для реалізації CRUD операцій. Аналогічно даній сторінці будуть створені і інші сторінки для менеджменту інших сутностей (проєкт та інші).

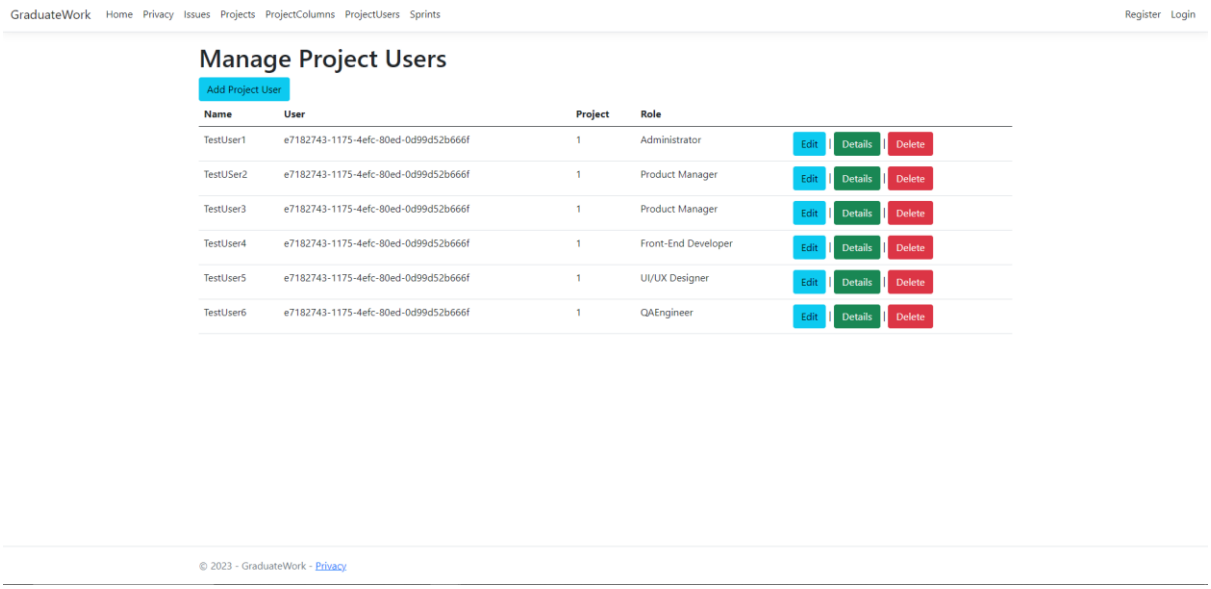


Рисунок 3.3.8 – Сторінка управління учасниками проєкту

Для дизайну сторінок “Sprint Statistics” та “Project Statistics” буде використаний один шаблон (рисунок 3.3.9). На сторінці буде головне меню та зображена за допомогою Chart.js стовпчикова діаграма:

Рисунок 3.3.9 – Дизайн сторінки для отримання статистики

В даному підрозділі було розроблено всі необхідні сторінки, після цього можна перейти до розробки серверної частини.

3.4 Розробка серверної частини

Для початку необхідно створити проєкт (для розробки буде використовуватися Visual Studio 2022), в якості шаблона проєкту необхідно обрати ASP.NET Core Web App (Model-View-Controller) C#. Далі обираємо ім’я та місцезнаходження проєкту. В графі додаткової інформації обираємо версію фреймворку 6.0, тип аутентифікації – Individual Accounts. Завдяки даному налаштуванню буде згенерована аутентифікація за замовчуванням.

Після цього необхідно встановити та налаштувати Entity Framework Core. Для цього необхідно скористатися пакетним менеджером NuGet. У вікні Solution Explorer потрібно натиснути правою кнопкою миші у структурі проєкту на вузол Dependencies і в меню обрати Manage NuGet Packages. Необхідно встановити такі пакети(рисунок 3.4.1):

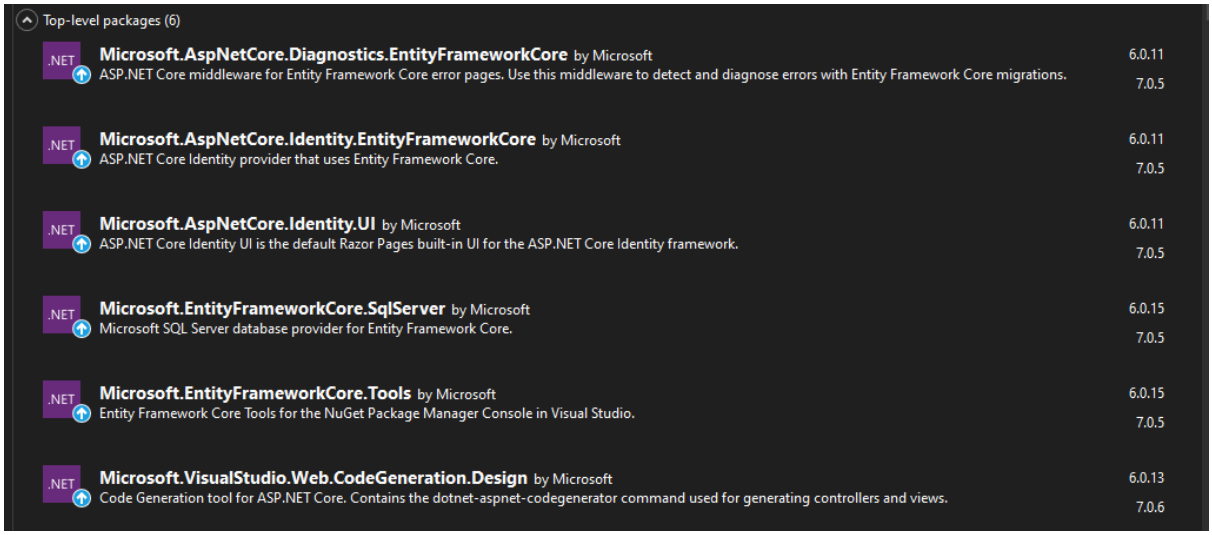


Рисунок 3.4.1 – необхідні пакети для використання Entity Framework

Тепер можна створити моделі. Для цього необхідно створити клас в папці Models, та описати в ньому необхідні атрибути. Для прикладу розглянемо модель проєкту, яка зображена на рисунку 3.4.2:

```
namespace GraduateWork.Models
{
    42 references
    public class Project
    {
        24 references
        public int Id { get; set; }
        24 references
        public string Name { get; set; }
        15 references
        public string Description { get; set; }
        15 references
        public string Key { get; set; }
        1 reference
        public ICollection<ProjectUser>? ProjectUsers { get; set; }
        6 references
        public List<ProjectColumn> ProjectColumns { get; set; } = new List<ProjectColumn>();
        1 reference
        public List<Sprint>? Sprints { get; set; }
    }
}
```

Рисунок 3.4.2 – Модель Project

Аналогічно даної моделі будуть написані моделі ProjectColumn, Sprint, ProjectUser та Issue. Але необхідно розглянути модель ApplicationUser, вона відрізняється від усіх інших. При створенні проєкту з заданим нами параметром аутентифікації в проєкті уже автоматично створюються деякі таблиці, вони наведені на рисунку 3.4.3.

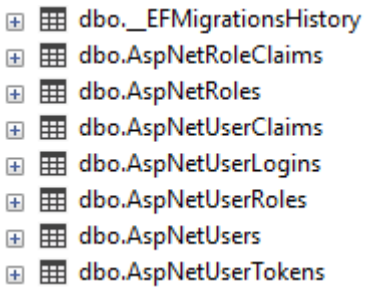


Рисунок 3.4.3 – Автоматично створені таблиці

В таблиці AspNetUsers зберігаються всі зареєстровані користувачі через нашу автозгенеровану систему аутентифікації. Проте для звернення для даної таблиці необхідно провести додаткові налаштування. Створимо модель ApplicationUser (рисунок 3.4.4). На відміну від інших моделей вона повинна наслідуватися від класу IdentityUser, для того щоб отримати доступ до полів та методів батьківського класу. Даний клас також необхідно наслідувати від класу IdentityDbContext після чого в трикутних дужках необхідно передати створений

нами клас ApplicationUser, для того щоб підключити також таблиці та зв'язки, необхідні для ASP.NET Core Identity.

```
using Microsoft.AspNetCore.Identity;

namespace GraduateWork.Models
{

    public class ApplicationUser : IdentityUser
    {

        public ICollection<ProjectUser>? ProjectUsers { get; set; }
        1 reference
        public virtual ICollection<Issue>? AssignedIssues { get; set; }
        1 reference
        public virtual ICollection<Issue>? ReportedIssues { get; set; }
    }
}
```

Рисунок 3.4.4 – Модель ApplicationUser

Після створення моделей необхідно налаштувати ApplicationDbContext. ApplicationDbContext — це клас, який представляє контекст бази даних для нашого проєкту. Це похідний клас від класу DbContext, наданого Entity Framework Core, який є структурою об'єктно-реляційного відображення (ORM). Даний клас відповідає за взаємодію з основною базою даних, для взаємодії з даними з таблиць ми можемо використовувати DbSet<Entity>, де Entity - назва моделі/таблиці створеної розробником. Також в даному класі ми пропишемо усі необхідні залежності. Код файлу наведений у додатку А.

Також у файлі Program.cs необхідно оновити параметри аутентифікації (наведено на рисунку 3.4.5), замість класу IdentityUser нам потрібно передавати ApplicationUser.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddControllersWithViews();

var app = builder.Build();
```

Рисунок 3.4.5 – Файл Program.cs

Також необхідно змінити також файл LoginPartial.cshtml у папці Shared.

Було :

```
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
```

Стало:

```
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager
```

Система аутентифікації повністю налаштована.

Тепер необхідно додати CRUD операції для всіх сутностей. Зробимо це на прикладі сутності ProjectUser. Усі операції будуть реалізовані з використанням часткових відображень, які будуть відкриватися поверх головної сторінки за допомогою JavaScript коду. В папці Controllers створимо новий клас ProjectUsersController, який в свою чергу наслідується від класу Controller. Спочатку створимо у ньому екземпляр класу ApplicationDbContext та призначимо його приватній змінній context. Після цього перейдемо до створення методів (код наведено у додатку Б) Index буде передавати всі записи з таблиці ProjectUser; Details(int? id) – шукає в БД необхідний запис та надає детальну інформацію про нього; CreateOrEdit(int? id = null) – метод, для створення або редагування об’єктів БД. Метод має параметр id типу null або int з null значенням за замовчуванням. Тобто якщо даний метод буде викликано без параметра, то змінна id набуде значення null. Після цього за допомогою умовних конструкцій ми буде визначено чи необхідно створити новий запис чи оновити існуючий. В Get запиті ми будемо робити запит на отримання сторінки створення/редагування, а в Post будемо передавати дані на сервер для обробки, після чого буде додано/оновлено дані в таблиці; get запит Delete(int? id) та пост метод DeleteConfirmed(int id) для видалення записів. Відповідно до методів також необхідно створити відповідні view з однойменними назвами. Для генерації часткових відображень поверх головної сторінки буде використано такі скрипти, як на рисунку 3.4.6 .

Рисунок 3.4.6 – Скрипти для відкриття часткових відображень для дій create, update та delete для моделі ProjectUser

В якості url ми описуємо повну сігнатуру методу контролеру, якщо даний метод знайдено, то всередині елементу div з id [action]Table ми будемо генерувати відображення з id [action]ProjectUser. Виклик функції в коді виглядає так:

```
<button type="button" data-toggle="modal" data-target="#addProjectUser"
onclick="addProjectUser()">
    Add Project User
</button>
```

Також в частковому відображенні необхідно створити контейнер-обгортку:

```
<div class="modal fade" role="dialog" tabindex="-1" id="addProjectUser"
aria-labelledby="addProjectUserLable" aria-hidden="true"></div>
```

Тепер при натисканні на кнопку буде відкриватися необхідне нам спливаюче вікно. Також варто відзначити, що при розробці буде використано

Razor код, його можна ідентифікувати за знаком “@”. Він використовується для отримання доступу до даних, переданих у контролері, підключення Layout-сторінки, передачі відображуваного заголовка сторінки, та для генерації необхідної кількості блоків та елементів всередині них, та виведення даних про об’єкти таблиці, їх атрибути і значення. Для прикладу наведемо код для виведення всіх даних з таблиці ProjectUser (рисунок 3.4.8). За допомогою циклу foreach ми генеруємо таку кількість рядків скільки є записів у таблиці, та отримуємо доступ до значень усіх атрибутів ProjectUser та виводимо їх.

```
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.User.Id)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Project.Id)
            </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Role)
                </td>
            <td>
                <button type="button" class="btn btn-info" data-toggle="modal" data-target="#addProjectUser" onclick="addProjectUser(@item.Id)">
                    Edit
                </button> |
                <button type="button" class="btn btn-success" data-toggle="modal" data-target="#projectUserDetails" onclick="projectUserDetails(@item.Id)">
                    Details
                </button> |
                <button type="button" class="btn btn-danger" data-toggle="modal" data-target="#deleteProjectUser" onclick="deleteProjectUser(@item.Id)">
                    Delete
                </button>
            </td>
        </tr>
}
```

Рисунок 3.4.8 – Код для виведення всіх даних з таблиці ProjectUser

Аналогічні дії проведемо для інших сутностей. Зупинимося на моделі проєкту, оскільки тут при створенні проєкту необхідно створювати одразу і всі необхідні колонки. Для генерації адаптивної форми буде використано такий скрипт (рисунок 3.4.9). Варто відзначити, що скрипт знаходиться в view CreateProject, скрипт для виклику форми аналогічний до раніше розглянутих.

```
<script>
    $("#numOfColumns").on('change', function(){
        var RowNumber = document.getElementById("numOfColumns").value;
        $("#columnsTable tbody").find("tr:not(:first)").remove();

        for(var i = 0; i < RowNumber - 1; i++) {
            var table = document.getElementById('columnsTable');
            var rows = table.getElementsByTagName('tr');
            var rowOuterHtml = rows[rows.length - 1].outerHTML;
            var lastrowIdx = document.getElementById('LastIndex').value;

            var nextrowIdx = eval(lastrowIdx) + 1;

            document.getElementById('LastIndex').value = nextrowIdx;
            rowOuterHtml = rowOuterHtml.replaceAll('_' + lastrowIdx + '_', '_' + nextrowIdx + '_');
            rowOuterHtml = rowOuterHtml.replaceAll('[' + lastrowIdx + ']', '[' + nextrowIdx + ']');
            rowOuterHtml = rowOuterHtml.replaceAll('-' + lastrowIdx, '-' + nextrowIdx);

            var newRow = table.insertRow();
            newRow.innerHTML = rowOuterHtml;
        }
    })
</script>
```

Рисунок 3.4.9 – Скрипт для генерації заданої кількості колонок

Також необхідно написати додаткову HTML розмітку для створення адаптивної таблиці всередині форми (рисунок 3.4.10).

```
<table class="table table-bordered" id="columnsTable">
    <thead>
        <tr>
            <th>
                Name
            </th>
        </tr>
    </thead>

    <tbody>
        @for (int i = 0; i < Model.ProjectColumns.Count; i++)
        {
            <tr class="form-item">
                <td>
                    @Html.EditorFor(p => p.ProjectColumns[i].Name, new { htmlAttributes = new {@class = "form-control"} })
                </td>
            </tr>
        }
    </tbody>
</table>
<input type="hidden" id="LastIndex" value="0" />
```

Рисунок 3.4.10 – Додаткова HTML розмітку для створення адаптивної таблиці

Тепер можливим стає додавання посилань для роботи з даними моделей в головне навігаційне меню. Спочатку створимо відповідні методи в контролері Home, який в свою чергу буде пересилати користувача на сторінку Index відповідної сутності, де будуть доступними написані CRUD операції. Розглянемо код на прикладі моделі Project (рисунок 3.4.11)

```
0 references
public IActionResult Projects()
{
    return RedirectToAction("Index", "Projects");
}
0 references
```

Рисунок 3.4.11 – код методу контролеру Home для пересилання на метод Index() контролеру Projects

Після цього необхідно відкрити файл _Layout.cshtml та створити в головному меню посилання, та передати на них відповідні дії контролера (рисунок 3.4.12).

```
<ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Issues">Issues</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Projects">Projects</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="ProjectColumns">ProjectColumns</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="ProjectUsers">ProjectUsers</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Sprints">Sprints</a>
    </li>
</ul>
```

Рисунок 3.4.12 – Оновлене навігаційне меню в файлі _Layout.cshtml

Тепер перейдемо до написання основної бізнес логіки проєкту. На початковій сторінці ми будемо створювати каталог проєктів та кнопку створення нового проєкту, для цього в методі Index (рисунок 3.4.13), ми будемо передавати вибірку проєктів з таблички Project.

```
0 references
public async Task<IActionResult> Index()
{
    return View( await _dbContext.Projects.ToListAsync());
}
```

Рисунок 3.4.13 – метод Index() контроллера Home

При натисканні на кнопку створення нового проєкту будемо викликати метод створення, аналогічний до того, який знаходить на view Index.cshtml в папці Projects. За умови, натиску на уже існуючий проєкт ми будемо викликати метод Board(int? Id, int? sprintId = null), який зображений на рисунку 3.4.14. Даний метод має два параметри, перший – id проєкту, для того, щоб отримати дані про всі колонки, для генерації таблиці завдань, другий – id спринту для реалізації фільтру завдань за спринтом. Другий параметр є опціональний, та за замовчуванням дорівнює нулю. Якщо параметр при виклику методу буде дорівнювати null – то в якості фільтру буде обрано перший активний спринт на даний момент (активним вважається той спринт, який почався до поточного моменту та закінчиться після нього ж). Реалізація функціоналу, на випадок відсутності будь-якого активного спринту не розглядатиметься в рамках даної роботи.

```
[HttpGet]
0 references
public async Task<IActionResult> Board(int? Id, int? sprintId = null)//Project id
{
    ViewData["SelectList"] = new SelectList(_dbContext.Sprints, dataValueField:"Id", dataTextField:"Name");
    IEnumerable<Sprint> sprints = await _dbContext.Sprints.Where(s => s.ProjectId == Id).ToListAsync();
    ViewBag.Sprint = sprints;


    DateTime current = DateTime.Now;

    if (sprintId == null)
    {
        sprintId = await _dbContext.Sprints
            .Where(s => s.ProjectId == Id && s.EndDate > current && s.StartDate < current)
            .Select(s => s.Id)
            .FirstOrDefaultAsync();
    }

    var columns = _dbContext.ProjectColumns
        .Include(c => c.Issues.Where(i => i.SprintId == sprintId))
        .Where(c => c.ProjectId == Id);
    return View(await columns.ToListAsync());
}
```

Рисунок 3.4.14 – метод Board(int? Id, int? sprintId = null)

Тепер напишемо метод End Sprint (рисунок 3.4.15), який буде повертати однойменне часткове відображення. Даний метод має один параметр – id проєкту. За даним id буде виконано пошук в базі даних всіх екземплярів спринтів у яких id проєкту дорівнює переданому id та даний спринт є активним

на поточний момент. Варто відзначити, що даний момент не завершує спринт, а лише повертає відповідне view.

```
[HttpGet]
0 references
public async Task<IActionResult> EndSprint(int Id)
{
    DateTime current = DateTime.Now;
    ViewBag.ID = Id;
    var sprint_context = _dbContext.Sprints.Where(s => s.EndDate > current && s.ProjectId == Id).Include(c => c.Project).ToListAsync();
    return PartialView(await sprint_context);
}
```

Рисунок 3.4.15 – метод EndSprint(int Id)

Тепер розглянемо метод FinishSprint(int Id), який в свою чергу буде викликатися на view EndSprint.cshtml, даний метод буде безпосередньо завершувати обраний користувачем спринт. У методу є один параметр – id спринту, всередині методу ми знаходимо екземпляр спринту, у якого id дорівнює переданому id, після чого встановлюємо йому дату завершення, яка є на момент виклику, та оновлюємо дані в базі даних, після чого переходимо на сторінку дошки, передаючи в параметри методу RedirectToAction, назву методу, контролеру та id проєкту.

```
[HttpGet]
0 references
public async Task<IActionResult> FinishSprint(int Id)//Sprint id
{
    Sprint sprint = await _dbContext.Sprints.Where(s => s.Id == Id).FirstOrDefaultAsync();
    sprint.EndDate = DateTime.Now;
    _dbContext.Update(sprint);
    await _dbContext.SaveChangesAsync();
    var project_id = await _dbContext.Projects.Where(s => s.Id == sprint.ProjectId).FirstOrDefaultAsync();
    return RedirectToAction("Board", "Home", project_id);
}
```

Рисунок 3.4.16 – метод FinishSprint(int Id)

Наступним буде метод ManageUsers(int? Id), який зображений на рисунку 3.4.17. Даний метод має один параметр – id проєкту. В методі ми знаходимо проєкт у якого id дорівнює отриманому параметрі, та пересилаємося на action Index контролеру ProjectUsers.

```
[HttpGet]
0 references
public async Task<IActionResult> ManageUsers(int? Id)//Project id
{
    var project_id = await _dbContext.Projects.Where(p => p.Id == Id).FirstOrDefaultAsync();
    return RedirectToAction("Index","ProjectUsers", project_id);
}
```

Рисунок 3.4.17 – метод ManageUsers(int? Id)

Тепер розглянемо метод SprintStatistic(int? Id) – він зображений на рисунку 3.4.18 . Даний метод має один параметр – id спринту, за заданим параметром виконується пошук екземплярів завдань після чого їх атрибути – назва, планово витрачений та фактично витрачений об’єми часу будуть записані в відповідні списки. Також за допомогою циклу розраховуватиметься сумарний час за тими ж атрибутами, який згодом буде доданий в кінець списку. Після чого дані будуть передані на відповідну сторінку.

```
[HttpGet]
0 references
public async Task<IActionResult> SprintStatistic(int? Id)
{
    var issues_context = await _dbContext.Issues.Where(Issues => Issues.SprintId == Id).ToListAsync();
    List<decimal> estimatedTimes = new List<decimal>();
    List<decimal> elapsedTimes = new List<decimal>();

    List<String> issueNames = new List<String>();
    decimal totalEstimatedTime = 0;
    decimal totalElapsedTime = 0;

    foreach (var issue in issues_context)
    {
        decimal estimateDecimalValue = Convert.ToDecimal(issue.EstimatedTime.TotalHours);
        decimal elapsedDecimalValue = Convert.ToDecimal(issue.EllapsedTime?.TotalHours ?? 0);
        totalEstimatedTime += estimateDecimalValue;
        totalElapsedTime += elapsedDecimalValue;
        issueNames.Add(issue.Title);
        estimatedTimes.Add(estimateDecimalValue);
        elapsedTimes.Add(elapsedDecimalValue);
    }

    estimatedTimes.Add(totalEstimatedTime);
    elapsedTimes.Add(totalElapsedTime);
    issueNames.Add("TotalTime");

    ViewBag.EstimatedTimes = estimatedTimes;
    ViewBag.ElapsedTimes = elapsedTimes;
    ViewBag.IssueNames = issueNames;
    DateTime current = DateTime.Now;
    var sprint = await _dbContext.Sprints.Where(s => s.Id == Id).FirstOrDefaultAsync();
    if (sprint == null)
    {
        ViewBag.Message = "You didn't select sprint!!!";
    }
    else {
    ViewBag.Message = "This is a statistic from " + sprint.Name + " sprint! In the last column you can see total count" ;
    }
    return View();
}
```

Рисунок 3.4.18 – метод SprintStatistic(int? Id)

Після цього на відповідному view необхідно написати скрипт (рисунок 3.4.19), який буде генерувати стовпчикову діаграму. На даний момент сумарні оцінюваний та витрачений час будуть наведені у останньому стовпчику.

```
<script>
    $(document).ready(function () {
        var estimatedTimes = @Html.Raw(Json.Serialize(ViewBag.EstimatedTimes));
        var elapsedTimes = @Html.Raw(Json.Serialize(ViewBag.ElapsedTimes));
        var issueNames = @Html.Raw(Json.Serialize(ViewBag.IssueNames));
        var ctx = document.getElementById("issueChart").getContext("2d");

        var chart = new Chart(ctx, {
            type: "bar",
            data: {
                labels: Array.from({ length: estimatedTimes.length }, (_, i) => issueNames[i]),
                datasets: [{
                    label: "Estimated Time",
                    data: estimatedTimes,
                    backgroundColor: "rgba(75, 192, 192, 0.2)",
                    borderColor: "rgba(75, 192, 192, 1)",
                    borderWidth: 1
                }, {
                    label: "Elapsed Time",
                    data: elapsedTimes,
                    backgroundColor: "rgba(255, 99, 132, 0.2)",
                    borderColor: "rgba(255, 99, 132, 1)",
                    borderWidth: 1
                }]
            },
            options: {
                scales: {
                    y: {
                        beginAtZero: true
                    }
                }
            }
        });
    });
</script>
```

Рисунок 3.4.19 – Скрипт для генерації стовпчикової діаграми

Для фільтрації завдань за фільтрами буде використано випадаючий список та кнопка, яка буде зчитувати обраний користувачем фільтр та передаватиме його в метод Board, після чого генеруватиме нову таблицю завдань. Для генерації списку буде використано код, наведений на рисунку 3.4.20.

```
<select id="sprintList" class="innerText">
    <option value="null">Choose sprint</option>
    @foreach (Sprint sprint in ViewBag.Sprint)
    {
        <option value="@sprint.Id">@sprint.Name</option>
    }

</select>
<button id="boardButton" class="innerText" asp-action="Board" asp-route-id="@project_id" asp-route-sprintId="null">Go to Board</button>
```

Рисунок 3.4.20 – Випадаючий список спринтів

Для зчитування даних з списку було розроблено такий скрипт (рисунок 3.4.21):

```
document.getElementById('boardButton').addEventListener('click', function () {
    var sprintId = document.getElementById('sprintList').value;
    var url = '/Home/Board?Id=@project_id&sprintId=' + sprintId;
    window.location.href = url;
});
```

Рисунок 3.4.21 – Скрипт для фільтрації завдань

Примітка. Код аналогічний даному також використаний для зчитування обраного спринту для методу SprintStatistic().

Отже, в даному підрозділі було розроблено код для сервісної частини інформаційної системи.

3.5 Аналіз функціональності інформаційної системи.

Для аналізу функціональності інформаційної системи спробуємо змоделювати процес розробки для даної роботи. Для початку спробуємо створити новий проєкт, назвемо його Graduate Project і одночасно створимо такі статуси завдань як To Do, In Progress, To Verify та Finished (рисунок 3.5.1).

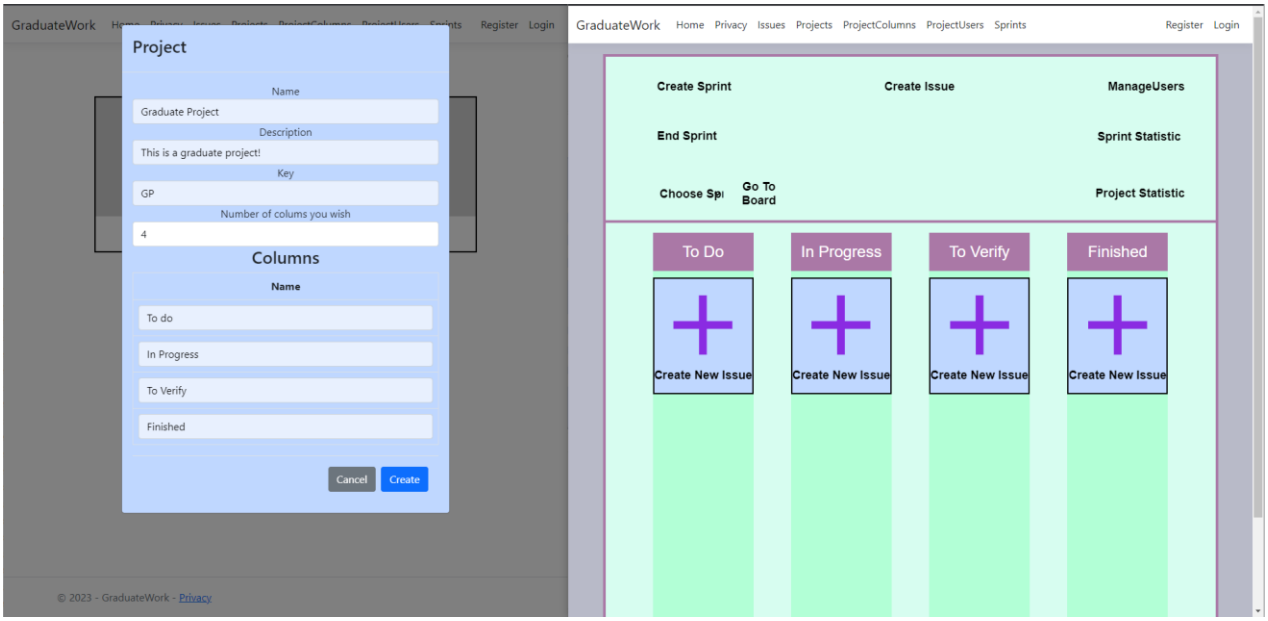


Рисунок 3.5.1 – Створення проєкту

Далі створюємо декілька спринтів, які будуть активними, наприклад “Front-end” та “Back-end”. Після цього створюємо відповідні завдання для кожного спринту, та перевіряємо фільтрування. Фільтрування залежно від спринтів наведено на рисунках 3.5.2 та 3.5.3 відповідно.

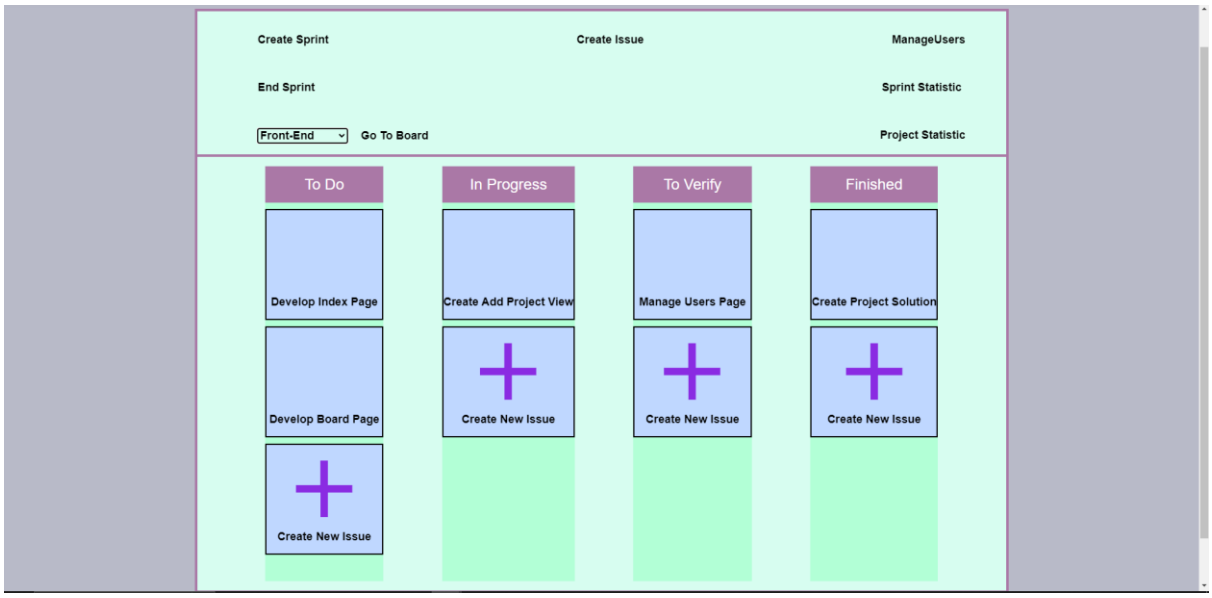


Рисунок 3.5.2 – Фільтрування завдань за спринтом “Front-end”

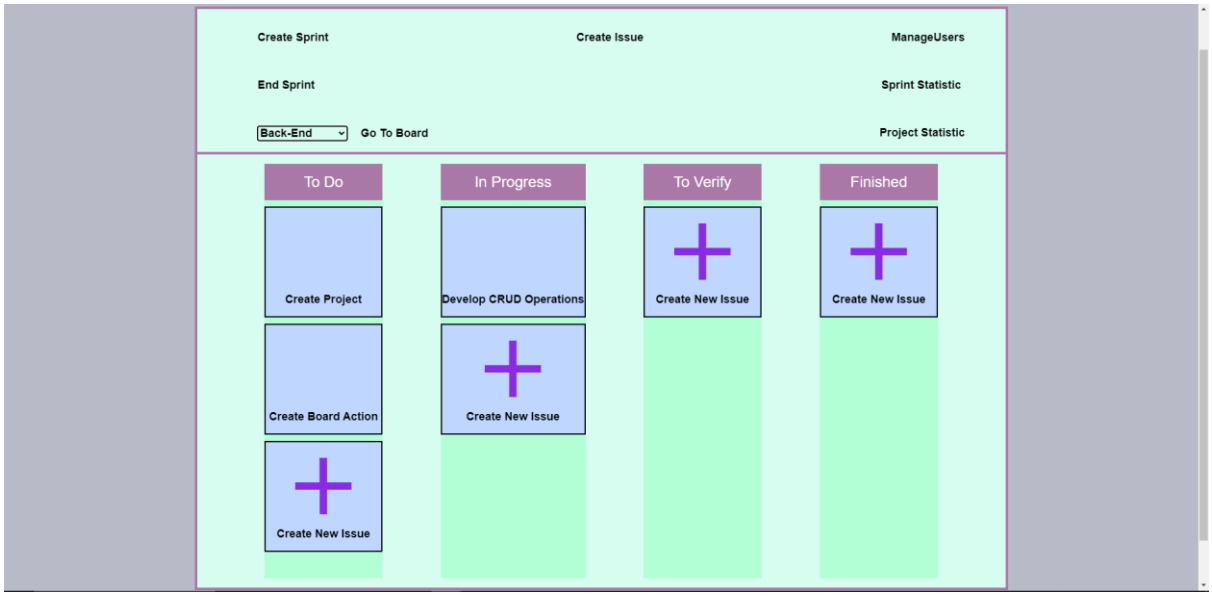


Рисунок 3.5.3 – Фільтрування завдань за спринтом “Back-end”

Тепер переведемо завдання в статус виконаних та встановимо їм

витрачений час (EllapsedTime). Після чого переглянемо статистику за спринтами(рисунок 3.5.4).

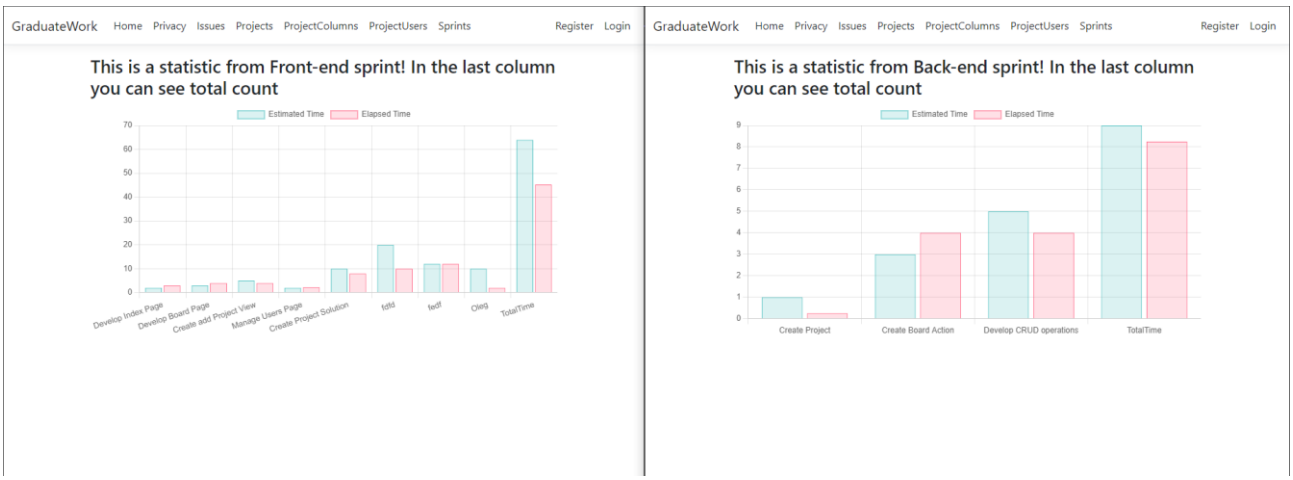


Рисунок 3.5.4 – Статистика кожного зі спринтів

Тепер спробуємо завершити спринт Front-end та переглянемо як зміниться його дата завершення на сторінці Index, контролеру Sprints (стоінка на якій виводяться усі спринти з таблички Sprint). Дані до зміни та після зміни наведені на рисунках 3.5.5 та 3.5.6 відповідно.

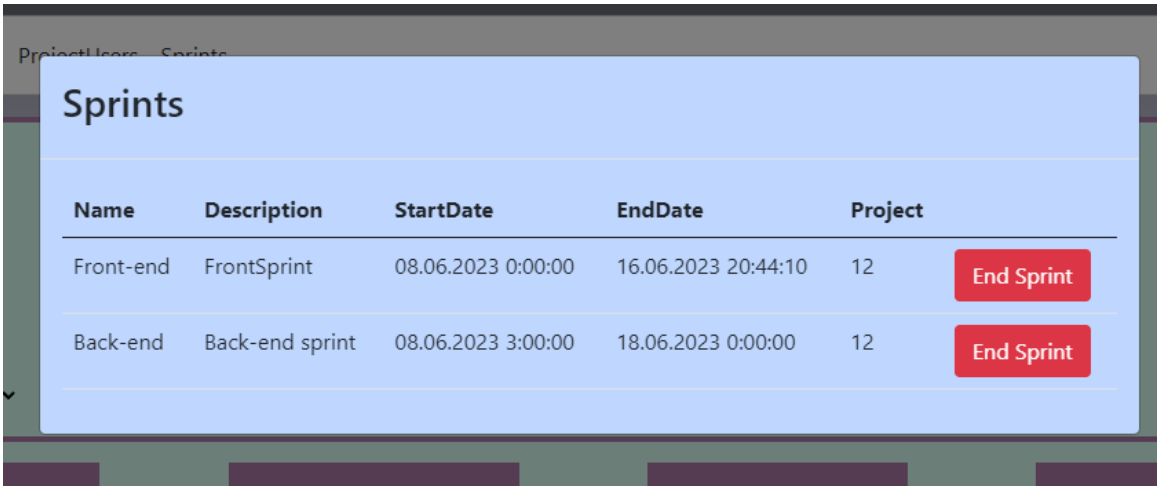


Рисунок 3.5.5 – Дані до завершення спринту “Front-end”

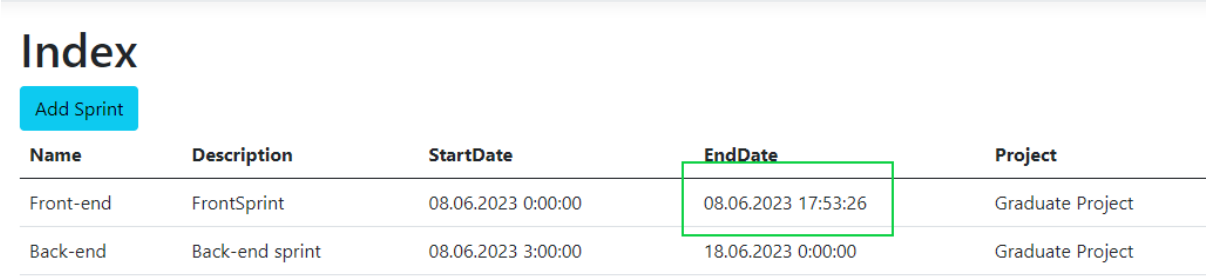


Рисунок 3.5.6 – Дані після завершення спринту “Front-end”

Решта функціоналу це операції взаємодії з моделями, вони працюють аналогічно до створення та редагування завдань, тому прийнято рішення упустити даний момент з записки.

В даному розділі було розроблено логічну та фізичну моделі даних та розроблено інформаційну систему для обліку часу та витраченого часу на розробку програмного забезпечення. Також було протестовано функціональність системи на прикладі управління процесом розробки даної дипломної роботи.

ВИСНОВКИ

1. В результаті проведеної роботи, вдалося розробити просту, інтуїтивну інформаційну систему для обліку часу та витрат на розробку програмного забезпечення.
2. Система має достатній функціонал для виконання поставлених переді мною задач, має простий доступний візуальний інтерфейс та структуру. Розроблена система на відміну від існуючих проста в інсталяції, налагоджуванні і супроводженні. Для початку роботи в системі не потрібно спеціального навчання і команда зможе швидко її опанувати і почати ефективну працю.
3. В той же час система використовує для збереження даних потужну промислову СУБД, захищені протоколи обміну даними і сучасну платформу реалізації, що дозволяє надійне і безпечне зберігання даних компанії, а також унеможливлює несанкціонований доступ до даних компанії.
4. Використання сучасної платформи реалізації дозволяє просте розгортання системи у хмарі та легке масштабування доступними хмарними засобами.
5. Прототип розробленої інформаційної системи показав свою ефективність та корисність на тестовому проєкті. Вона дозволяє збирати, систематизувати та аналізувати дані про передбачувані та фактичні часові витрати на розробку ПЗ, що сприяє покращенню планування та управління проєктами.
6. Результати розробки розміщені на GitHub за посиланням https://github.com/BohdanKhomik/ProjectManagementSystem
7. Застосування розробленої системи може мати значний позитивний вплив на організації, які займаються розробкою програмного забезпечення. Вона дозволить зекономити час, збільшити продуктивність команди та покращити управління проєктами.

8. Враховуючи швидкий розвиток сфери програмного забезпечення та постійно зростаючі вимоги до ефективності та якості розробки, інформаційна система для обліку часу та витрат на розробку програмного забезпечення має великий потенціал для впровадження у практику різних компаній.

У майбутньому рекомендується дослідити можливості розширення інформаційної системи, включаючи функції прогнозування та оптимізації витрат на розробку програмного забезпечення. Також варто розглянути можливість інтеграції з іншими інструментами управління проєктами та ресурсами для ще більшої ефективності та автоматизації процесів.

Також в систему необхідно впровадити адаптивний доступ до функцій та повноцінну систему реєстрації користувачів. Це зробить систему більш захищеною, та буде краще забезпечувати цілісність даних.

Загалом, розробка інформаційної системи для обліку часу та витрат на розробку програмного забезпечення є важливим кроком у покращенні управління проєктами та забезпеченні успіху у сфері програмної розробки. Вона допомагає забезпечити точність, ефективність та відкритість у витратних процесах, що веде до поліпшення результатів розробки.

ДЖЕРЕЛА:

1. Jira Software - URL: https://www.atlassian.com/ru/software/jira
2. Jira (software) Wikipedia - URL: https://en.wikipedia.org/wiki/Jira_(software)
3. Best Jira Alternatives - URL: https://clickup.com/blog/jira-alternatives/
4. Binfire Project Management Software - URL: https://www.binfire.com/
5. Basecamp Project Management Software - URL: https://basecamp.com/
6. Trello Project Management Software - URL: https://trello.com/uk
7. Asana Project Management Software - URL: https://asana.com/ru
8. Monday.com Project Management Software - URL: https://monday.com/lang/ru
9. ClickUp Project Management Software - URL: https://clickup.com/
10. Model–view–controller Design Pattern - URL: Model-View-Controller Pattern - Wikipedia
11. Client–server model - URL: Client–server model - Wikipedia
12. Monolithic application - URL: Monolithic application - Wikipedia
13. Microservices – URL : Microservices - Wikipedia
14. Benefits of MVC and Microservices - URL: Benefits of MVC and Microservices
15. HTTP - URL: https://uk.wikipedia.org/wiki/HTTP
16. HTTPS - URL: https://uk.wikipedia.org/wiki/HTTPS
17. GitHub - URL: GitHub - Wikipedia
18. Microsoft Azure - URL: Microsoft Azure - Wikipedia
19. Azure SQL Database - URL: Azure SQL Database
20. ASP.NET - URL: https://uk.wikipedia.org/wiki/ASP.NET
21. C# - URL: https://ru.wikipedia.org/wiki/C_Sharp
22. Entity Framework - URL: https://en.wikipedia.org/wiki/Entity_Framework
23. Bootstrap - https://uk.wikipedia.org/wiki/Bootstrap
24. Ієрархічна модель даних - URL: Hierarchical database model - Wikipedia
25. Мережева модель даних - URL: Network model - Wikipedia
26. Реляційна модель даних - URL: Relational model - Wikipedia

27.Chart.js Library - URL: https://www.chartjs.org/docs/latest/

28.JQuery - https://uk.wikipedia.org/wiki/JQuery

29.Концептуальна модель даних - URL: Conceptual model - Wikipedia

30.Partial views in ASP.NET Core - URL: Partial views in ASP.NET Core

ДОДАТОК А

```
using GraduateWork.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using System.Reflection.Emit;

namespace GraduateWork.Data
{
   public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
   {
      public DbSet<Issue> Issues { get; set; }
      public DbSet<Project> Projects { get; set; }
      public DbSet<ProjectColumn> ProjectColumns { get; set; }
      public DbSet<ProjectUser> ProjectUsers { get; set; }
      public DbSet<Sprint> Sprints { get; set; }
      public DbSet<ApplicationUser> ApplicationUsers { get; set; }

      public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
         : base(options)
      {

      }

      protected override void OnModelCreating(ModelBuilder builder)
      {
         base.OnModelCreating(builder);

         //builder.Entity<ApplicationUser>().ToTable("AspNetUsers");

         builder.Entity<ProjectUser>()
            .HasOne(pu => pu.Project)
            .WithMany(p => p.ProjectUsers)
            .HasForeignKey(pu => pu.ProjectId);

         builder.Entity<ProjectUser>()
            .HasOne(pu => pu.User)
            .WithMany(au => au.ProjectUsers)
            .HasForeignKey(pu => pu.UserId);

         builder.Entity<Project>()
```

```
            .HasMany(p => p.Sprints)
            .WithOne(s => s.Project)
            .HasForeignKey(s => s.ProjectId)
            .OnDelete(DeleteBehavior.Restrict);

        builder.Entity<Project>()
            .HasMany(p => p.ProjectColumns)
            .WithOne(pc => pc.Project)
            .HasForeignKey(pc => pc.ProjectId)
            .OnDelete(DeleteBehavior.Restrict);

        builder.Entity<ProjectColumn>()
            .HasMany(pc => pc.Issues)
            .WithOne(i => i.ProjectColumn)
            .HasForeignKey(i => i.ColumnId)
           .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<Sprint>()
            .HasMany(s => s.Issues)
            .WithOne(i => i.Sprint)
            .HasForeignKey(i => i.SprintId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<Issue>(b =>
        {
            b.HasOne(i => i.Assignee)
            .WithMany(u => u.AssignedIssues)
            .HasForeignKey(i => i.AssigneeUserId)
            .OnDelete(DeleteBehavior.Restrict);

            b.HasOne(i => i.Reporter)
            .WithMany(u => u.ReportedIssues)
            .HasForeignKey(i => i.ReporterUserId)
            .OnDelete(DeleteBehavior.Restrict);
        });

    }
  }
}
```

ДОДАТОК Б

Код класу ProjectUsersController :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using GraduateWork.Data;
using GraduateWork.Models;

namespace GraduateWork.Controllers
{
   public class ProjectUsersController : Controller
   {
      private readonly ApplicationDbContext _context;

      public ProjectUsersController(ApplicationDbContext context)
      {
         _context = context;
      }

      // GET: ProjectUsers
      public async Task<IActionResult> Index(int? Id)
      {
         var applicationDbContext = _context.ProjectUsers.Where(p => p.ProjectId
== Id).Include(p => p.Project).Include(p => p.User);
         return View(await applicationDbContext.ToListAsync());
      }

      // GET: ProjectUsers/Create
      public async Task<IActionResult> CreateOrEdit(int? id = null)
      {
         ViewData["ProjectId"] = new SelectList(_context.Projects, "Id", "Id");
         IEnumerable<Project> projects = _context.Projects;
         ViewBag.Project = projects;

         var users = _context.ApplicationUsers.ToList();
         ViewData["UserId"] = new SelectList(users, "Id", "Id");
         ViewBag.Users = users;
         ProjectUser? projectUser = null;
         if (id == null)
         {
```

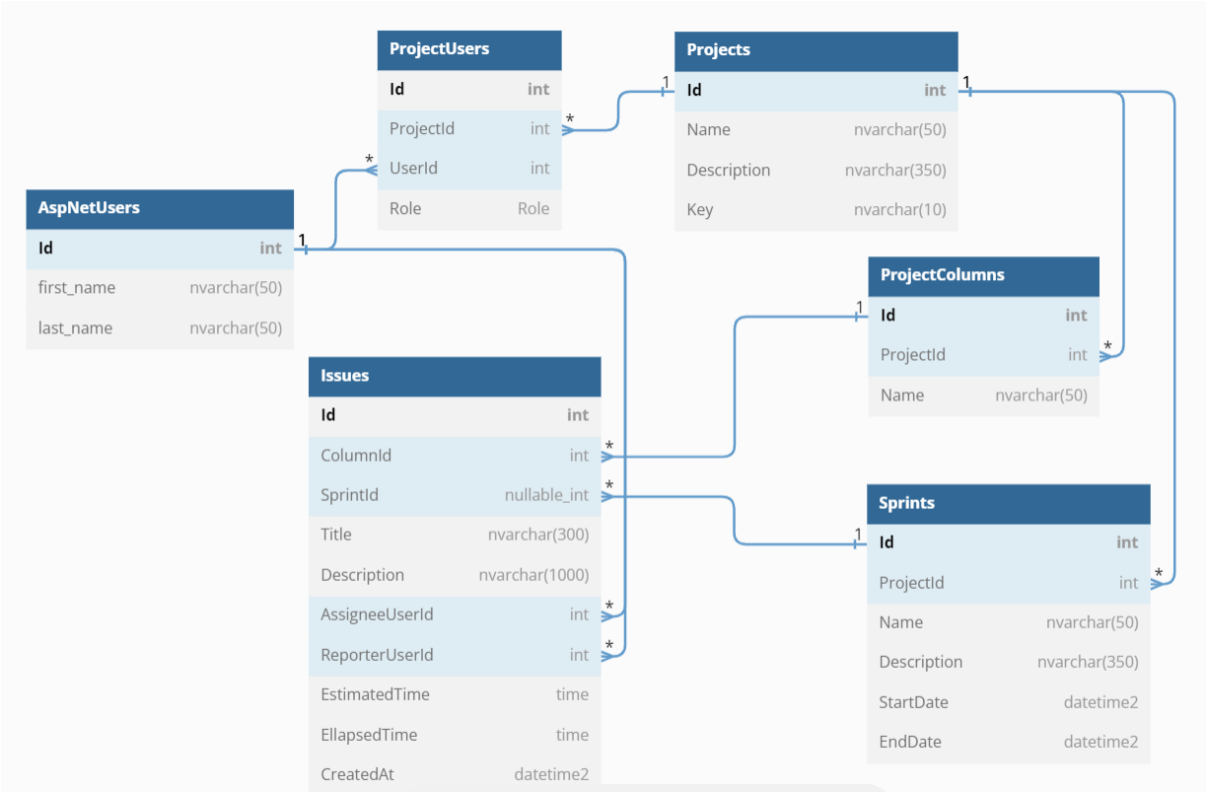
```
            projectUser = new ProjectUser();
        }
        else
        {
            projectUser = await _context.ProjectUsers.FindAsync(id);
            if (projectUser == null)
            {
                return NotFound();
            }
        }
        return PartialView("CreateOrEditProjectUser", projectUser);
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> CreateOrEdit(ProjectUser projectUser)
    {
        ViewData["ProjectId"] = new SelectList(_context.Projects, "Id", "Id",
projectUser.ProjectId);
        ViewData["UserId"] = new SelectList(_context.ApplicationUsers, "Id",
"Id", projectUser.UserId);
        if (projectUser.Id == 0)
        {
            if (ModelState.IsValid)
            {
                _context.Add(projectUser);

            }
            else
            {
                return BadRequest("Not valid");
            }
        }
        else
        {
            if (ModelState.IsValid)
            {
                _context.Update(projectUser);
            }
            else
            {
                return BadRequest("Not valid");
            }
        }
        await _context.SaveChangesAsync();
        return RedirectToAction("Index", new { Id = projectUser.ProjectId });
```

```
}
public async Task<IActionResult> Details(int? id)
{
    if (id == null || _context.ProjectUsers == null)
    {
        return NotFound();
    }

    var projectUser = await _context.ProjectUsers
        .Include(p => p.Project)
        .Include(p => p.User)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (projectUser == null)
    {
        return NotFound();
    }

    return PartialView("Details", projectUser);
}
// GET: ProjectUsers/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null || _context.ProjectUsers == null)
    {
        return NotFound();
    }

    var projectUser = await _context.ProjectUsers
        .Include(p => p.Project)
        .Include(p => p.User)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (projectUser == null)
    {
        return NotFound();
    }

    return PartialView("Delete", projectUser);
}

// POST: ProjectUsers/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    if (_context.ProjectUsers == null)
```

```
            {
                return Problem("Entity set 'ApplicationDbContext.ProjectUsers'  is
null.");
            }
            var projectUser = await _context.ProjectUsers.FindAsync(id);
            if (projectUser != null)
            {
                _context.ProjectUsers.Remove(projectUser);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
    }
}
```

ПЕРЕЛІК ГРАФІЧНОГО МАТЕРІАЛУ

Схема бази даних:



Скріни бізнес логіки проекту:
Методи Index() та Board(int? Id, int? sprintId = null):

```
public async Task<IActionResult> Index()
{
    return View( await _dbContext.Projects.ToListAsync());
}

[HttpGet]
0 references
public async Task<IActionResult> Board(int? Id, int? sprintId = null)//Project id
{
    ViewData["SelectList"] = new SelectList(_dbContext.Sprints, dataValueField:"Id", dataTextField:"Name");
    IEnumerable<Sprint> sprints = await _dbContext.Sprints.Where(s => s.ProjectId == Id).ToListAsync();
    ViewBag.Sprint = sprints;


    DateTime current = DateTime.Now;

    if (sprintId == null)
    {
        sprintId = await _dbContext.Sprints
            .Where(s => s.ProjectId == Id && s.EndDate > current && s.StartDate < current)
            .Select(s => s.Id)
            .FirstOrDefaultAsync();
    }

    var columns = _dbContext.ProjectColumns
        .Include(c => c.Issues.Where(i => i.SprintId == sprintId))
        .Where(c => c.ProjectId == Id);
    return View(await columns.ToListAsync());
}
```

Методи EndSprint(int Id), FinishSprint(int Id), ManageUsers(int? Id):

```
[HttpGet]
0 references
public async Task<IActionResult> EndSprint(int Id)
{
    DateTime current = DateTime.Now;
    ViewBag.ID = Id;
    var sprint_context = _dbContext.Sprints.Where(s => s.EndDate > current && s.ProjectId == Id).Include(c => c.Project).ToListAsync();
    return PartialView(await sprint_context);
}

[HttpGet]
0 references
public async Task<IActionResult> FinishSprint(int Id)//Sprint id
{
    Sprint sprint = await _dbContext.Sprints.Where(s => s.Id == Id).FirstOrDefaultAsync();
    sprint.EndDate = DateTime.Now;
    _dbContext.Update(sprint);
    await _dbContext.SaveChangesAsync();
    var project_id = await _dbContext.Projects.Where(s => s.Id == sprint.ProjectId).FirstOrDefaultAsync();
    return RedirectToAction("Board", "Home", project_id);
}

[HttpGet]
0 references
public async Task<IActionResult> ManageUsers(int? Id)//Project id
{
    var project_id = await _dbContext.Projects.Where(p => p.Id == Id).FirstOrDefaultAsync();
    return RedirectToAction("Index","ProjectUsers", project_id);
}
```

Метод SprintStatistic(int? Id):

```
[HttpGet]
0 references
public async Task<IActionResult> SprintStatistic(int? Id)
{
    var issues_context = await _dbContext.Issues.Where(Issues => Issues.SprintId == Id).ToListAsync();
    List<decimal> estimatedTimes = new List<decimal>();
    List<decimal> elapsedTimes = new List<decimal>();

    List<String> issueNames = new List<String>();
    decimal totalEstimatedTime = 0;
    decimal totalElapsedTime = 0;

    foreach (var issue in issues_context)
    {
        decimal estimateDecimalValue = Convert.ToDecimal(issue.EstimatedTime.TotalHours);
        decimal elapsedDecimalValue = Convert.ToDecimal(issue.EllapsedTime?.TotalHours ?? 0);
        totalEstimatedTime += estimateDecimalValue;
        totalElapsedTime += elapsedDecimalValue;
        issueNames.Add(issue.Title);
        estimatedTimes.Add(estimateDecimalValue);
        elapsedTimes.Add(elapsedDecimalValue);
    }

    estimatedTimes.Add(totalEstimatedTime);
    elapsedTimes.Add(totalElapsedTime);
    issueNames.Add("TotalTime");

    ViewBag.EstimatedTimes = estimatedTimes;
    ViewBag.ElapsedTimes = elapsedTimes;
    ViewBag.IssueNames = issueNames;
    DateTime current = DateTime.Now;
    var sprint = await _dbContext.Sprints.Where(s => s.Id == Id).FirstOrDefaultAsync();
    if (sprint == null)
    {
        ViewBag.Message = "You didn't select sprint!!!";
    }
    else {
    ViewBag.Message = "This is a statistic from " + sprint.Name + " sprint! In the last column you can see total count" ;
    }
    return View();
}
```

Метод projectStatistic(int? Id):

```
[HttpGet]
0 references
public async Task<IActionResult> projectStatistic(int? Id)
{
    var issues_context = await _dbContext.Issues.Where(Issues => Issues.ProjectColumn.ProjectId == Id).ToListAsync();
    List<decimal> estimatedTimes = new List<decimal>();
    List<decimal> elapsedTimes = new List<decimal>();
    List<String> issueNames = new List<String>();
    decimal totalEstimatedTime = 0;
    decimal totalElapsedTime = 0;

    foreach (var issue in issues_context)
    {
        decimal estimateDecimalValue = Convert.ToDecimal(issue.EstimatedTime.TotalHours);
        decimal elapsedDecimalValue = Convert.ToDecimal(issue.EllapsedTime?.TotalHours ?? 0);
        totalEstimatedTime += estimateDecimalValue;
        totalElapsedTime += elapsedDecimalValue;
        estimatedTimes.Add(estimateDecimalValue);
        elapsedTimes.Add(elapsedDecimalValue);
        issueNames.Add(issue.Title);
    }

    estimatedTimes.Add(totalEstimatedTime);
    elapsedTimes.Add(totalElapsedTime);
    issueNames.Add("TotalTime");
    ViewBag.IssueNames = issueNames;
    ViewBag.EstimatedTimes = estimatedTimes;
    ViewBag.ElapsedTimes = elapsedTimes;
    var project = await _dbContext.Projects.Where(p => p.Id == Id).FirstOrDefaultAsync();
    ViewBag.Message = "This is a statistic from " + project.Name + " project! In the last column you can see total count";
    return View("SprintStatistic");
}
```