Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Ткаченка Андрія Сергійовича*
(ПІБ)

академічної групи *122-19-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка сайту-системи відстеження помилок, призначеної для управління проектами та організації робочого процесу з використанням фреймворків .Net та Angular*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В.*			
розділів:				
спеціальний	*доц. Кабак Л.В.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-2* (група) *Ткаченка А.С.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка сайту-системи відстеження помилок, призначеної для управління проектами та організації робочого процесу з використанням фреймворків .Net та Angular*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Ткаченко А.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 90 с., 64 рис., 3 дод., 20 джерел.

Об'єкт розробки: сайт-система відстеження помилок, призначена для управління проектами та організації робочого процесу з використанням фреймворків .Net та Angular.

Мета кваліфікаційної роботи: розробити зручний веб-сайт, використовуючи фреймворки .Net та Angular. Забезпечити користувачів функціоналом для планування задач.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної області, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів, було виконано аналіз вже існуючих рішень.

У другому розділі описано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування підсистеми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Актуальність веб-додатку для відстеження помилок визначається великим попитом користувачів та цілих комерційних організацій на подібні інструменти.

Список ключових слів: WEB-ДОДАТОК, САЙТ, SPA, SASS, ANGULAR, TYPESCRIPT, HTML, .NET, ASP.NET, ENTITY FRAMEWORK, POSTGRESQL, SCRUM, AGILE, KANBAN.

ABSTRACT

Explanatory note: 90 pgs, 64 pics, 3 apps, 20 srcs.

Object of development: a bug tracking website designed for project management and workflow organisation using the .Net and Angular frameworks.

The purpose of the qualification work: to develop a user-friendly website using the .Net and Angular frameworks. Provide users with functionality for task planning.

The introduction discusses the analysis and current state of the problem, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic and clarifies the task statement.

The first chapter analyses the subject area, determines the relevance of the task and the purpose of the development, develops the task statement, sets requirements for software implementation, technologies and software tools, and analyses existing solutions.

The second section describes the choice of the development platform, the design and development of the application, the description of the algorithm and structure of the subsystem, the input and output data, the characteristics of the composition of the parameters of technical means, the description of the application call and download, and the application operation.

In the economic section, the labour intensity of the developed information subsystem is determined, the cost of the work on creating the application is calculated, and the time for its creation is estimated.

The relevance of the web application for bug tracking is determined by the high demand of users and entire commercial organisations for such tools.

List of keywords: WEB APPLICATION, WEBSITE, SPA, SASS, ANGULAR, TYPESCRIPT, HTML, .NET, ASP.NET, ENTITY FRAMEWORK, POSTGRESQL, SCRUM, AGILE, KANBAN.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК – персональний комп’ютер;

БД – база даних;

VS – Visual Studio;

HTML – HyperText Markup Language;

CSS – Cascading Style Sheets;

SASS – Syntactically Awesome Style Sheets;

HTTP – Hypertext Transfer Protocol;

UI – User Interface;

UX – User Experience;

SPA – Single Page Application;

JSON – JavaScript Object Notation;

REST – Representational State Transfer;

JWT – JSON Web Token;

Backend – серверна частина;

Frontend – клієнтська частина;

ПЗ – програмне забезпечення;

ВСТУП

Ця кваліфікаційна робота присвячена розробці сайту-системи відстеження помилок, який призначений для управління проектами та організацією робочого процесу. Розробка системи здійснюється з використанням фреймворків .Net та Angular.

У сучасному світі, де швидкість та точність виконання проектів мають вирішальне значення, ефективне управління проектами та організація робочого процесу стають ключовими факторами успіху. Недоліки та помилки в процесі розробки можуть призвести до затримок, перешкоджати співпраці між командами та погіршувати якість продукту. Тому виникає необхідність у створенні ефективної системи, яка б допомагала відстежувати, керувати та вирішувати проблеми, що виникають у процесі розробки.

Метою цієї кваліфікаційної роботи є розробка сайту-системи відстеження помилок, яка буде забезпечувати управління проектами та організацію робочого процесу. Система буде побудована з використанням сучасних фреймворків .Net та Angular, що надають широкий функціонал та зручний інтерфейс.

Актуальність даної теми обумовлена необхідністю покращення ефективності робочого процесу та управління проектами. Створення системи, яка дозволить відстежувати помилки, керувати завданнями та спрощувати комунікацію між учасниками проекту, дозволить збільшити продуктивність роботи, знизити час втрат і покращити якість результатів.

Завданнями даної роботи є:

- Аналіз організації роботи з управління проектами та організації робочого процесу;
- Визначення вимог до функціоналу системи відстеження помилок;
- Розробка алгоритму, бази даних та реалізація програмного забезпечення;

Результатом роботи буде функціональна система, яка надасть можливість ефективно відстежувати помилки, керувати завданнями та полегшить

організацію робочого процесу. Розроблена система буде корисною для фахівців та команд будь яких професій, що дозволить цьому веб додатку стати популярним серед багатьох людей, котрі планують свою роботу та дозвілля.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

Планування для проектів, організацій та бізнесу є ключовим елементом успіху та правильного функціонування. Ефективне управління та планування дозволяють досягати поставлених цілей, забезпечуючи оптимальне використання ресурсів та мінімізацію ризиків.

Для досягнення оптимального управління та планування використовуються різні методології. Однією з них є Agile - ітеративна та інкрементальна методологія, яка широко використовується в розробці програмного забезпечення та проектних роботах.

Ітеративність означає, що розробка продукту відбувається через серію повторюваних ітерацій або циклів. Кожна ітерація представляє собою короткий проміжок часу, під час якого виконується певний обсяг робіт. У кожній ітерації команда займається плануванням, розробкою, тестуванням та впровадженням нового функціоналу або фрагменту продукту. Після закінчення ітерації отриманий результат оцінюється, відбувається ретроспектива і планується наступна ітерація. Цей цикл повторюється протягом усього процесу розробки.

Інкрементальність означає поступове розширення функціональності продукту шляхом додавання нових інкрементів або версій. Замість того, щоб очікувати на повний та закінчений продукт, Agile підхід передбачає, що продукт розвивається шляхом послідовного додавання нових функцій, можливостей або вдосконалень з кожною ітерацією. Кожен інкремент є придатним для використання та може бути представлений користувачам або зацікавленим сторонам для отримання зворотного зв'язку.

Agile була придумана з метою підвищення гнучкості, здатності швидко реагувати на зміни та покращення комунікації між учасниками проекту.

Основними принципами Agile є:

- Постійна зміна.
- Взаємодія замість формальності.
- Самоорганізація команди.
- Спрямованість на результат.

Agile цикл розробки включає послідовність кроків, що відбуваються від планування до запуску продукту. Цей цикл можна уявити як неперервну петлю, де кожен етап зміцнює результати попереднього, сприяючи створенню високоякісного та гнучкого програмного рішення.

Існують такі базові етапи як: планування, дизайн, розробка, тестування, розгорнення, огляд та запуск (рис. 1.1.).

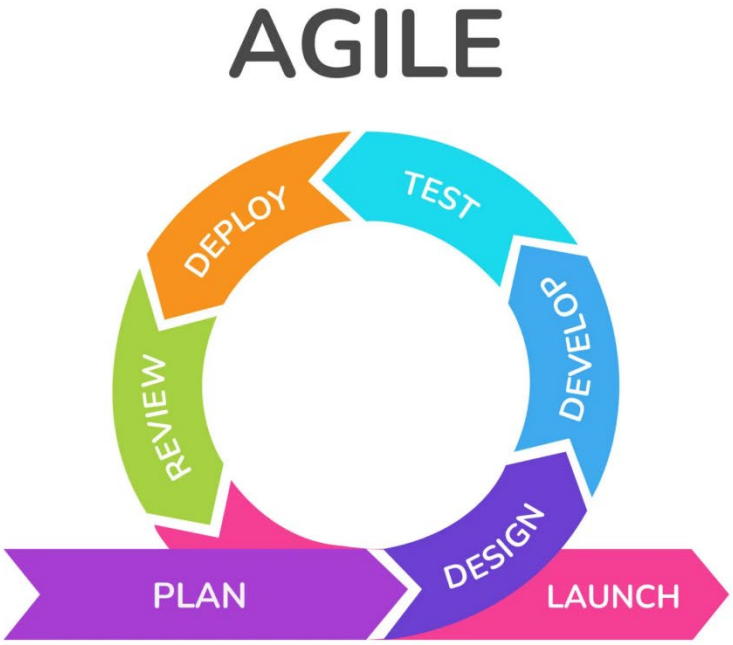


Рис. 1.1. Структура Agile циклу.

У методології Agile використовуються різні фреймворки для управління та планування проектів. Два найпопулярніші фреймворки - Scrum та Kanban - виявилися дуже ефективними для багатьох команд.

Scrum — це ітеративний фреймворк, який дозволяє команді організувати свою роботу у короткі періоди, називані спринтами. Він забезпечує чітку рольову розподіл роботи, встановлення приоритетів завдань та постійний взаємозв'язок між членами команди.

Процес розробки складається з беклогу застосунку - задач на створення застосунку, планування спрінту, беклогу спрінта - задач на конкретний спрінт, кожнодневного скраму - коротких обговорень задач, огляду спрінту та його ретроспективи - аналізу спрінту (рис. 1.2.).

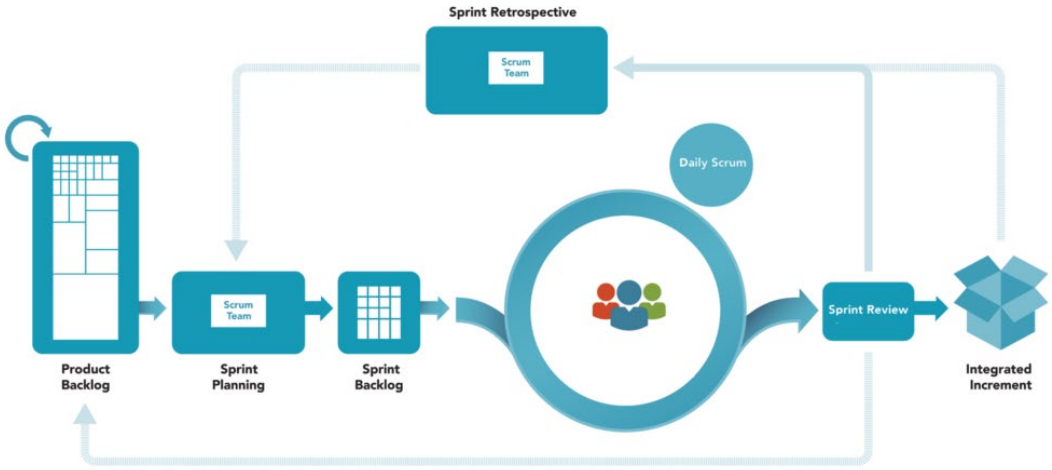


Рис. 1.2. Процес розроби фреймворку Scrum.

Kanban — це фреймворк для управління робочим процесом, який базується на візуальному представленні завдань та контролі за їх прогресом. Цей фреймворк походить з японського виробництва та був вперше використаний в Toyota для оптимізації виробничих процесів. Основними особливостями фреймворку Kanban є: візуалізація, обмеження роботи в процесі, управління потоком роботи, очевидні правила та процедури, зворотний зв’язок та покращення (рис. 1.3.).

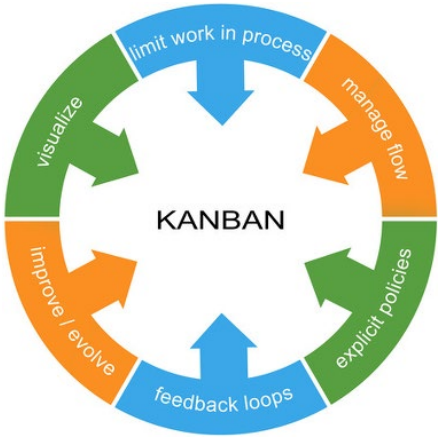


Рис. 1.3. Основні характеристики фреймворку Kanban.

Хоча Scrum та Kanban є часто використовуваними фреймворками Agile, вони мають свої відмінності. Обидва фреймворки зосереджені на постійній взаємодії команди та гнучкому реагуванні на зміни, але мають різний акцент.

Scrum передбачає фіксовану тривалість спринту та строго визначені ролі, такі як:

- Scrum Master — відповідає за координацію та підтримку команди розробки, забезпечуючи впровадження Scrum-процесу та досягнення максимальної продуктивності та Product Owner.
- Product Owner — відповідає за визначення, пріоритизацію та управління вимогами до продукту, забезпечуючи створення цінної та відповідної потребам робочої продукції. Він виступає як представник бізнесу або клієнта, встановлюючи пріоритети та забезпечуючи створення високоякісного продукту, який задовольняє потреби користувачів.

Виконання завдань у Scrum зазвичай відбувається впродовж спринту, іноді з можливістю вносити зміни на початку нового спринту.

Kanban, натомість, наголошує на візуалізації робочого процесу та обмеженні одночасно виконуваних завдань. Колонки на Kanban дошці дозволяють команді відстежувати потік роботи та виявляти можливі затори.

При використанні як Scrum, так і Kanban, важливою складовою є використання Kanban дошки для управління завданнями та візуалізації робочого процесу.

Kanban дошка є інструментом для візуалізації робочого процесу та керування потоком завдань. Вона може бути фізичною дошкою, використовувати електронні інструменти або спеціалізовані онлайн-сервіси. Дошка складається з колонок, які відображають стадії виконання завдань, наприклад, "Список задач", "В розробці", "Перевірка" та "Виконано" (рис. 1.4.).

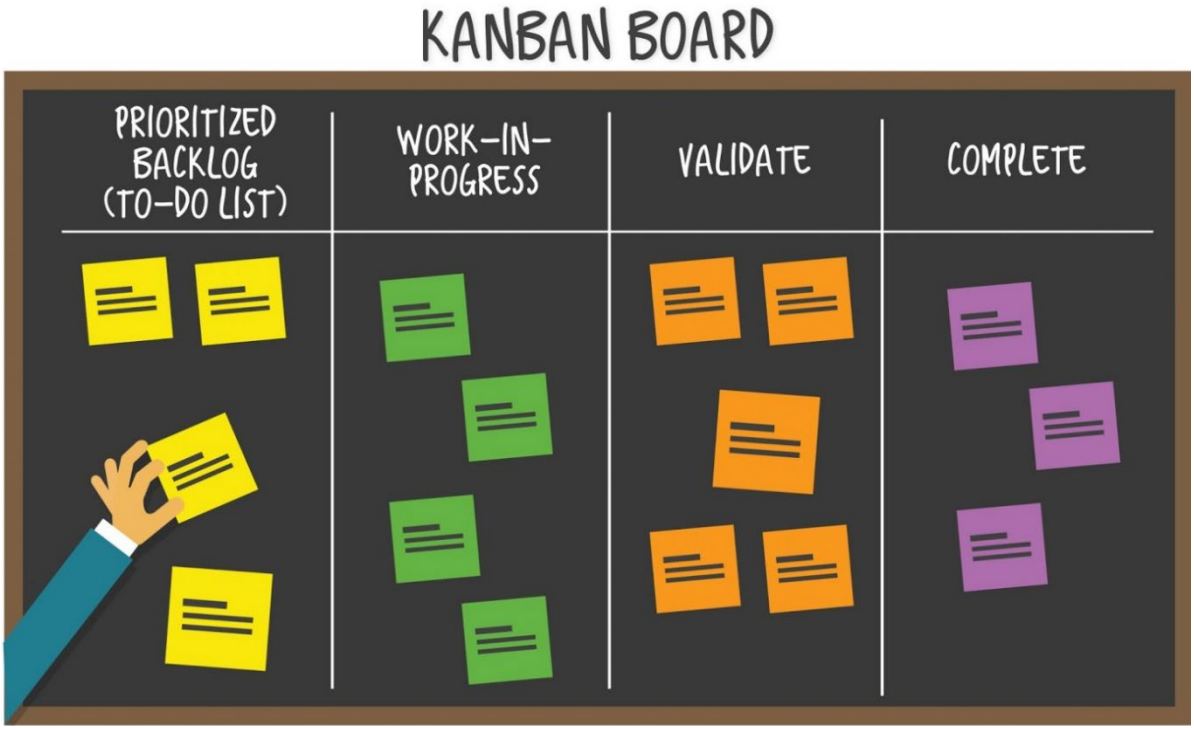


Рис. 1.4. Структура Kanban-дошки.

Kanban дошки широко використовуються в багатьох галузях та командах для планування та керування проектами, управління завданнями та потоком роботи. Вони дозволяють відстежувати прогрес, виділяти пріоритети та забезпечувати взаємодію між членами команди.

Згідно до популярності використання методології Agile, існує багато сайтів для планування, котрі використовують Kanban дошки. Основними можна виділити Jira, Asana та Trello. Розглянемо плюси й мінуси кожного додатку.

Jira — це одна з найпопулярніших системм управління проєктами, яка дозволяє закривати майже всі завдання PM-а в рамках одного інструмента: від планування до контролю процесів та результатів.

Основні плюси Jira:

- Широкий набір функцій. Ви зможете адаптувати Jira до роботи з проєктами практично будь-якої складності.
- Великий інтеграційний потенціал. Jira приязна до інтеграції з різними сервісами: Github, Salesforce, Outlook, Slack, Gmail, Teams.
- Можливість розширення функціоналу, використовуючи різні плагіни. Найбільш популярні — Tempo, Script Runner, EazyBI, Big Picture, Structure.

- Можливість роботи за методологіями Scrum або Kanban «з коробки». Приклад (рис. 1.5.).

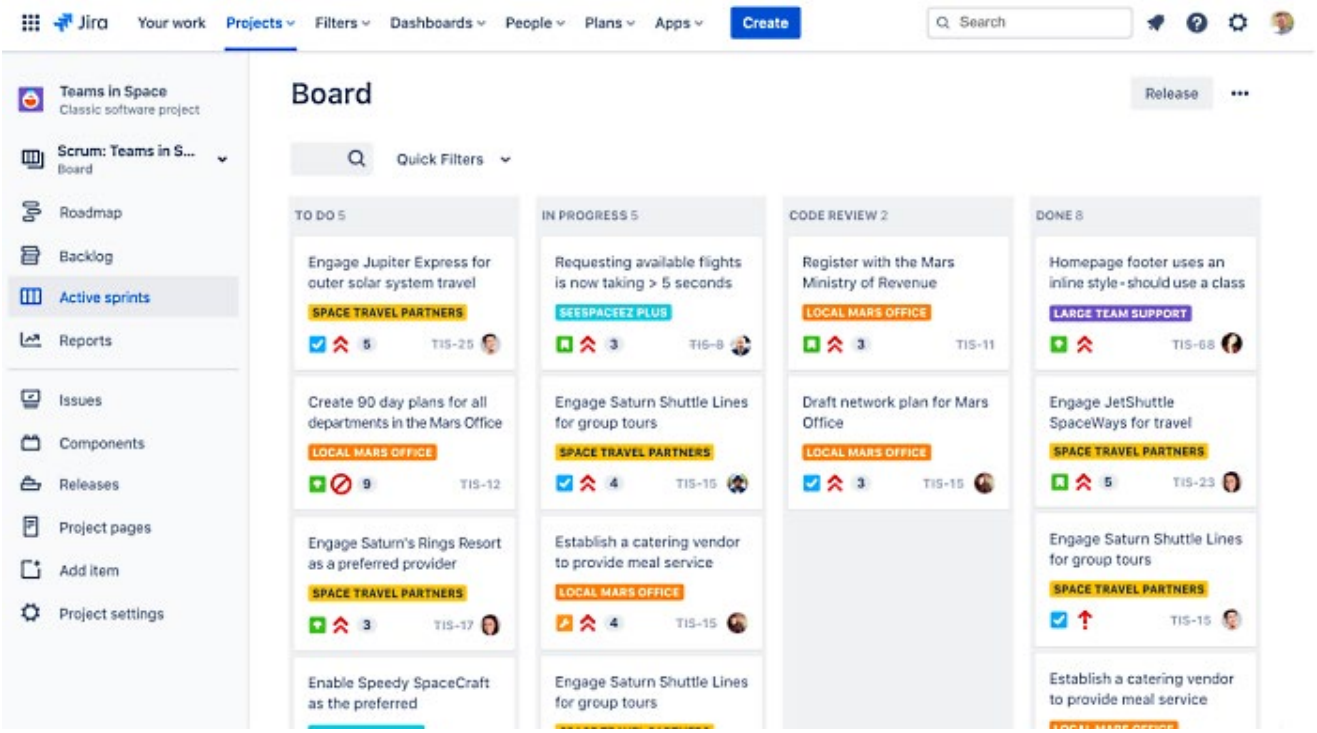


Рис. 1.5. Інтерфейс спрінту Kanban-дошки Jira.

Jira має кілька недоліків, серед яких тривалий процес налаштування під конкретні робочі процеси та складний інтерфейс. Безкоштовний план також має свої обмеження, такі як відсутність планування потужностей та обмежена автоматизація, а також обмежена можливість використання лише в одному проекті. Крім того, платна підписка коштує недешево, що робить цей додаток більш комерційним, ніж дружелюбним для звичайних користувачів.

Asana — це інструмент для управління проектами та спільної роботи, який надає командам зручність та ефективність при плануванні та виконанні робочих завдань. Приклад інтерфейсу Asana (рис. 1.6.).

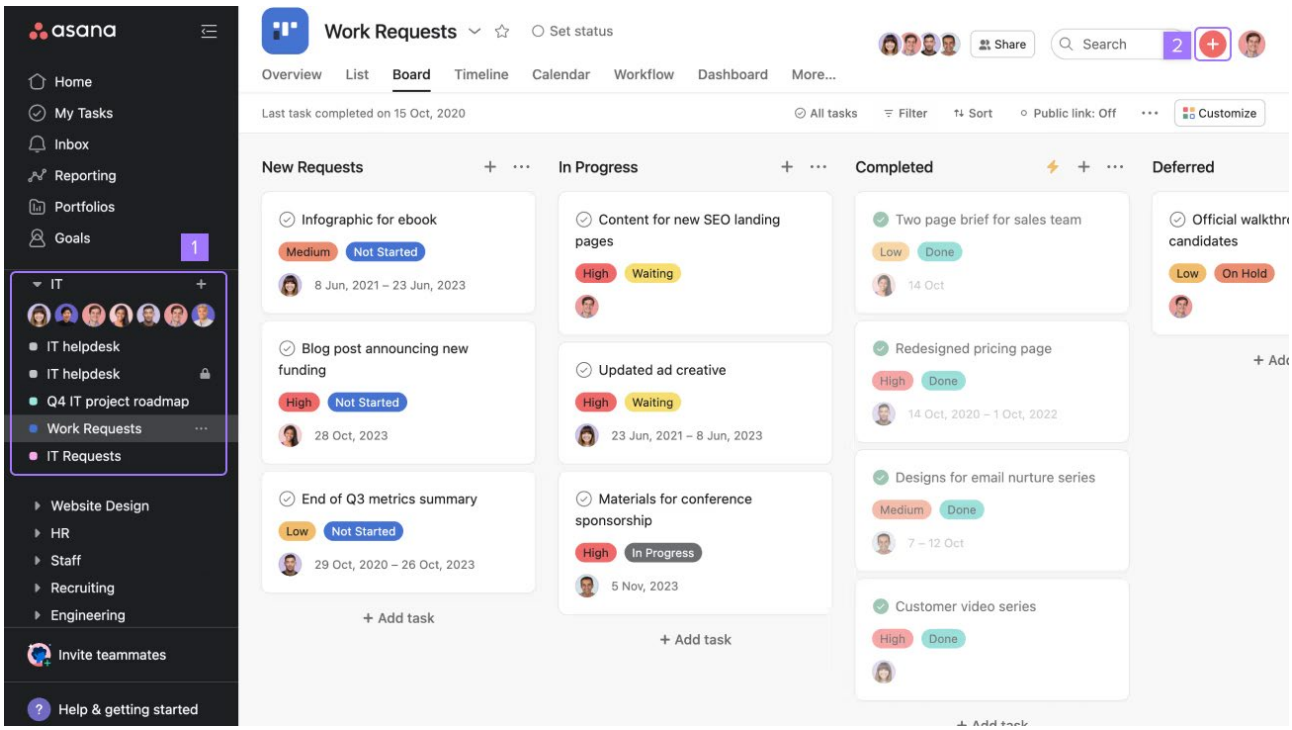


Рис. 1.6. Інтерфейс Kanban-дошки Asana.

Ось деякі плюси та мінуси Asana порівняно з іншими аналогами:

Плюси Asana:

- Інтуїтивний інтерфейс: Asana пропонує зручний та легкий у використанні інтерфейс.
- Гнучкість та налаштовування: Asana надає широкі можливості налаштування робочих процесів та завдань.
- Спільна робота: Asana забезпечує зручні засоби спілкування, спільної роботи та обміну документами.
- Відстежування прогресу: Asana надає зручні засоби для відстежування прогресу завдань, статистики виконання та звітності.

З мінусів, як і в Jira можна виділити високу вартість та обмежений багатьма функціями безкоштовний план.

Trello — це інструмент для управління проектами та спільної роботи, який базується на концепції Kanban дошки та карток. Він дозволяє командам створювати, організовувати та відстежувати завдання та проекти у вигляді карток, які можна переміщувати між списками на дошці (рис. 1.6.). Цікаво, що

Trello також належить Atlassian, тій самій компанії, яка володіє Jira. Але це не облегшена версія Jira.

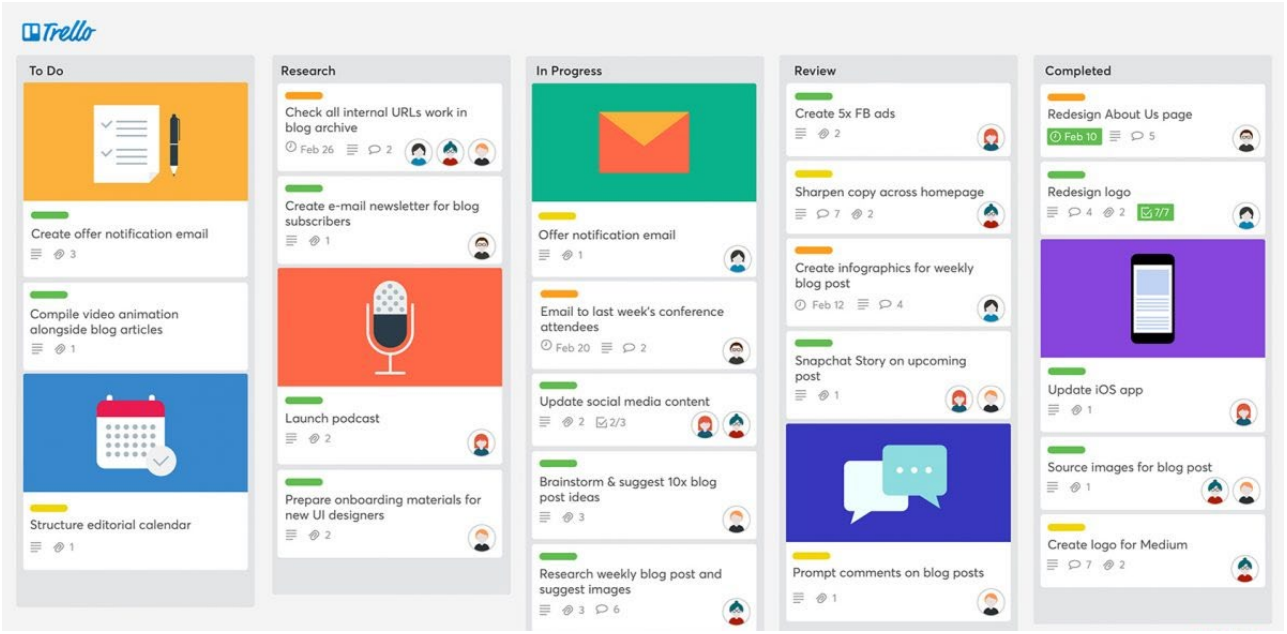


Рис. 1.7. Інтерфейс Kanban-дошки Trello.

Ось деякі плюси та мінуси Trello:

Плюси Trello:

- Простота використання: Trello має дуже інтуїтивний та простий інтерфейс, що робить його легким у використанні для новачків та досвідчених користувачів.
- Гнучкість та налаштовування: Trello дозволяє командам створювати та налаштовувати дошки, списки та картки під свої потреби.
- Візуальний підхід: Завдяки дошкам та карткам, Trello надає візуальне відображення прогресу, статусу та приоритету завдань.
- Інтеграція з іншими інструментами: Trello має можливість інтеграції з іншими популярними інструментами та сервісами, такими як Slack, Google Drive, Dropbox.

Мінуси Trello:

- Обмежені функціональні можливості: У порівнянні з деякими іншими інструментами для управління проектами, Trello може бути менш функціональним, особливо для складних та багатоетапних проектів.
- Обмежена структура завдань: Trello має просту структуру, що складається лише з дошок, списків та карток.
- Відсутність детальних засобів звітності: У порівнянні з деякими іншими інструментами, Trello має обмежені можливості зі створення детальних звітів та аналітики проектів.

1.2 **Призначення розробки та галузь застосування.**

Темою бакалаврської кваліфікаційної роботи виступає: «Створення сайту-системи відстеження помилок, призначеної для управління проектами та організації робочого процесу з використанням фреймворків .Net та Angular». В якості функціональної основи взятий фреймворк Scrum з використанням Kanban дошки.

Метою роботи є створення веб-сайту, який представляє собою інструмент користувача для управління організаціями, проєктами організацій, спрінтами проєктів та Kanban дошками спрінтів.

Головними критеріями розроблювального сайту є:

- Зручність в використанні.
- Легкість в освоєнні.
- Проста розгортування додатку.
- Використання сучасних фреймворків.

Сайт-система призначена для:

- Візуалізації планування задач.
- Відстеження помилок.
- Управління проєктами та організації робочого процесу.

Веб-додаток дає можливість реєструватися, авторизуватися, створювати організації, проєкти, спрінти - візуалізовуючи їх прогрес за допомогою Kanban дошок.

1.3 **Підстави для розробки**

Відповідно до ОКХ та ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- ОКХ та ОПП за напрямом підготовки 6.050101 «Комп’ютерні науки»;
- Графік навчального процесу та навчальний план;
- Наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
- Завдання на кваліфікаційний проект на тему «Створення сайту-системи відстеження помилок, призначеної для управління проектами та організації робочого процесу з використанням фреймворків .Net та Angular»;

1.4 **Постановка завдання**

Метою кваліфікаційної роботи є розробка сайту-системи для відстеження помилок, яка буде використовуватися для управління проектами та організації робочого процесу.

Призначення системи полягає у забезпеченні ефективного планування проектів, відстеження помилок, а також підтримки робочого процесу використовуючи фреймворк Scrum та Kanban дошки для візуалізації виконання цілей, виправлення помилок та управління ресурсами.

Система має мати наступні характеристики:

- Для розробки мають використовуватися фреймворк .Net - для розробки backend частини, та Angular - для фронтенду.
- Розроблений сайт повинен мати можливіть для реєстрації та авторизації
- Автентифікація має бути розроблена з використанням стандарту JWT.
- Можливість реактивно оперувати даними та зберігати їх в базі даних за допомогою CRUD операцій.

Створення проєкту потребує:

- Аналізу існуючи рішень для управління проєктами, .
- Проектування архітектури backend та frontend частин.
- Проектування бази даних.
- Проектування інтерфейсу додатку.
- Реалізації підготовлених рішень.

1.5. Вимоги до програми або програмного виробу.

1.5.1. Вимоги до функціональних характеристик.

Результат розробки програмного забезпечення повинен підтримувати виконання наступних функціональних вимог:

- Інтуїтивно зрозумілий користувальницький інтерфейс (UI).
- Приємний та незаплутуючий користувацький досвід (UX).
- Дані повинні вводитися користувачем.
- Додаток повинен мати REST-ful архітектуру для можливості легкого розширення фінкціоналу.
- Сайт повинен підтримувати сучасні веб-браузери та працювати швидко як на стаціонарних комп’ютерах, так і на мобільних пристроях.

1.5.2 Вимоги до інформаційної безпеки.

Безпечна та безперервна робота програми потребує реалізацію:

- Автетифікації за допомогою JWT токену.
- Ізольованісті даних користувачів.
- Валідації даних, котрі зберігаються в базі даних.
- Хешованості паролів користувачів, для конфіденційності приватних даних.

1.5.3 Вимоги до складу та параметрів технічних засобів.

Щоб забезпечити стабільне функціонування додатку потрібна обчислювальна машина, на якій буде хоститися сервер. Вона повинна мати такі рекомендовані технічні характеристики:

- Операційна система: Linux, macOS, Windows. Потрібно впевнитися, що версія цих операційних систем підтримують інструмент для контейнеризації Docker.
- Процесор: 64-бітний процесор.
- Не менш ніж 4Гб оперативної пам’яті.
- Мінімум 20Гб вільного місця на жорсткому диску.
- Стабільне підключення до мережі інтернет.
- Підтримка апаратної віртуалізації (Intel VT-x або AMD-v).

1.5.4 Вимоги інформаційної та програмної сумісності.

Для забезпечення інформаційної та програмної сумісності потрібно враховувати наступні вимоги:

- Backend частина системи повинна бути розроблена з використанням фреймворку .Net, мови програмування C# та платформи ASP.Net. Це

забезпечить сумісність з існуючими інструментами та сервісами, побудованими на базі .Net технологій.

- Frontend частина системи повинна бути написана з використанням фреймворку Angular та мови програмування Typescript. Використання цих технологій забезпечить сумісність з сучасними веб-браузерами та платформами.
- Для візуалізації інтерфейсу користувача повинен бути використаний препроцесор CSS — SASS. Це дозволить забезпечити зручну та гнучку стилізацію компонентів та елементів веб-інтерфейсу.
- В якості реляційної бази даних для системи була обрана PostgreSQL. Використання цієї бази даних забезпечить надійне зберігання та організацію даних, а також сумісність з відповідними інструментами та сервісами.

Для контейнеризації застосунку має використовуватися Docker. Це дозволить створити ізольоване середовище для розгортання та виконання системи, забезпечуючи портативність, легкість установки та масштабованість.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Функціональне призначення системи

В результаті виконання даної кваліфікаційної роботи буде створена сайт-система, яка матиме таке функціональне призначення:

- Управління проєктами організацій.
- Планування задач і спрінтів відповідно до методології Scrum.
- Візуалізація задач та помилок на Kanban дошці.

Користувачі системи зможуть:

- Зареєструватися та створювати організації для зберігання проєктів.
- Бути запрошеними як учасники до проєктів.
- Створювати спрінти, додавати задачі до них і розпочинати спрінти.
- Управляти задачами на Kanban дошці, переміщуючи їх між колонками.
- Завершувати спрінти після завершення роботи.
- Переглядати завершені спрінти в беклозі.

2.2. Опис застосованих математичних методів

У зв'язку з особливостями предметної області задачі, які не вимагають застосування математичних методів, під час розробки веб-додатку не було використано жодних математичних методів.

2.3. Опис використаних технологій та мов програмування

Даний веб-додаток складається з двох частин серверної та клієнтської. Для створення серверної частини було обрано використати платформу розробки програмного забезпечення .Net, мову програмування C# та фреймворк для розробки веб-застосунків ASP.NET. Для клієнтської – фреймворк Angular.

.NET – це безкоштовний, відкритий, крос-платформовий фреймворк для розробки сучасних додатків та потужних хмарних сервісів [1]. Він розроблений і підтримується на GitHub, дому для мільйонів розробників, які хочуть створювати великі речі разом. .NET допомагає створювати додатки для вебу, мобільних пристроїв, настільних комп'ютерів, хмарних сервісів та багато іншого. Завдяки великій підтримуючій екосистемі та потужним інструментам, .NET однією з найпродуктивніших платформ для розробників.

Серед численних переваг на функцій даної екосистеми можна виділити наступне:

- Багатопоточність: .NET надає зручні інструменти для роботи з багатопотоковим програмуванням, що дозволяє розробникам ефективно використовувати потоки і паралельні обчислення.
- JIT-компіляція (Just-In-Time Compilation): Код, написаний на мові програмування для .NET, компілюється в проміжний байт-код (IL - Intermediate Language). Під час виконання програми JIT-компілятор перетворює цей байт-код в машинний код, що дозволяє оптимізувати виконання програми.
- Сміттєзбирач (Garbage Collector): .NET включає автоматичний збирач мусору, який відповідає за автоматичне управління пам'яттю, звільняючи розмір пам'яті, який більше не використовується. Внаслідок цього не потрібно саморуч звільняти пам’ять при створенні динамічних об’єктів - .Net зробить це за нас автоматично, коли це буде потрібно.

Вибір .NET був обґрунтований його продуктивністю виконання, швидкістю розробки та активною спільнотою.

.NET підтримує використання різних мов програмування, серед яких основними є C#, VB.NET і F#. В даному проєкті я використав мову програмування C#, яка є не лише найпопулярнішої мову в екосистемі .Net, але й усьому світі.

Якою ж є ця одна з найкращих комерційних мов C#? C# (вимовляється як "сі шарп") – це сучасна, об'єктно-орієнтована та типобезпечна мова

програмування. C# дозволяє розробникам створювати безліч типів безпечних і надійних додатків, які працюють у середовищі .NET. C# має свої коріння в родині мов C і буде миттєво знайомою для програмістів на C, C++, Java та JavaScript [2].

Основні особливості C# допомагають створювати надійні додатки:

- Усі типи C#, включаючи примітивні, наприклад int і double, успадковуються від кореневого типу object, що дозволяє використовувати спільний набір операцій. Значення будь-яких типів можуть бути збережені та оброблені однаковим стандартним способом.
- Nullable типи дозволяють захистити від змінних, що не посилаються на виділені об'єкти. Це допомагає уникнути помилок, пов'язаних з невизначеністю значень.
- Обробка винятків надає структурований та розширюваний підхід до виявлення та відновлення помилок. Це дозволяє ефективно реагувати на непередбачені ситуації.
- Лямбда-вирази підтримують функціональні техніки програмування, що дозволяє реалізувати потужні алгоритми.
- Синтаксис LINQ (Language Integrated Query) надає загальний шаблон для роботи з даними з будь-яких колекцій. Це спрощує операції з обробкою даних.

Враховуючи зазначені переваги, використання C# є надійним вибором для даної роботи.

Для серверної частини не достатньо оприділитися з платформою та мовою програмування. Звичайно, що можна написати свій фреймворк для веб-серверу. Але навіщо придумувати велосипед, якщо вже існує рішення для платформи .NET і це є фреймворк ASP.NET.

ASP.NET – це безкоштовний, кросплатформений фреймворк для створення веб-додатків та сервісів з використанням .NET і C# з відкритим вихідним кодом на Github [3]. ASP.Net розширює платформу .NET з

інструментами та бібліотеками, котрі спеціалізуються на створенні веб-додатків.

З основних переваг використання ASP.NET можна виділити:

- Архітектурна модель MVC (Model-View-Controller): Розділення логіки на модель, відображення та контролер дозволяє покращити розширюваність коду і організацію проекту.
- Підтримка сторонніх бібліотек: Можливість використовувати готові бібліотеки спрощує та прискорює розробку веб-застосунків.
- Розширені налаштування з коробки: Використання контейнерів DI (Dependency Injection) спрощує керування залежностями між компонентами, полегшує розширення та тестування коду. Також є можливість налаштовувати маршрутизацію, безпеку та додавати middleware (компоненти, котрі виконуються між прийомом HTTP запиту і відправкою відповіді) для додаткової функціональності.

Для тестування створених в ASP.NET ендпоінтів я використовував Swagger – розширення, яке автоматично генерує документацію API та дозволяє перевіряти роботу ендпоінтів (API-точок доступу, URL адрес, до яких можна звернутися щоб отримати певні дані) безпосередньо з UI інтерфейсу (рис. 2.1.).

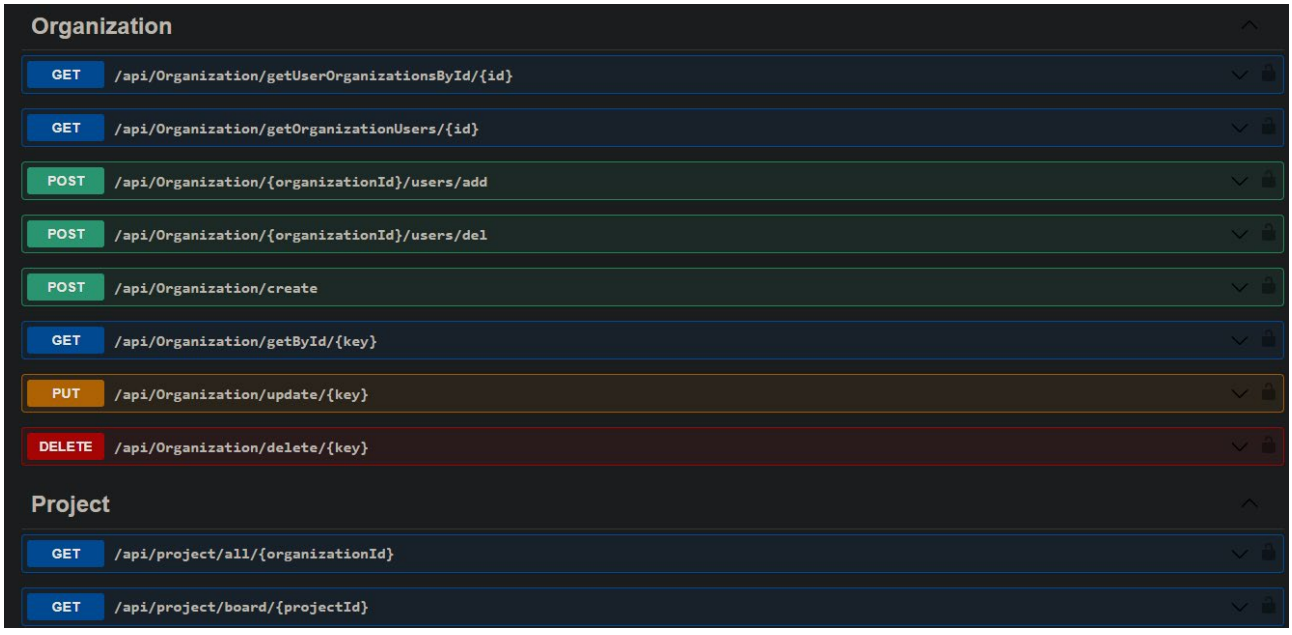


Рис. 2.1. Приклад автоматично згенерованої Swagger'ом документації API

Для збереження даних застосунку було обрано використати реляційну систему управління базами даних (СУБД) PostgreSQL з відкритим вихідним кодом. Вона є безкоштовною та використовує мову структурованих запитів (SQL) для визначення та маніпуляції даними.

Було обрано SQL базу даних замість NoSQL з таких причин:

- Потреба в структурованому зберіганні даних та підтримці складних запитів, що найкраще задовольняються реляційними базами даних.
- SQL бази даних забезпечують підтримку транзакційної безпеки та можливість забезпечити цілісність даних.
- Використання Entity Framework - ORM (Object-Relational Mapping) для .NET, додатково спрощує роботу з реляційними базами даних. Entity Framework дозволяє взаємодіяти з базою даних за допомогою об'єктно-орієнтованого підходу, а не безпосередньо з SQL-запитами. Він автоматично генерує SQL-запити на основі моделі даних, що дозволяє зосередитись на розробці функціональності додатку замість написання складних SQL-запитів.
- Бібліотека LINQ платформи .Net повністю підтримує Entity Framework та реляційні бази даних, що сприяє швидкому та ефективному розробленню додатку.

Для створення фронтенду (клієнтської частини) було використано фреймоворк Angular. Angular – це веб-фреймворк, розроблений компанією Google, який використовується для створення односторінкових SPA додатків (Single Page Applications). Він пропонує широкий набір інструментів та функціональностей для розробки веб-додатків.

Основними особливостями Angular при розробці є:

- Компонентна архітектура: Angular розбиває додаток на невеликі незалежні компоненти, які забезпечують відокремлену логіку та розподілення обов'язків.

- Прив'язка даних: Angular пропонує потужну систему прив'язки даних, яка дозволяє автоматично синхронізувати дані між компонентами та шаблонами.
- Сервіси: Angular надає механізм сервісів для організації спільної логіки та обміну даними між компонентами.
- Маршрутизація: Angular має вбудований механізм маршрутизації, який дозволяє переходити між сторінками додатка та обробляти параметри Url.

Angular компоненти базується на мові програмування Typescript, мові розмітки HTML та мові стилей CSS.

Що ж таке Typescript? TypeScript – це мова програмування, яка є розширенням JavaScript (мова програмування, котра була створена задля виконання скріптів в браузері, хоча зараз вже використовується в усіх сферах). Вона додає до JavaScript додаткові функції і можливості, що полегшують розробку складних програмних систем.

TypeScript пропонує статичну типізацію, що дозволяє (й навіть по конвенції змушує) вказувати типи для змінних, параметрів функцій та повернутого значення функцій. Через статичну типізацію розробка веб-додатків стає ефективною в чистоті коду та часі, адже помилки, або якісь неточності коду видно відразу ж в IDE, а не на етапі запуску проєкту.

Після завершення розробки проєкту, перед його розгортанням виконується процес збірки (build). Під час цього процесу код, написаний на TypeScript, перетворюється на JavaScript, щоб його можна було використовувати в браузерах.

Для зображення даних не лише в фреймворці Angular, але й при будь-якому підході веб-розробки використовується HTML. HTML (HyperText Markup Language) є основною мовою розмітки, використовуваною для створення структури веб-сторінок. Вона дозволяє визначати елементи, такі як заголовки, параграфи, таблиці, списки та інші, а також використовувати теги для створення посилань, зображень, форм і багатьох інших елементів веб-сторінки.

В сукупності з CSS HTML дозволяє створювати інтерактивні та привабливі веб-сторінки. CSS (Cascading Style Sheets) – це мова таблиць стилів, що використовується для задання вигляду та форматування документа, написаного у HTML. Вона визначає зовнішній вигляд веб-сторінок, такі як розташування контейнерів та елементів, їх кольори, шрифти та анімацію.

За допомогою CSS можна визначити різні стилі і застосовувати їх до різних елементів веб-сторінки. CSS працює за допомогою виборів (селекторів) HTML-елементів за їх тегами, класами або ідентифікаторами та застосування до них конкретних стилів.

Однак використання чистого CSS не є найзручнішим та ефективнішим для розробки. Для цього існують пропрецесори CSS, які додають розширені можливості до звичайно CSS. В даному проєкті я використав Sass (Syntactically Awesome Style Sheets).

Використання Sass дозволяє зменшити дублювання коду, полегшити управління стилями та забезпечити більшу гнучкість при розробці веб-інтерфейсу.

Для розгортки всього проєкту було прийнято рішення використати Docker. Docker – платформа для контейнеризації додатків, що дозволяє пакувати програмне забезпечення та всі його залежності в контейнери.

Контейнеризація забезпечує швидку та незалежну встановлення додатків, зручне масштабування та управління ресурсами.

Docker дозволяє швидко розгортати додатки на різних середовищах без проблем з сумісністю та залежностями в порівнянні зі звичайним встановленням проєктів.

Для використання Docker’у потрібно написати Dockerfile - файл, в котрому описані інструкції щодо автоматичного створення Docker образу. Він визначає середовище та налаштування, необхідні для збірки та запуску програми.

Також для зручності Docker контейнери можна згрупувати використовуючи Docker Compose – інструмент для керуванням

багатоконтейнерними додатками. Він дозволяє описати конфігурацію додатків у YAML-файлі, вказуючи налаштування для кожного контейнеру, описаному в звичайному Dockerfile.

2.4. Опис структури системи та алгоритмів її функціонування

В створеному веб-застосунку для організаціх коду використовується тришарова архітектура (Three-Tier Architecture) – підхід до організації програмного забезпечення, в основі якого лежить розподіл функціональності на три логічні шари або рівні. Кожен рівень має свою відповідальність та функціональність, і вони взаємодіють між собою для забезпечення повноцінної роботи додатка.

Основними компонентами тришарової архітектури є:

1. Presentation Layer (шар представлення): верхній рівень архітектури, який взаємодіє з користувачем або зовнішніми системами. Його головна відповідальність - представлення даних та інтерфейсу користувача. В цьому рівні розташований веб-інтерфейс, а конкректно його реалізація на фреймворці Angular.
2. Business Logic Layer (шар бізнес-логіки): цей рівень містить бізнес-логіку або бізнес-правила додатка. Він відповідає за обробку даних, валідацію, обчислення та інші бізнес-специфічні операції. Тут зосереджені класи та компоненти, які реалізують функціональність додатка.
3. Data Access Layer (шар доступу до даних): цей рівень забезпечує доступ до даних, збереження, отримання, оновлення та видалення даних з різних джерел, таких як бази даних або зовнішні сервіси.

Розглянемо структуру бекенд проєкту та реалізацію троьхшарової архітектури (рис. 2.2.).

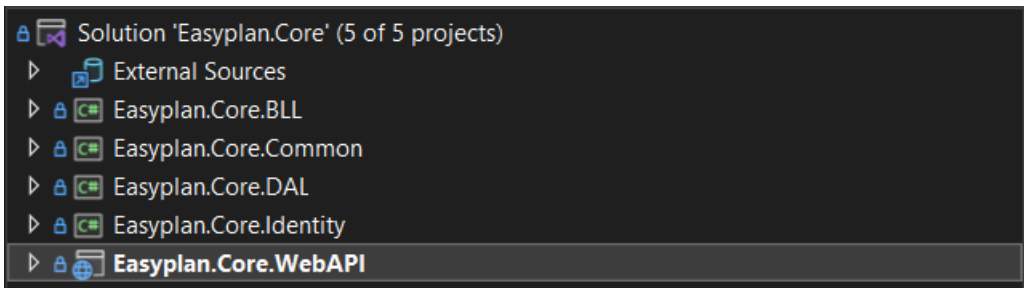


Рис. 2.2. Загальна структура бекенд проєкту

На даному рисунку гарно видно, що я створив чотири бібліотеки класів та один веб-API застосунок, серед яких:

- Easyplan.Core.BLL: шар бізнес логіки, в якому створені класи, котрі будують основу додатка, а саме: винятки, методи розширення, хелпери, інтерфейси, профілі для мапінгу створені для використання Automapper'ом та сервіси (рис. 2.3.).

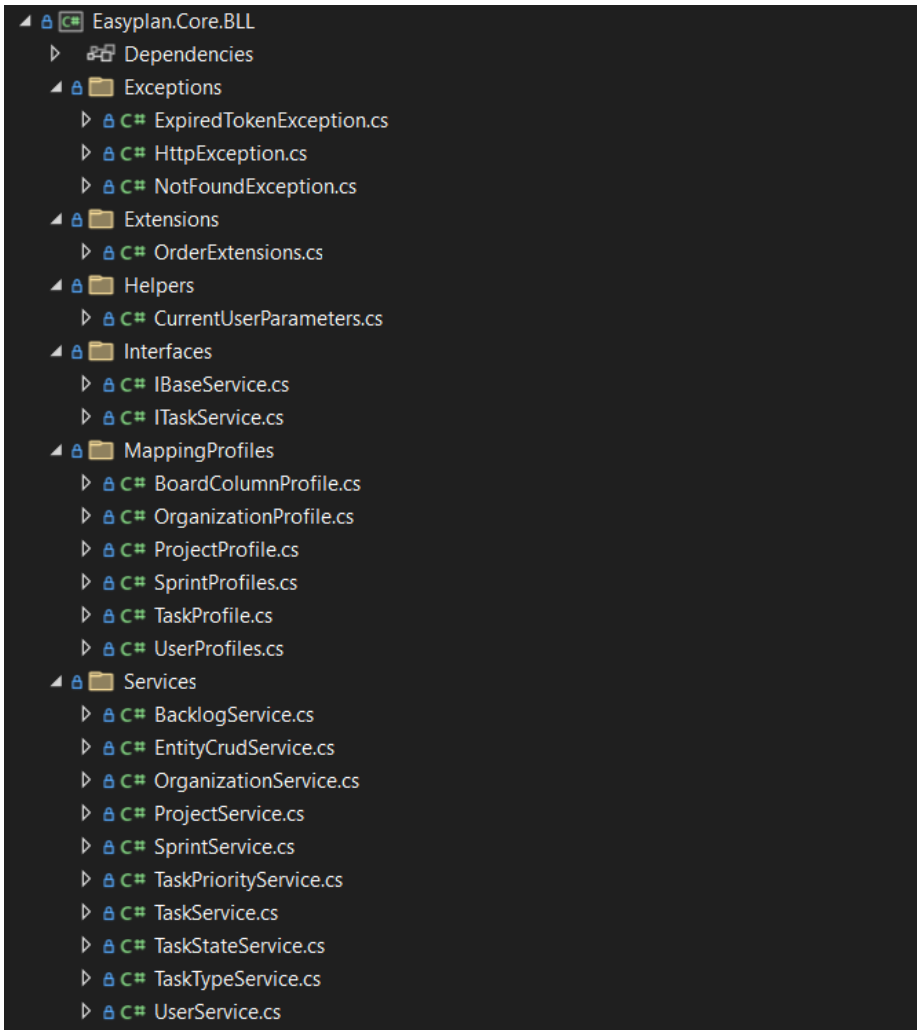


Рис. 2.3. Структура BLL - шару бізнес логіки

- Easyplan.Core.Common: бібліотека класів, котра має на меті зберігати всі моделі, котрі використовуються на серверній частині проєкту. Ці моделі поділяються на ті, що використовуються для оперування даними з БД - сутності (Entities) та на DTO (Data Transfer Object) - моделі, котрі використовуються для передачі даних від серверної до клієнтських частин та навпаки. (рис. 2.4.)

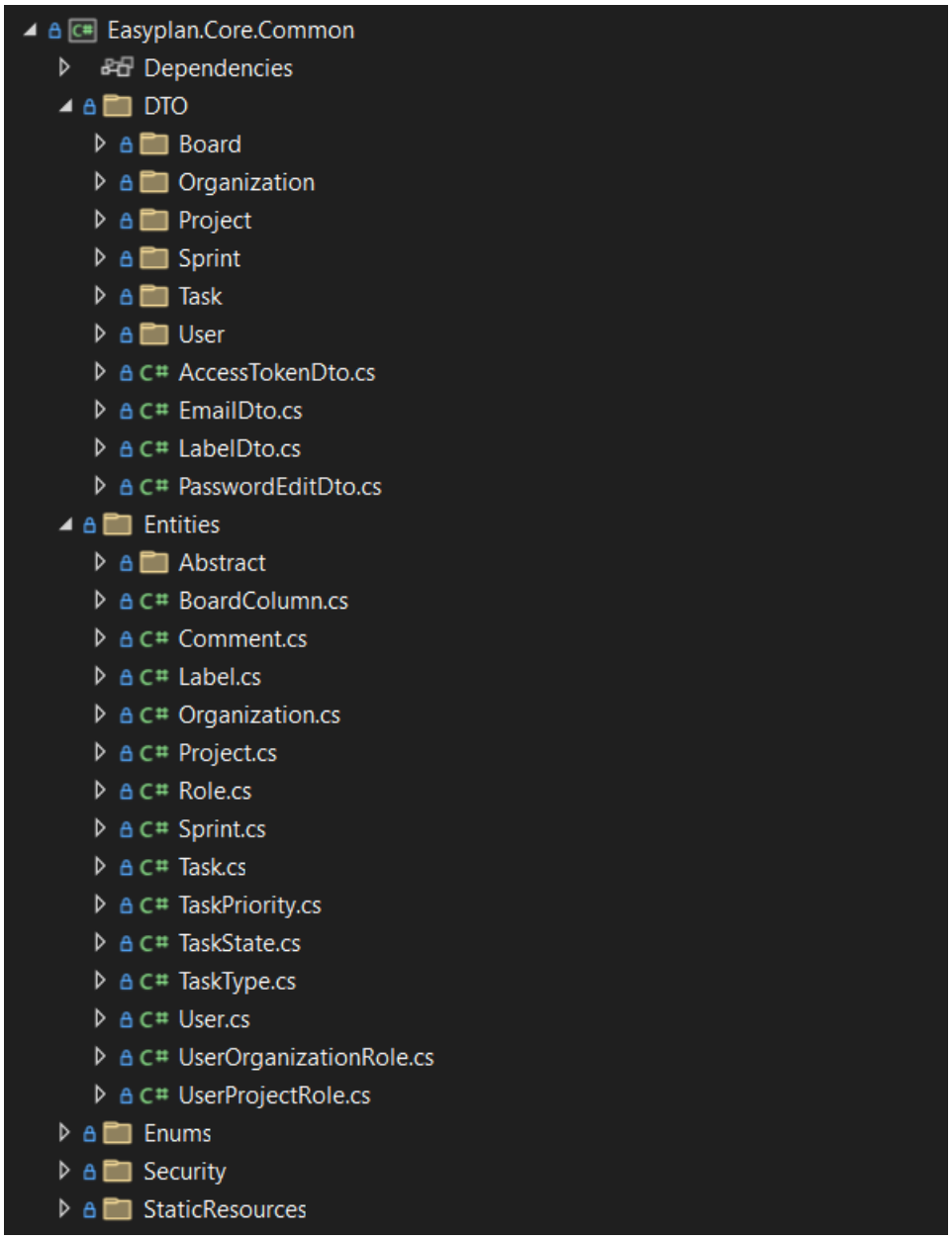


Рис. 2.4. Структура бібліотеки класів Common

- Easyplan.Core.DAL: шар доступу до даних, в якому зберігається схема бази даних, описана кодом, яка розташована в файлі EasyplanContext. Також це є місцем для збереження налаштувань сутностей БД, її міграцій (файлів, котрі

зберігають процес зміни структури БД з метою забезпечення сумісності з новими версіями ПЗ) та сідингу даних (заповнення таблиць БД певними рядками, під час першого створення бази даних). (рис. 2.5.)

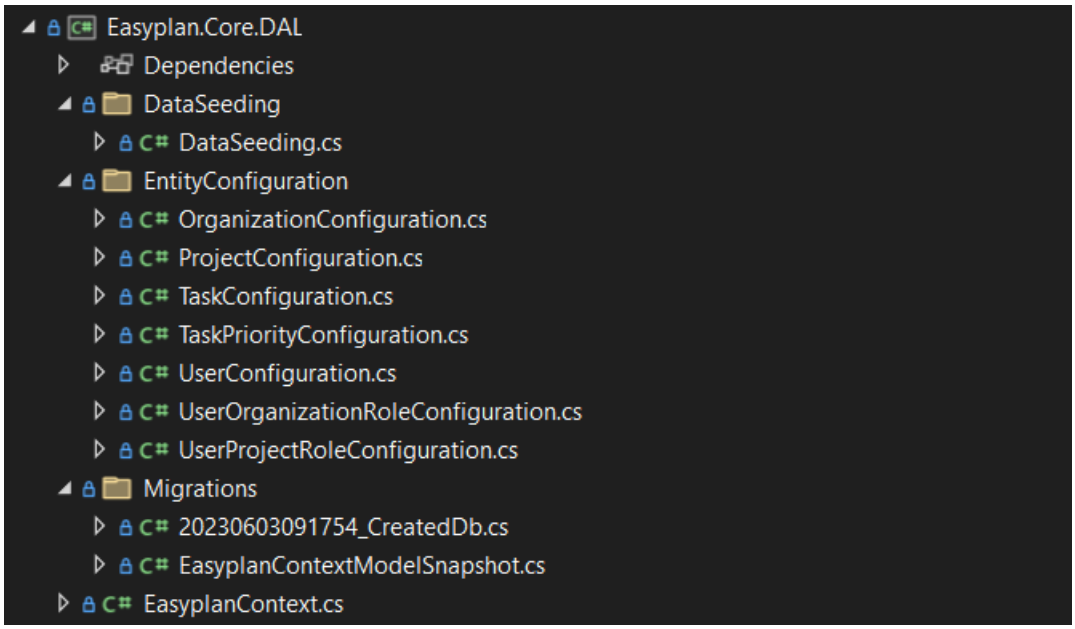


Рис. 2.5. Структура DAL - шару доступу до даних

- Easyplan.Core.Identity: бібліотека класів, котра має на меті в собі реалізувати JWT автетифікацію. (рис. 2.6.)

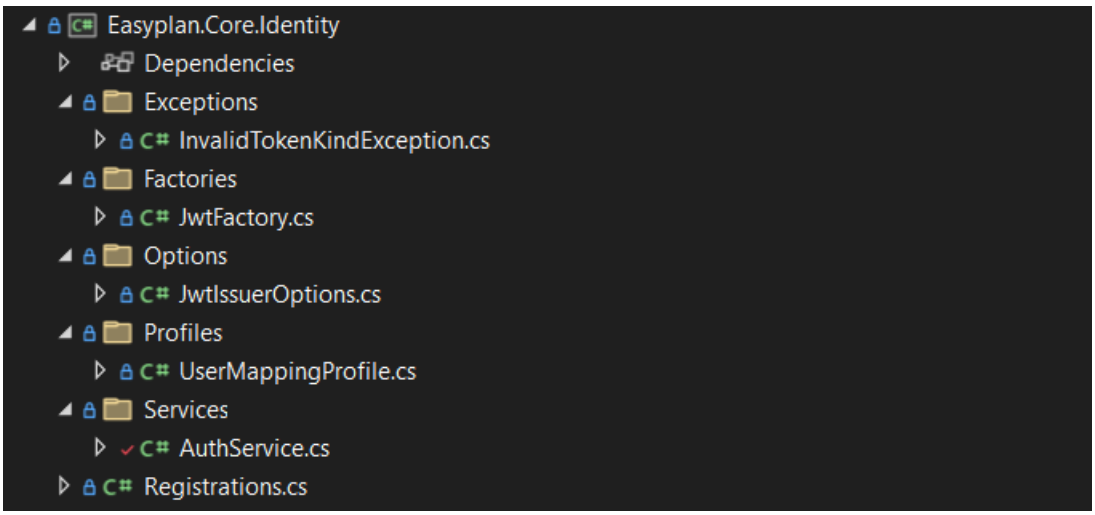


Рис. 2.6. Структура бібліотеки класів Identity

- Easyplan.Core.WebAPI: є веб-сервісом, розробленим з використанням фреймворку ASP.NET. Його основною метою є управління цілим серверним застосунком та наданням API-точок доступу, які відповідають за обробку різних типів HTTP-запитів, таких як POST, PUT, DELETE та GET. Ці точки доступу визначені у контролерах, які містять відповідні дії для обробки

вхідних запитів та генерації відповідей та використовують відповідні до контролерів сервіси, створені в BLL. (рис. 2.7.)

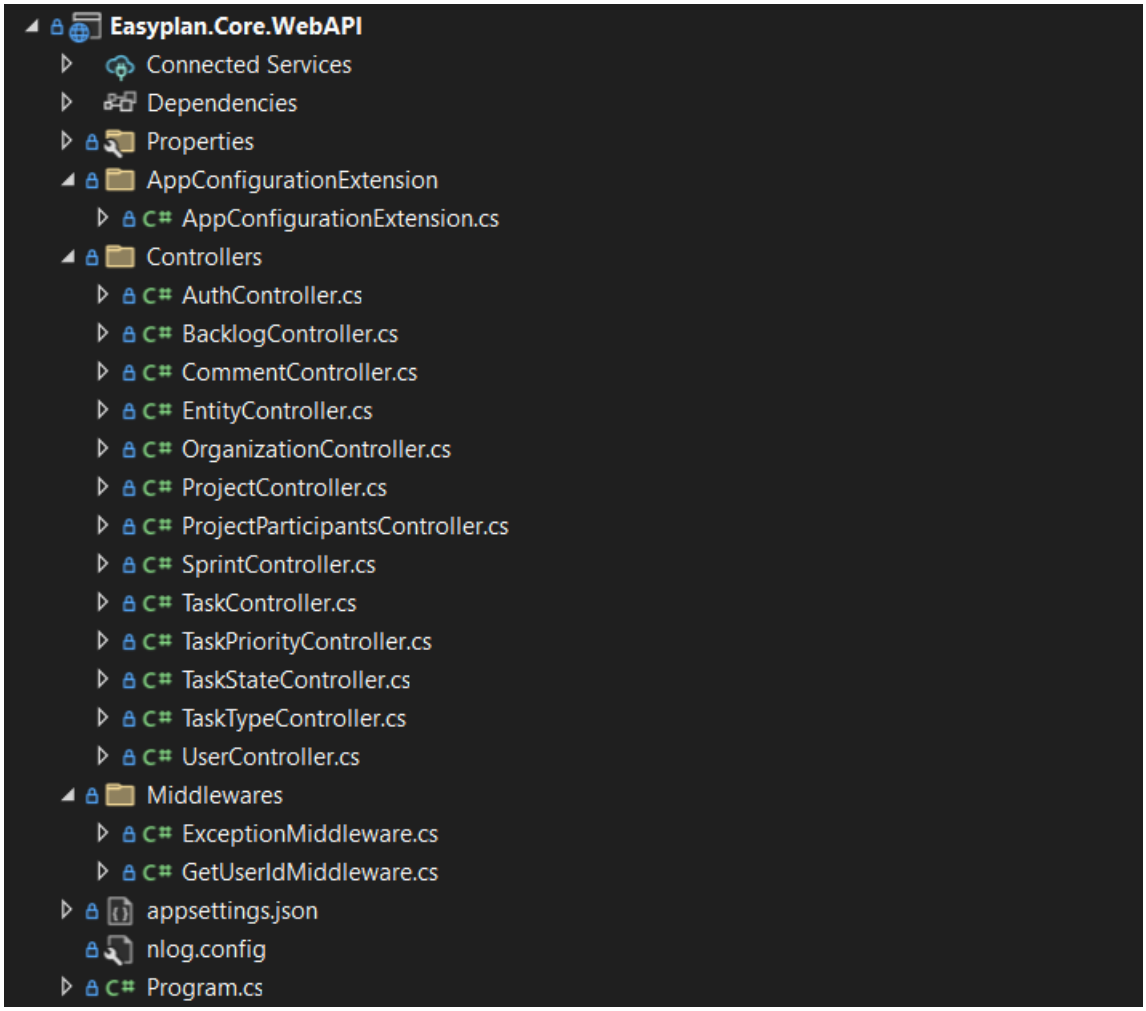


Рис. 2.7. Структура ASP.NET проєкту WebAPI

Взагалі, WebAPI є основним проєктом, що керує всім додатком. Запуск додатка відбувається у файлі Program.cs, де встановлюються основні конфігурації, такі як вибір IP-адреси, на якій буде розміщений сервер, вибір бази даних, автоматичний запуск міграцій, налаштування DI-контейнера та реєстрація сервісів зі шару BLL.

Також, як я й казав в попередньому підрозділі, ASP.NET має middleware. В моєму проєкті я створив два middleware:

ExceptionMiddleware, який перехоплює будь-які помилки, що виникли під час HTTP-запиту, та повертає форматований результат про помилку клієнту для правильної обробки.

GetUserIdMiddleware, який отримує ідентифікатор користувача з JWT-токену і передає його іншим сервісам для подальшої обробки.

База даних була спроєктована за допомогою Entity Framework з використанням підходу Code First. Це означає, що я власноруч визначив таблиці та зв'язки між ними за допомогою мови програмування C#. При створенні міграції "CreatedDb" ORM автоматично згенерував ці зв'язки. Нижче наведено діаграму БД, згенеровану утилітою "PgAdmin"(рис. 2.8).

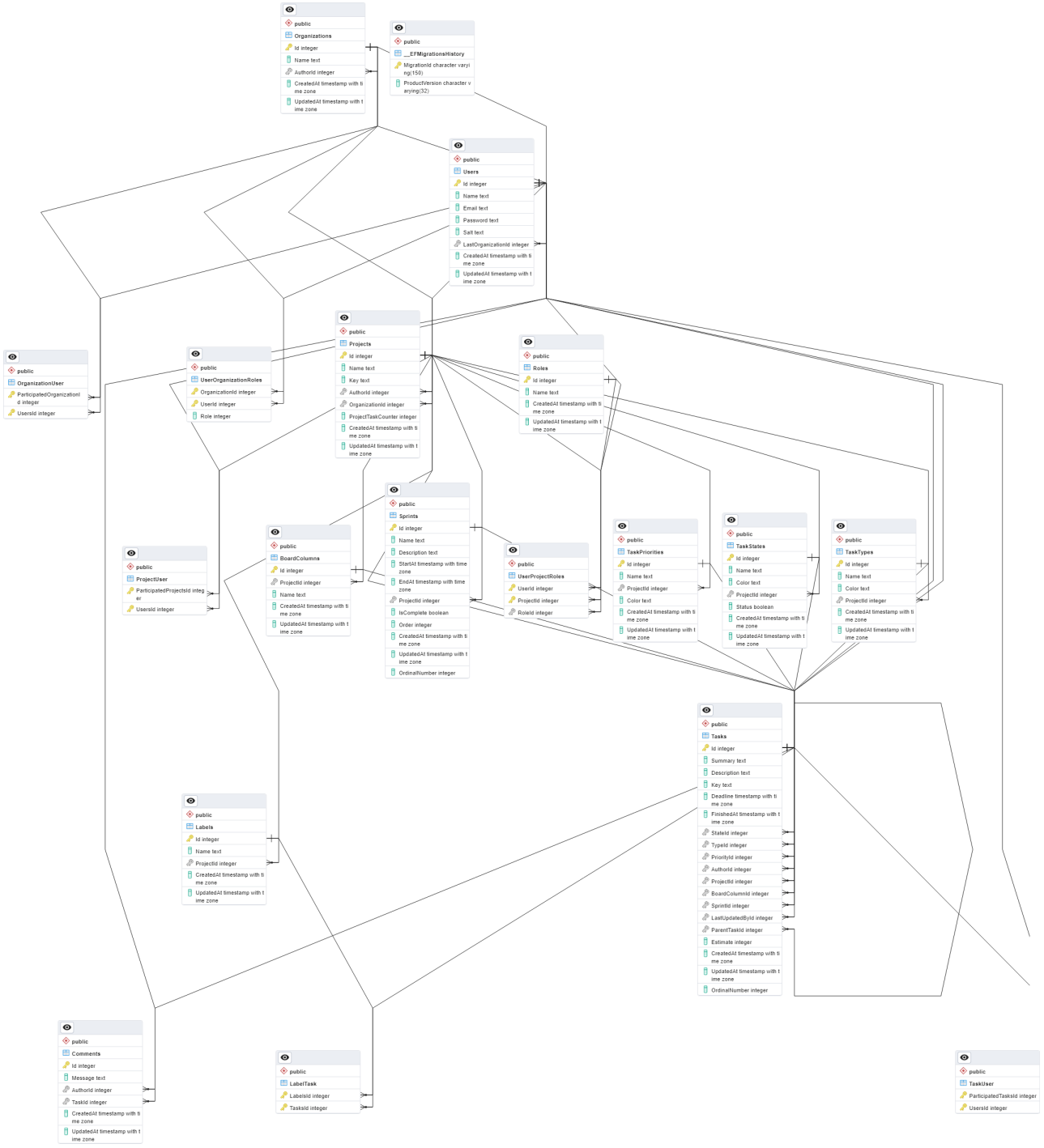


Рис. 2.8. ER діаграма бази даних

У проєкті головною таблицею є "Задачі" (Tasks). Вона має стандартні колонки, такі як Id, дата створення, дата оновлення, а також зовнішній ключ до користувача, який створив задачу, проєкта, в якому вона була створена, спрінта і колонки на Kanban дошці.

Іншою важливою таблицею є "Користувачі" (Users). Вона також містить стандартні колонки, такі як Id, дата створення, дата оновлення, а також основні колонки, які включають ім'я, електронну пошту, захешований пароль і сіль до нього. Крім того, вона містить зовнішні ключі до інших таблиць.

Такі таблиці, як "Проєкти" і "Організації", також тісно пов'язані і мають схожі колонки.

Більш зрозумілішу схему БД, але без зв’язків можна поглянути на рисунку нижче (рис. 2.9).

```
public class EasyplanContext : DbContext
{
    0 references
    public EasyplanContext(DbContextOptions<EasyplanContext> options) : base(options) { }
    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfigurationsFromAssembly(typeof(ProjectConfiguration).Assembly);
        modelBuilder.Seed();
    }

    9 references
    public DbSet<Organization> Organizations { get; set; }
    23 references
    public DbSet<User> Users { get; set; }
    21 references
    public DbSet<Project> Projects { get; set; }
    17 references
    public DbSet<Task> Tasks { get; set; }
    4 references
    public DbSet<Comment> Comments { get; set; }
    19 references
    public DbSet<Sprint> Sprints { get; set; }
    3 references
    public DbSet<BoardColumn> BoardColumns { get; set; }
    0 references
    public DbSet<Label> Labels { get; set; }
    0 references
    public DbSet<Role> Roles { get; set; }
    4 references
    public DbSet<TaskPriority> TaskPriorities { get; set; }
    8 references
    public DbSet<TaskState> TaskStates { get; set; }
    6 references
    public DbSet<TaskType> TaskTypes { get; set; }
    7 references
    public DbSet<UserProjectRole> UserProjectRoles { get; set; }
    5 references
    public DbSet<UserOrganizationRole> UserOrganizationRoles { get; set; }
}
```

Рис. 2.9. Методи для роботи з таблицями БД

Іншим важливим аспектом даної роботи є JWT автентифікація. Її розроблено в проєкті Easyplan.Core.Identity. JWT (JSON Web Token) – це

стандарт для створення токенів, що використовуються для автентифікації та авторизації в розподілених системах. Аутентифікація за допомогою JWT полягає у використанні токенів, які містять цифровий підпис інформації про користувача, що підтверджує його ідентичність.

JWT складається з трьох основних частин: заголовку (header), тіла (payload) і підпису (signature). Заголовок містить тип токена та алгоритм шифрування. Тіло містить дані про користувача або іншу інформацію, яку потрібно передати. Підпис генерується з використанням секретного ключа, який забезпечує цілісність та автентичність токена.

При використанні JWT для автентифікації, сервер видаватиме токен користувачеві після успішної авторизації або реєстрації. Спочатку користувач повинен бути успішно зареєстрованим або авторизованим, потім з авторизаційного сервісу викликається метод GetAccessToken, який використовуючи Id, ім’я та електронну пошту користувача, створює токен. (рис. 2.10)

```
namespace Easyplan.Core.WebAPI.Controllers
{
    [Route(template: "api/user")]
    [AllowAnonymous]
    [ApiController]
    1 reference
    public class AuthController : ControllerBase
    {
        private readonly AuthService _service;

        0 references
        public AuthController(AuthService service)
        {
            _service = service;
        }

        [HttpPost(template: "login")]
        0 references
        public async Task<IActionResult> Login([FromBody] UserLoginDto loginInfo)
        {
            var user = await _service.Login(loginInfo);
            var token = _service.GetAccessToken(user.Id, user.Name, user.Email);
            return Ok(token);
        }

        [HttpPost(template: "register")]
        0 references
        public async Task<IActionResult> Register([FromBody] UserRegisterDto registerInfo)
        {
            var user = await _service.Register(registerInfo);
            var token = _service.GetAccessToken(user.Id, user.Name, user.Email);
            return Ok(token);
        }
    }
}
```

Рис. 2.10. AuthController і його ендпоінти

Ось метод, котрий генерує токен, використовуючи фабрику JWT. (рис. 2.11)

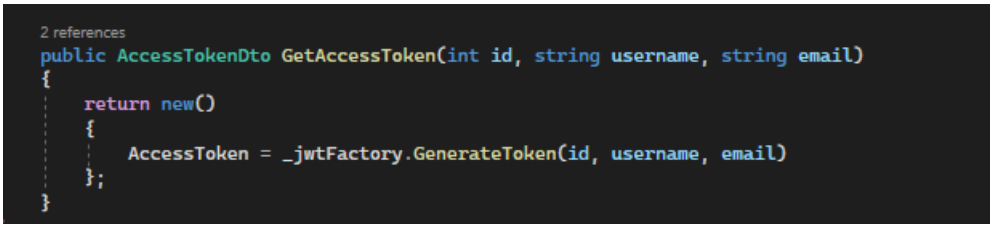


Рис. 2.11. Метод GetAccessToken

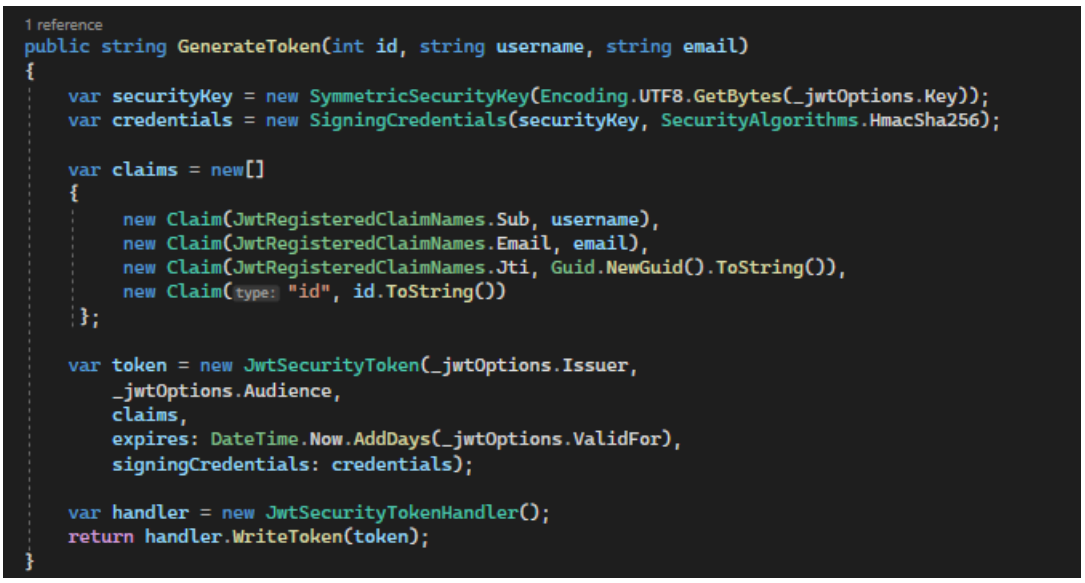
Далі я поясню сам метод GenerateToken (рис. 2.12). Спочатку створюється об'єкт SymmetricSecurityKey, який використовується для підпису токену. Для цього використовується ключ (Key) з JwtIssuerOptions (котрі налаштовуються в appsettings.json), який передається у вигляді байтів у форматі UTF-8.

Далі створюються об'єкт SigningCredentials з використанням SymmetricSecurityKey, використовуючи алгоритм HmacSha256 для підпису токену.

В масиві claims вказуються дані, які будуть включені в тіло токену, такі як ім'я користувача, електронна пошта та унікальний ідентифікатор. Також додається власний клейм "id" з переданим значенням id.

Далі створюється об'єкт JwtSecurityToken, який представляє сам токен. Він приймає параметри видавця (Issuer), аудиторії (Audience), масиву claims, дату завершення дії (expires) та об'єкт SigningCredentials для підпису.

Завершується метод генерацією токену, використовуючи об’єкт JwtSecurityTokenHandler з бібліотеки System.IdentityModel.Tokens.Jwt.



```
1 reference
public string GenerateToken(int id, string username, string email)
{
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtOptions.Key));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, username),
        new Claim(JwtRegisteredClaimNames.Email, email),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new Claim(type: "id", id.ToString())
    };

    var token = new JwtSecurityToken(_jwtOptions.Issuer,
        _jwtOptions.Audience,
        claims,
        expires: DateTime.Now.AddDays(_jwtOptions.ValidFor),
        signingCredentials: credentials);

    var handler = new JwtSecurityTokenHandler();
    return handler.WriteToken(token);
}
```

Рис. 2.12. Метод створення JWT токену GenerateToken

Потім, при отриманні та збереженні токену, користувач повинен буде включати цей токен у заголовку або запиті кожного наступного запиту до сервера (рис. 2.13.). Сервер перевірятиме цей токен (використовуючи middleware), розшифровуючи його і перевіряючи підпис, тим самим підтверджуючи ідентичність користувача та його права доступу.

```
You, 10 hours ago | 1 author (You)
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  public intercept(req: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    const accessToken = localStorage.getItem('token');
    if (accessToken) {

      req = req.clone({ setHeaders: { Authorization: `Bearer ${accessToken}` } });
    }

    return next.handle(req);
  }
}
```

Рис. 2.13. Реалізація Angular інтерсептора, котри додає збережений в локальному сховищі токен до заголовку запиту

Шаром представлення в даному проєкті є фреймворк Angular. Давайте розглянемо архітектуру фронтенд частини додатку.

Angular має модульну та компонентну архітектуру. Тож основними модулями в цій роботі є:

- app.module.ts: Головний модуль, який імпортує всі використовані модулі та компоненти в додатку.

- app-routing.module.ts: Модуль, в якому налаштовується основна маршрутизація. Цей модуль імпортується лише один раз в app.module.ts.
- shared.module.ts: Модуль, в якому створюються компоненти, які можуть бути використані у будь-якій частині додатку. Цей модуль може бути імпортований багато разів.
- auth.module.ts: Модуль, в якому реалізована основна логіка для автентифікації, авторизації та реєстрації. Цей модуль імпортується лише один раз в app.module.ts.
- core.module.ts: Модуль, який представляє ядро функціональності додатку. Тут створюються всі сервіси, які використовуються в компонентах, всі моделі, які отримуються з сервера та використовуються в додатку, налаштування, інтерсептор JWT авторизації та глобальний обробник помилок. Модуль імпортується лише один раз.

Компоненти модулів. З основних компонентів можна виділити ті, які створені в Shared модулі та використовуються на багатьох сторінках (рис. 2.14.).

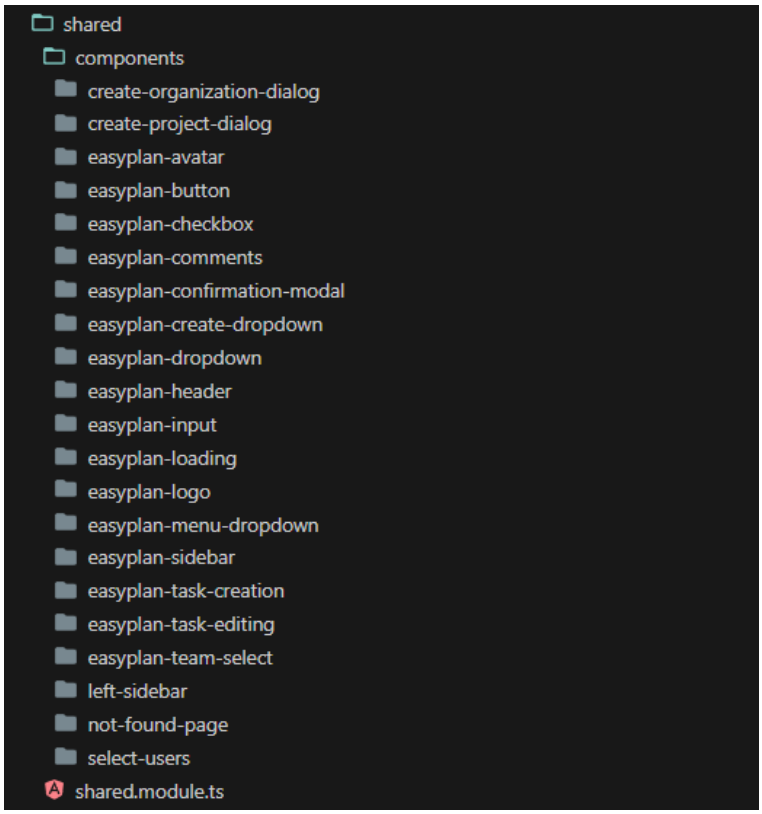


Рис. 2.14. Компоненти модуля Shared

З часто-використовуваних компонентів можна виділити:

- not-found-page: сторінка, на котру маршрутизатор перенаправляє, якщо вказана неправильна сторінка в Url.
- easyplan-header: компонент-верхня частина сторінки, де присутні навігаційні компоненти.
- easyplan-avatar: компонент аватару, котрий генерує аватар для користувачів, використовуючи перші літери їхніх імен.
- easyplan-sidebar: компонент, ліва частина сторінки, на якому є навігаційні елементи для маршрутизацією в межах одного проєкту.
- easyplan-button: звичайна кастомізована кнопка.
- easyplan-logo: лого, зображення сайту.
- easyplan-menu-dropdown: дропдаун елемент для меню, використовується для вибору організації в хедері та для вибору проєкту.
- easyplan-dropdown: дропдаун для більш складних задач, використовується для вибору даних при створенні та редагуванні задачі.
- select-users: компонент для вибору або видалення користувачів з організації, проєкту, задачі.
- create-organization-dialog, create-project-dialog: діалогові вікна для створення організації та проєкту, використовуються easyplan-sidebar.
- easyplan-sidebar: діалогове вікно, котре випливає з правої сторони й займає половину екрану.
- easyplan-input: поле для вводу тексту, стилізоване під дизайн сайту.
- easyplan-task-creation, easyplan-task-editing: діалогові вікна для створення та редагування задач.

Також важливими є компоненти, збережені прямо в модулі App, більшість з яких є сторінками (рис. 2.15.).

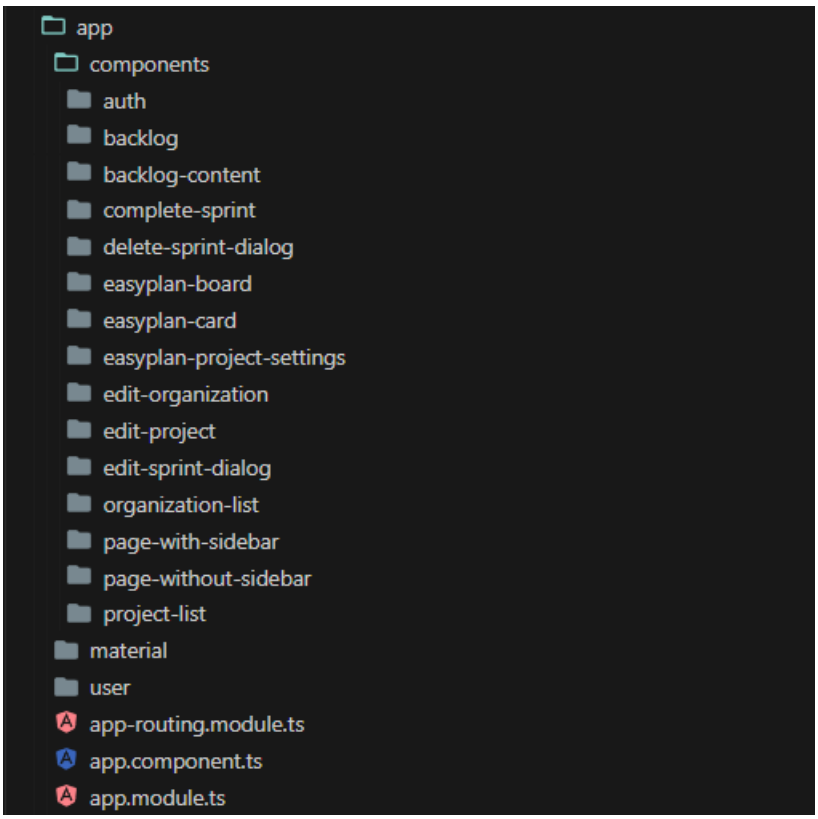


Рис. 2.15. Компоненти модуля App

Список сторінок сайту та компонентів, котрі використані в них:

- organization-list: список доступних організацій користувача, є можливість створення організації, редагувати ім’я організації, додавати та видаляти учасників.
- project-list: список доступних проєктів, є можливість створювати проєкти, редагувати їх, додавати та видаляти учасників.
- easyplan-board: сторінка, на якій розташована Kanban дошка запущеного в обраному проєкті спрінта. Використовує компоненти-картки easyplan-card для відображення та оперування задачами. Використовує CdkDragDrop з бібліотеки Angular Material для перетягування карток між колонками. Після закінчення робочого процесу, користувач може завершити спрінт, натиснувши відповідну кнопку.
- backlog: сторінка на якій створюються та заповнюються спрінти. Також використовує CdkDragDrop для перетягування тасок в спрінт. Є можливість переглядати історію вже завершених спрінтів.

- easyplan-project-settings: сторінка налаштувань проєкту. Є можливість додавати, редагувати та видаляти типи, пріоритети та стани задач.

Не слід й забувати про важливість автетифікації. Компоненти модуля Auth є сторінками, які використовуються для валідації вхідних даних користувачів (рис. 2.16.).

Також в застосунку використовуються гарди (guards), котрі забороняють користувачам без доступу (або неавторизованим користувачам) переходити на відповідні сторінки.

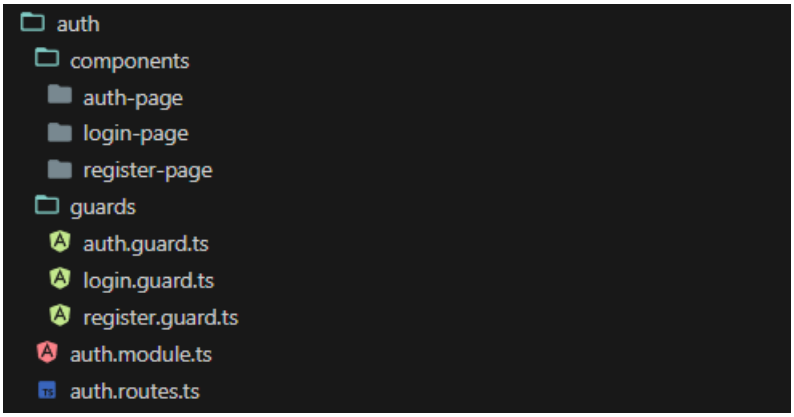


Рис. 2.16. Компоненти модуля Auth

Тепер я розповім про розгортання сайту-системи. Для цього я написав два Dockerfile: один для серверної частини (рис. 2.17.), інший - для клієнтської частини (рис. 2.18.).

```
1  FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-environment
2  WORKDIR /app
3
4  COPY . ./
5  RUN dotnet restore
6  RUN dotnet publish -c Release -o out
7
8  FROM mcr.microsoft.com/dotnet/aspnet:6.0
9  WORKDIR /app
10 COPY --from=build-environment /app/out .
11 ENTRYPOINT ["dotnet", "Easyplan.Core.WebAPI.dll"]
```

Рис. 2.17. Dockerfile бекенду

```
FROM node:18-alpine3.15 AS frontend-build
WORKDIR /usr/src/app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:1.23.1-alpine
COPY nginx.conf /etc/nginx/nginx.conf
COPY --from=frontend-build /usr/src/app/dist/easyplan /usr/share/nginx/html
```

Рис. 2.18. Dockerfile фронтенду

Кожен Dockerfile має подібні кроки. Спочатку визначається базовий образ середовища, в якому буде виконуватися білд додатку. Далі копіюються вихідні файли, необхідні для білду. Потім відбувається білд проекту, і після цього файли копіюються на Docker образ, на якому буде розміщено даний продукт.

Крім того, я створив docker-compose файл, який об'єднує ці контейнери в єдину систему. До складу цього файлу входить також контейнер для PostgreSQL БД та PgAdmin для управління цією СУБД. З використанням цього файлу, систему можна легко розгорнути на будь-якому сервері з встановленою Docker-платформою. Але перед цим потрібно буде налаштувати .env файли згідно до вимог та потреб користувача.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Користувач заповнює вхідні дані веб-додатку, використовуючи інтерфейс, який надається застосунком. Це може включати введення тексту, вибір параметрів, тощо.

Вихідні дані системи організовані у вигляді веб-сторінок, які користувач може переглядати та взаємодіяти з ними за допомогою браузера. Сторінки можуть містити текстову, графічну та іншу інформацію.

2.6. Опис розробленої системи.

2.6.1. Використані технічні засоби.

Щоб забезпечити стабільне функціонування додатку потрібна обчислювальна машина, на якій буде хоститися сервер. Вона повинна мати такі рекомендовані технічні характеристики:

- Процесор: 64-бітний процесор.
- Не менш ніж 4Гб оперативної пам’яті.
- Мінімум 20Гб вільного місця на жорсткому диску.
- Стабільне підключення до мережі інтернет.
- Підтримка апаратної віртуалізації (Intel VT-x або AMD-v), при деплої застосунку, використовуючи Docker.

Клієнтові потрібен лише доступ до мережі інтернет та браузер з підтримкою Javascript.

2.6.2. Використані програмні засоби.

- Потрібно переконатися, що використовувана операційна система (Linux, macOS, Windows) підтримує Docker для контейнеризації. В цьому випадку потрібно встановити лише Docker.
- Для безпосереднього деплою додатка на сервер без використання Docker, потрібно встановити .NET Framework і СУБД PostgreSQL.
- Якщо власноруч потрібно збудувати застосунок, то необхідно встановити: .NET SDK для розробки, Node.js для використання фреймворку Angular, npm пакети для Angular та БД PostgreSQL.

2.6.3. Виклик та завантаження програми.

Для завантаження програми потрібно мати встановлений Docker. Після налаштування змінних середовища і запуску команди “docker-compose up”, серверна та клієнтська частини будуть запущені.

Клієнтові, який бажає відвідати сайт, просто потрібно відкрити відповідну URL-адресу в браузері.

2.6.4. Опис інтерфейсу користувача.

Після того як користувач заходить на сайт його автоматично перенаправляє на сторінку авторизації (рис. 2.19.).

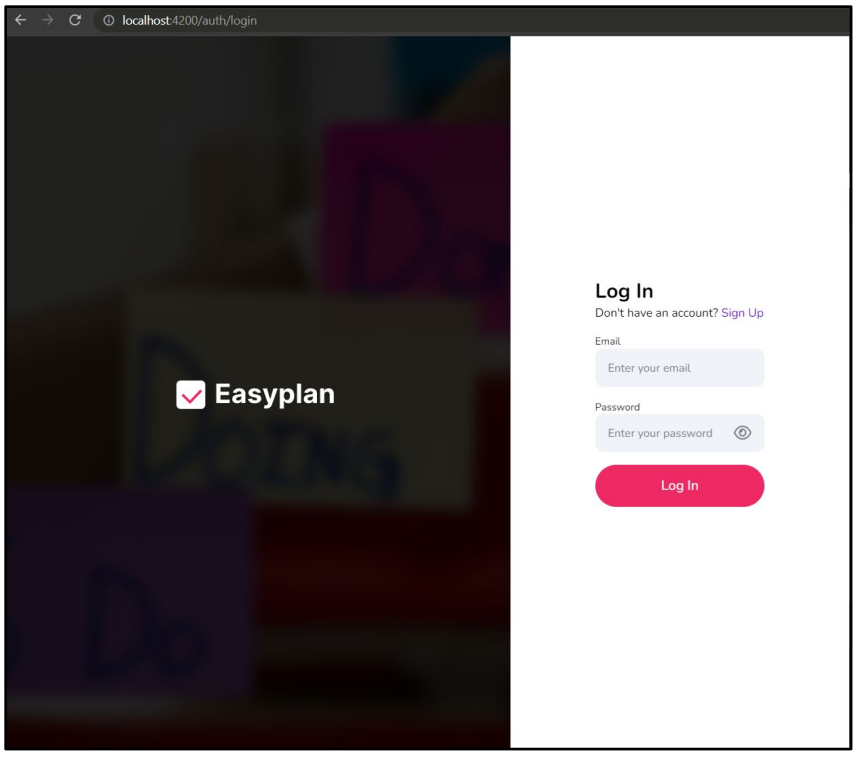


Рис. 2.19. Стартова сторінка сайту

Користувачу запропоновано або авторизуватися, або зареєструватися якщо акаунту немає. Натиснемо на кнопку реєстрації. Бачимо, що змінилась URL на register та з’явилися додаткові поля для вводу (рис. 2.19.).

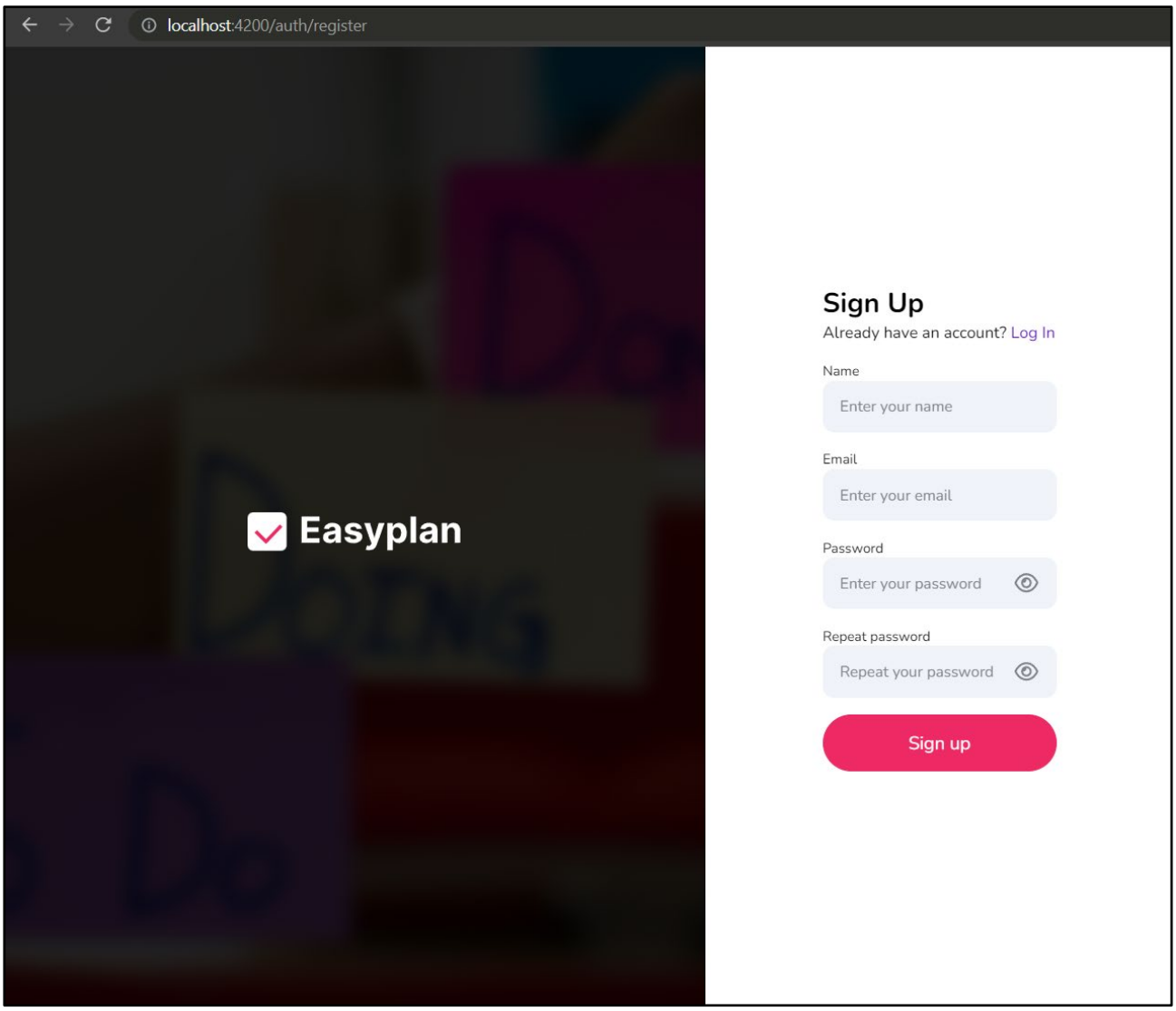


Рис. 2.20. Сторінка реєстрації

Заповнимо їх. Бачимо, що поля для вводу паролю відображаються у вигляді прихованих символів * (рис. 2.21.).

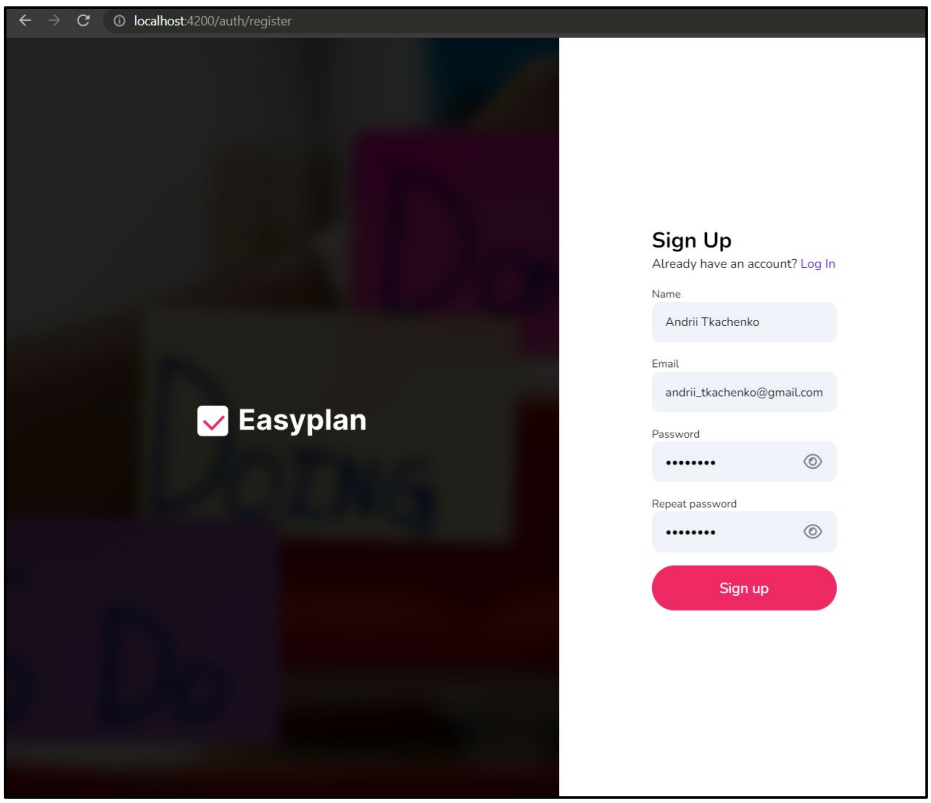


Рис. 2.21. Заповнені реєстраційні поля

Натиснемо кнопку Sign Up (зареєструватися). Бачимо, що ми були зареєстровані в системі та перенаправлені на Url “organizations” (рис. 2.22.).

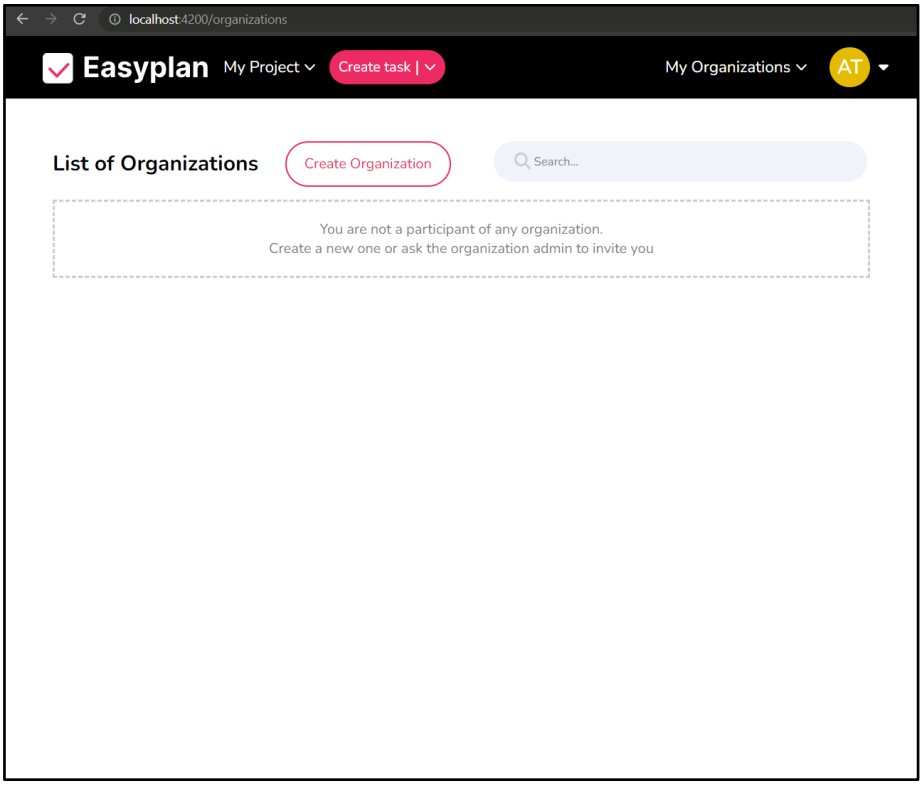


Рис. 2.22. Успішна реєстрація та сторінка “Список організацій”

Наразі ми не є частиною жодної організації, давайте її створимо. Для цього натиснемо кнопку “Create Organization”. Тепер ми бачимо модальне вікно для створення організації. Введемо назву нової організації (рис. 2.23.).

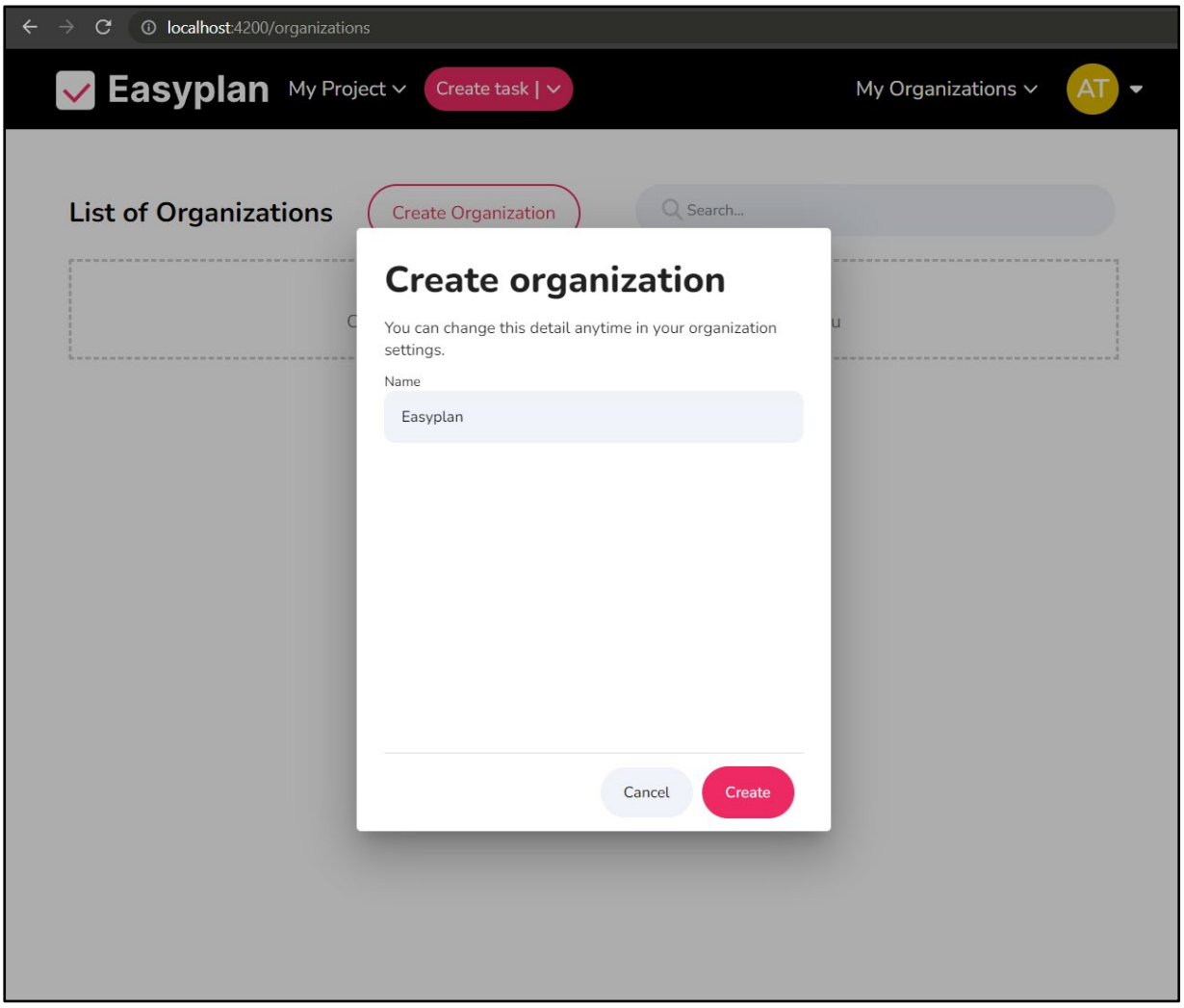


Рис. 2.23. Модальне вікно створення організації

Після натискання кнопки ми побачимо повідомлення, що організація була успішно створена. Крім того, нас автоматично перенаправлять на сторінку з проєктами (рис. 2.24.).

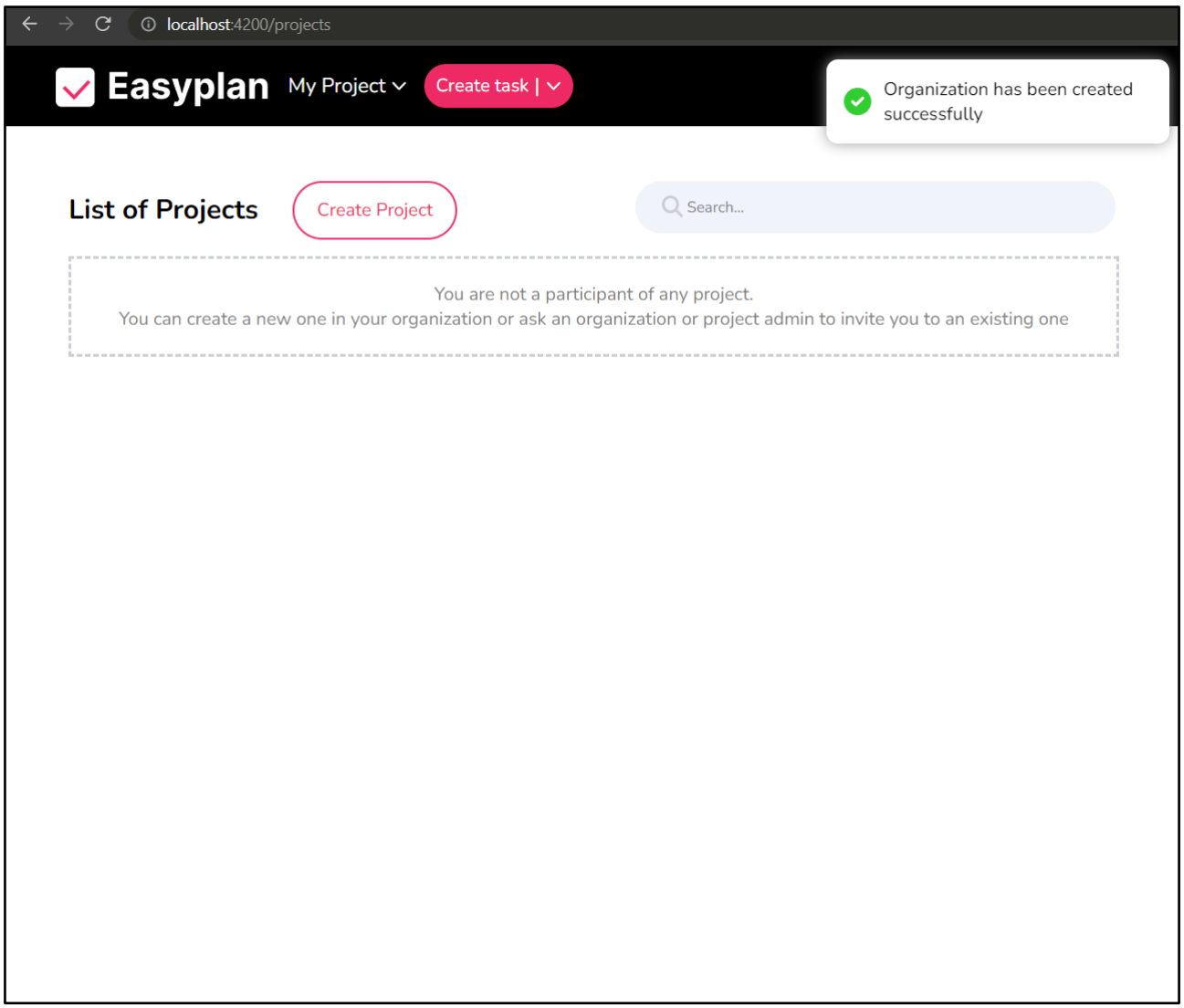


Рис. 2.24. Успішнє створення організації та сторінка “Список проєктів”

Бачимо, що у дропдауні “My Organizations” з’явилася наша новостворена організація (рис. 2.25.).

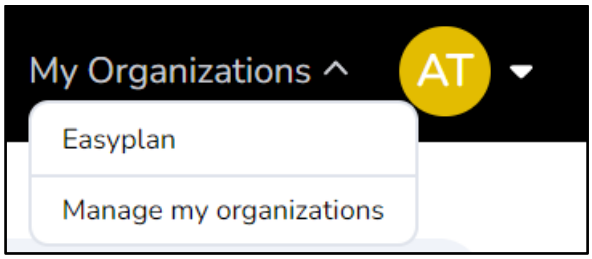


Рис. 2.25. Нова організація присутня в дропдауні

Зараз ми перейдемо до створення нашого першого проекту. Для цього натиснемо кнопку "Створити проект" і введемо ім'я проекту та його ключ. Цей

ключ буде використовуватися за замовчуванням для нових спрінтів у цьому проєкті. (рис. 2.26.)

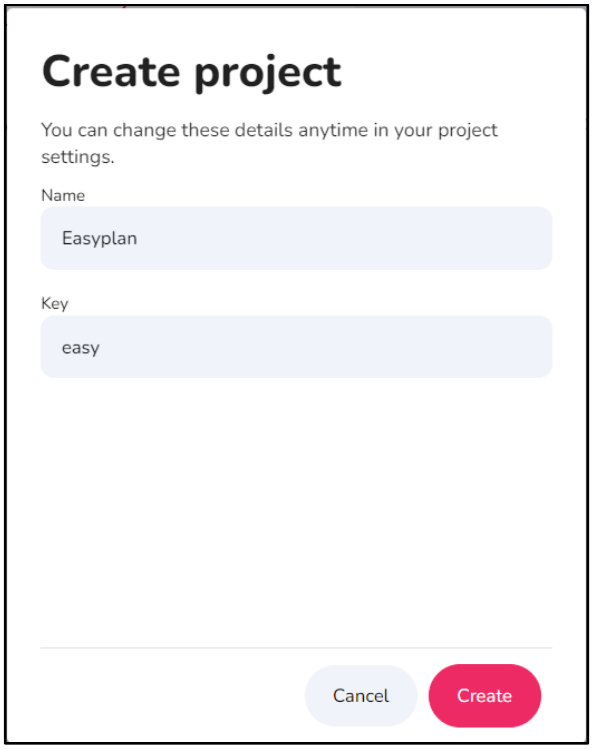


Рис. 2.26. Модальне вікно створення проєкту

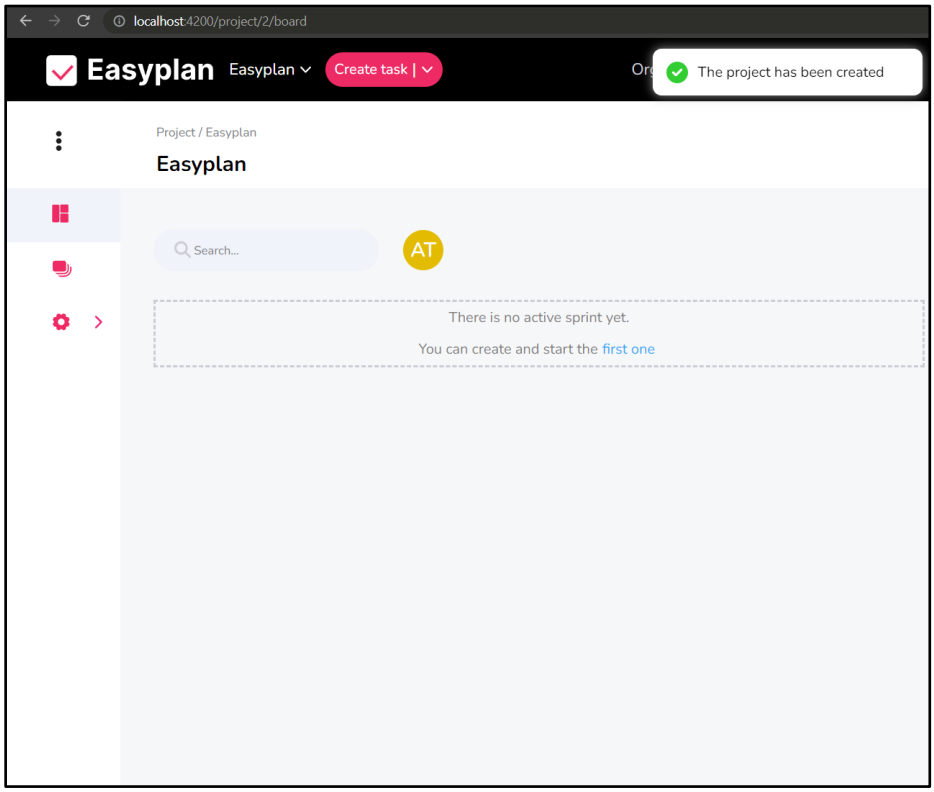


Рис. 2.27. Після успішного створення проєкту нас перенаправляє на пусту сторінку дошки

Після натискання на текст "first one" нас автоматично перенаправлять на сторінку беклогу. На цій сторінці ми матимемо можливість створити та спланувати спрінт, додавши до нього задачі. (рис. 2.28.)

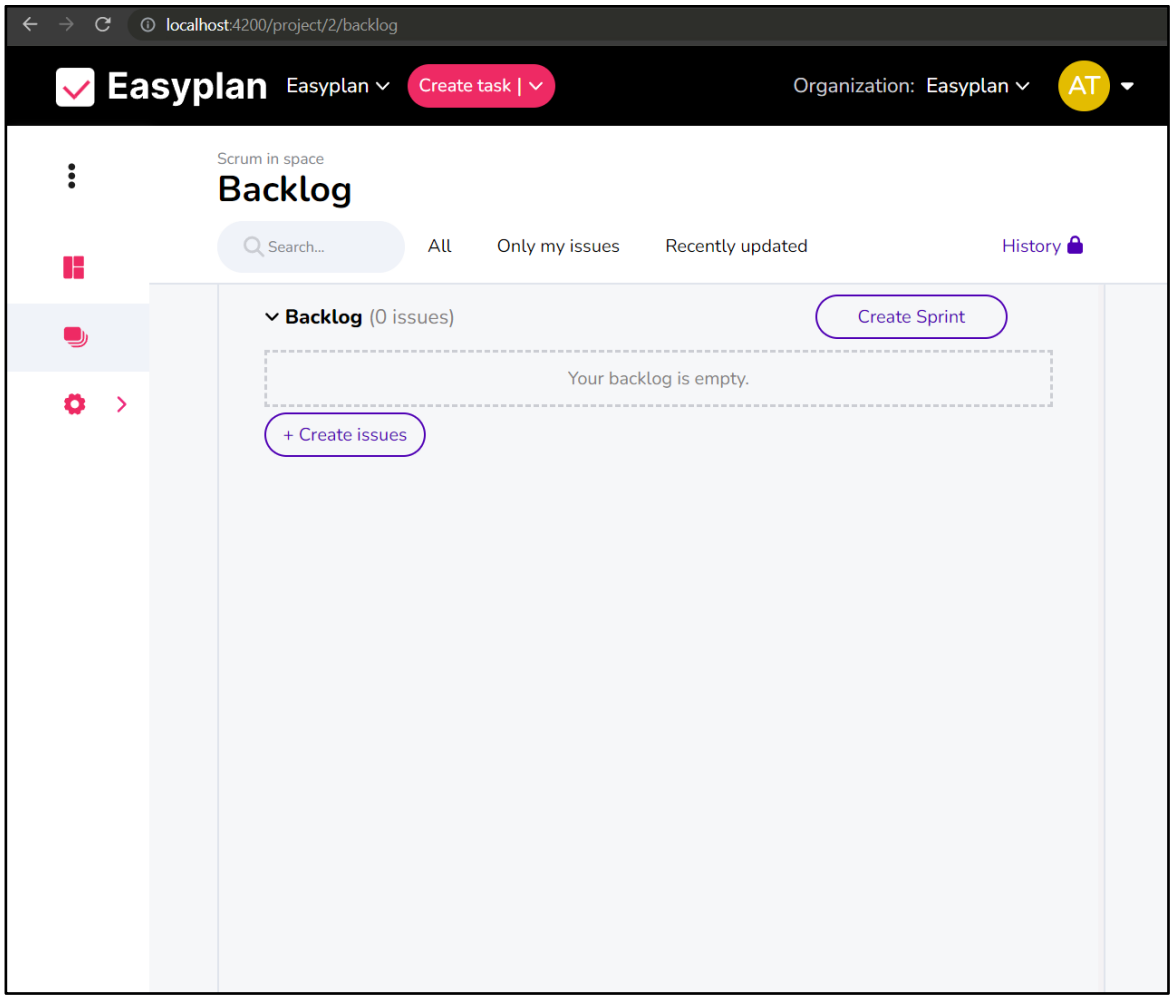


Рис. 2.28. Сторінка беклогу

Після натискання кнопки “Create Sprint” (“створити спрінт”) з’явився пустий спрінт, котрий не можна розпочати, поки в ньому немає жодної задачі (рис. 2.29.).

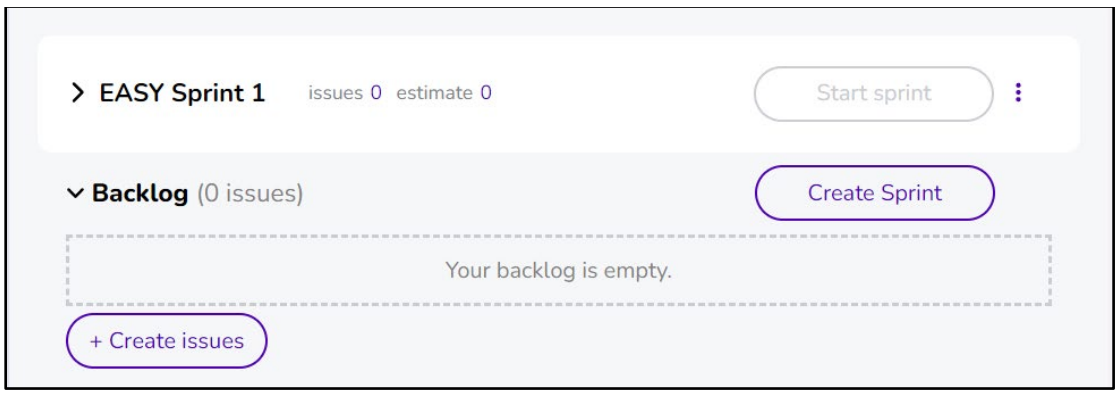


Рис. 2.29. Створено новий спрінт

Створимо першу задачу, натиснувши кнопку “Create issues”. Бачимо відкрите модальне вікно зі створення таски. Заповнимо його написаши назву задачі, її опис, тип, пріоритет та стан (рис. 2.30.).

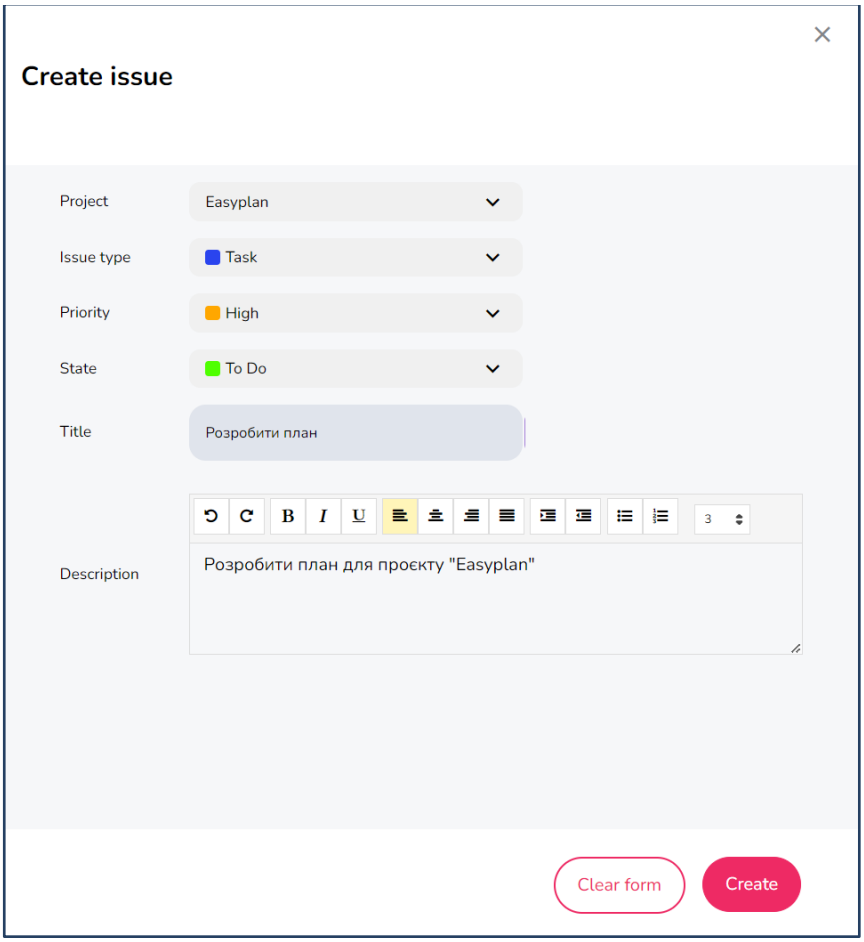


Рис. 2.30. Модальне вікно створення задачі

Після натискання на кнопку “Create” побачимо повідомлення про успішне створення (рис. 2.31.) та що задача була додана до спрінта (рис. 2.32.).

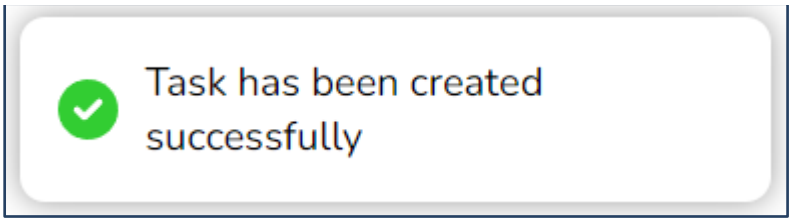


Рис. 2.31. Нотифікація, про успішне створення задачі

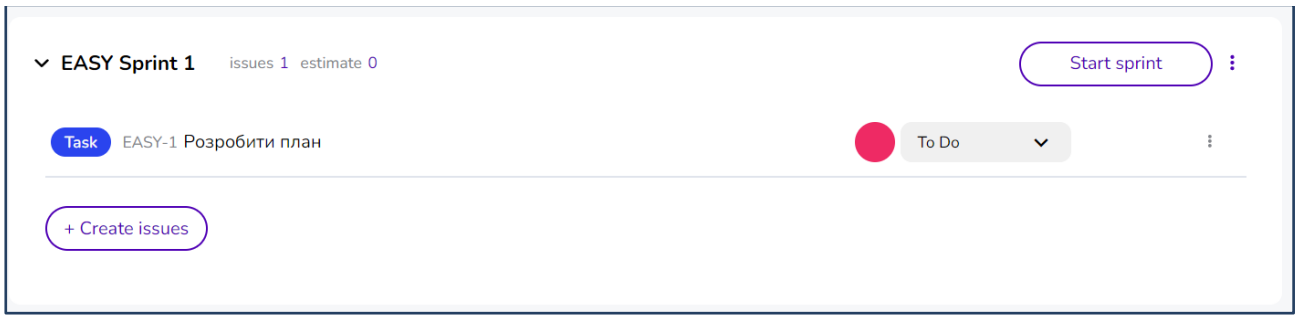


Рис. 2.32. Таска тепер знаходиться в спрінті

Після натиску на кнопку “Розпочати спрінт” ми побачимо діалогове вікно створення спрінта (рис. 2.33.).

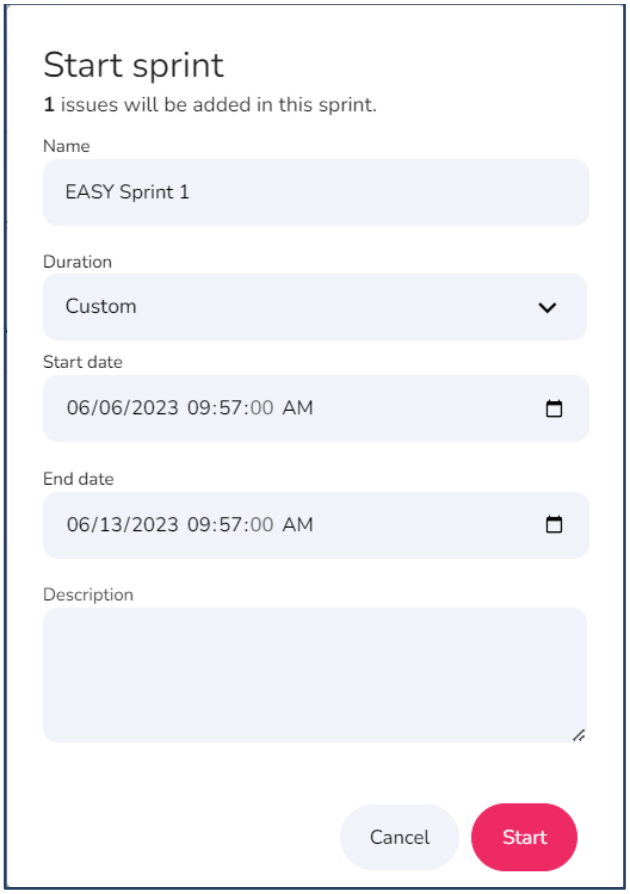


Рис. 2.33. Діалогове вікно створення спрінта

Далі потрібно перейти на сторінку “Board” – тепер ми бачимо Kanban дошку для цього спрінту, вже з наявною задачою “Розробити план” в колонці “To Do” (рис. 2.34.).

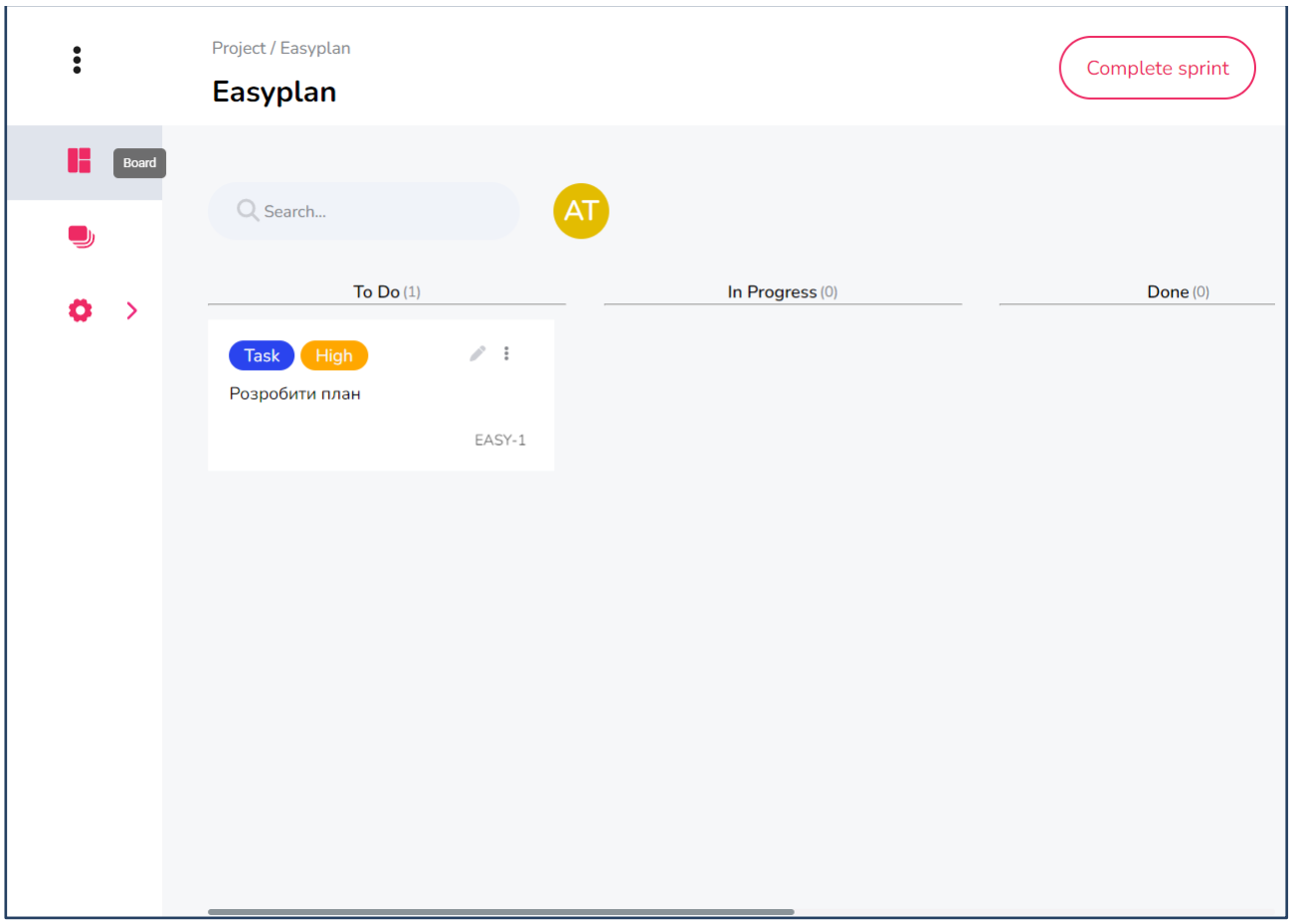


Рис. 2.34. Kanban дошка спрінта

Щоб змінити стан виконання цієї задачі потрібно або перетягнути за допомогою мишки з одної колонки в іншу (рис. 2.35.) або ж натиснути на олівець та змінити відповідне поле цієї задачі (рис. 2.36.).

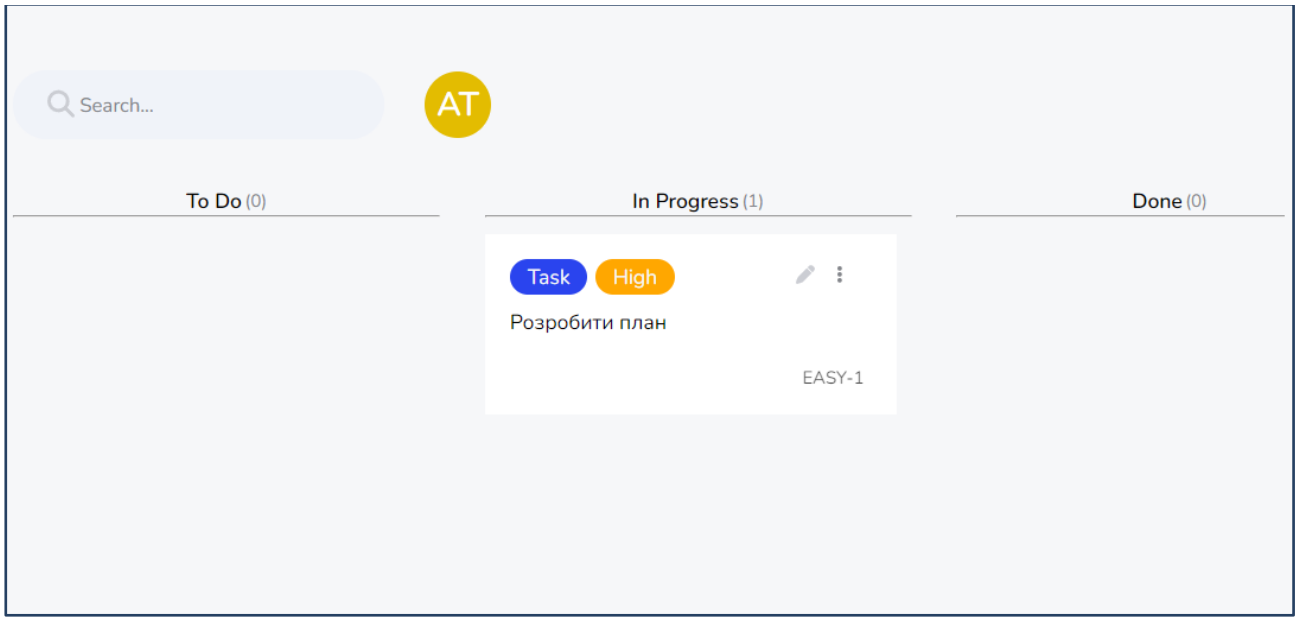


Рис. 2.35. Задачу було перетягнуто в колонку “In Progress”, використовуючи технологію Drag&Drop

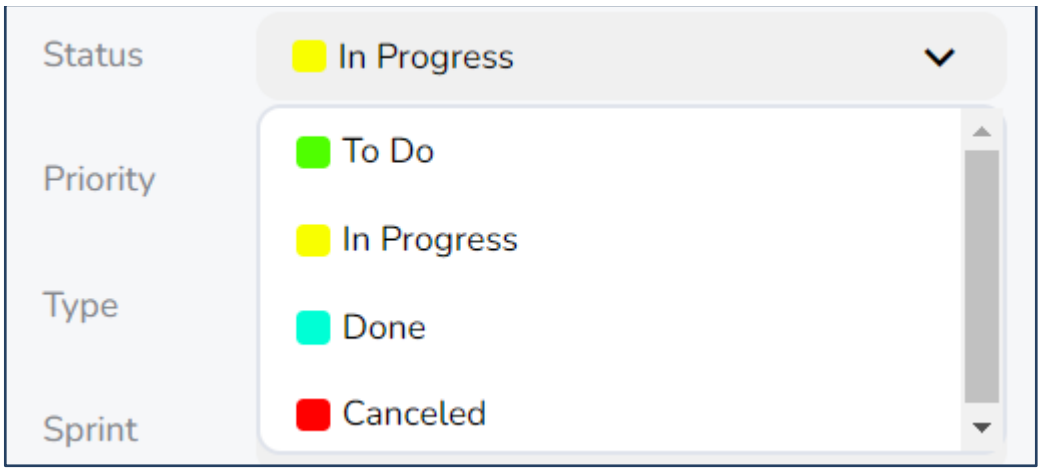


Рис. 2.36. Зміна статусу задачі

Після зміни задачі нас про це повідомлять та ми побачимо, що вона тепер знаходиться в обраній нами колонці (рис. 2.37.).

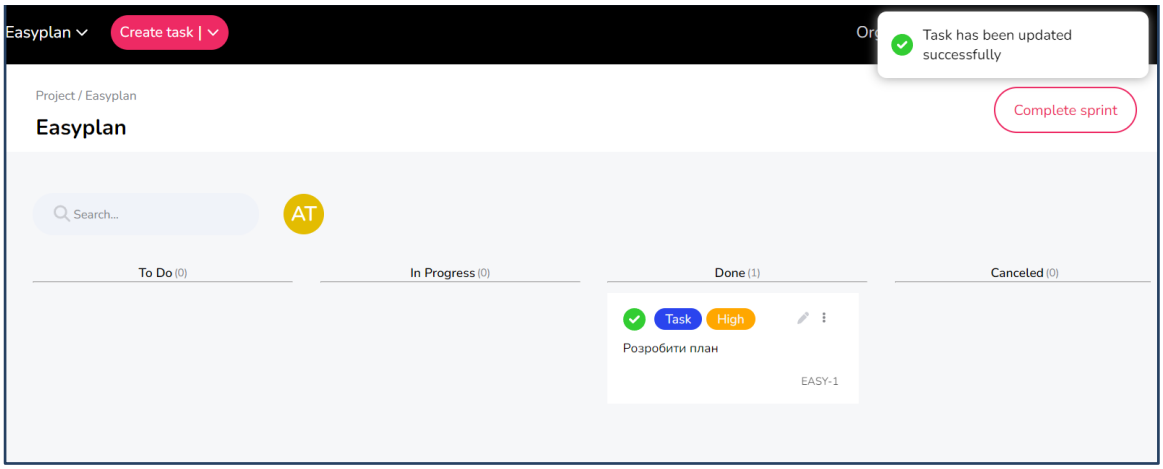


Рис. 2.37. Зміна статусу задачі

Тепер давайте створимо іншого користувача, щоб ми могли працювати з ним над проєктом разом. Для цього натисніть на аватар поточного користувача в хедері та натисніть “Log out” (рис. 2.38.). Як бачимо, виконається вихід з акаунту був виконаний та нас перенаправить на сторінку авторизації (рис. 2.39.).

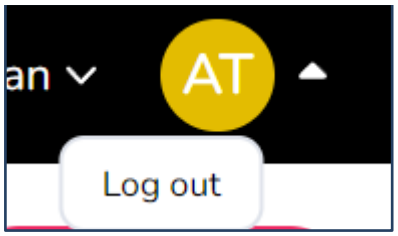


Рис. 2.38. Кнопка логауту, як опція в дропдауні

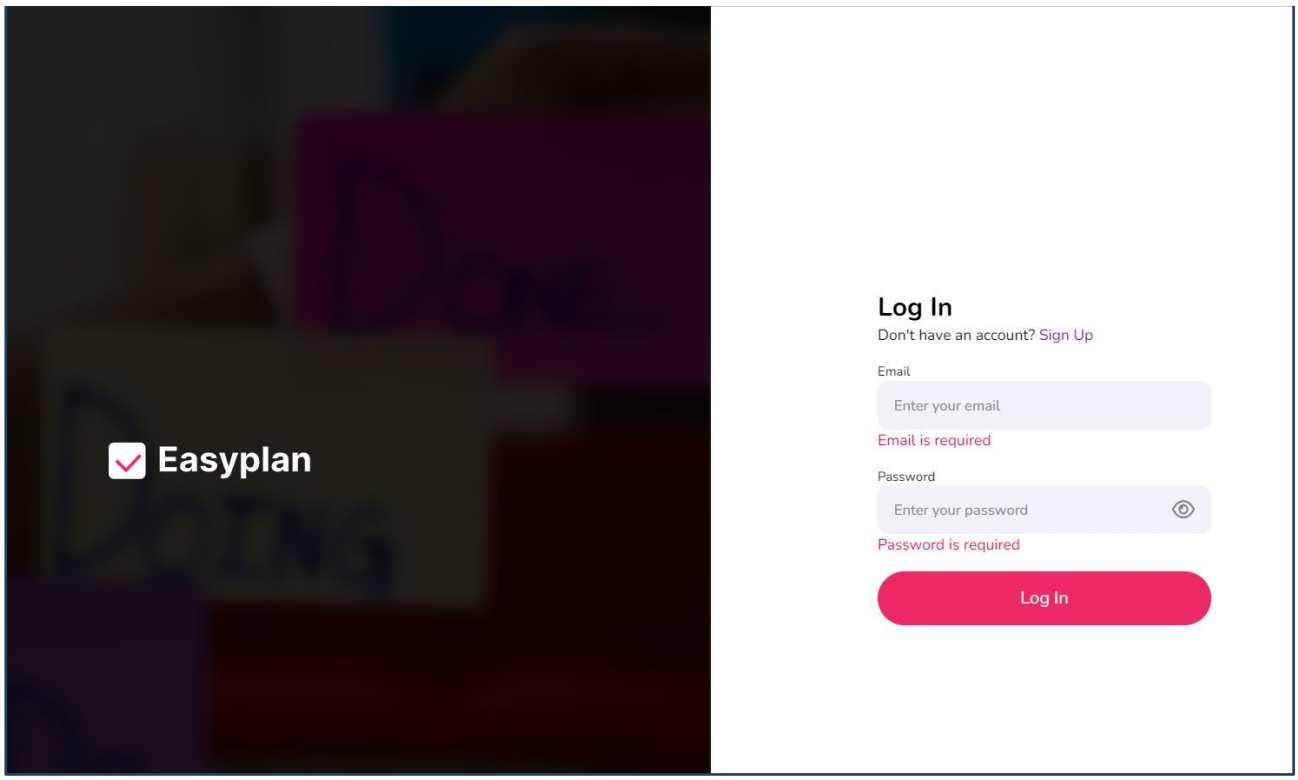


Рис. 2.39. Знову бачимо сторінку авторизації

Потім я проробив ті ж дії для реєстрації нового користувача, він не буде учасником жодного проєкту та організації (рис. 2.40.), тому я знову авторизуюся під першим користувачем (рис. 2.41.) та перейду до додавання нового користувача до організації. Для цього я натисну кнопку “Edit” в організації та введу пошту нового користувача, щоб його додати (рис. 2.42.).

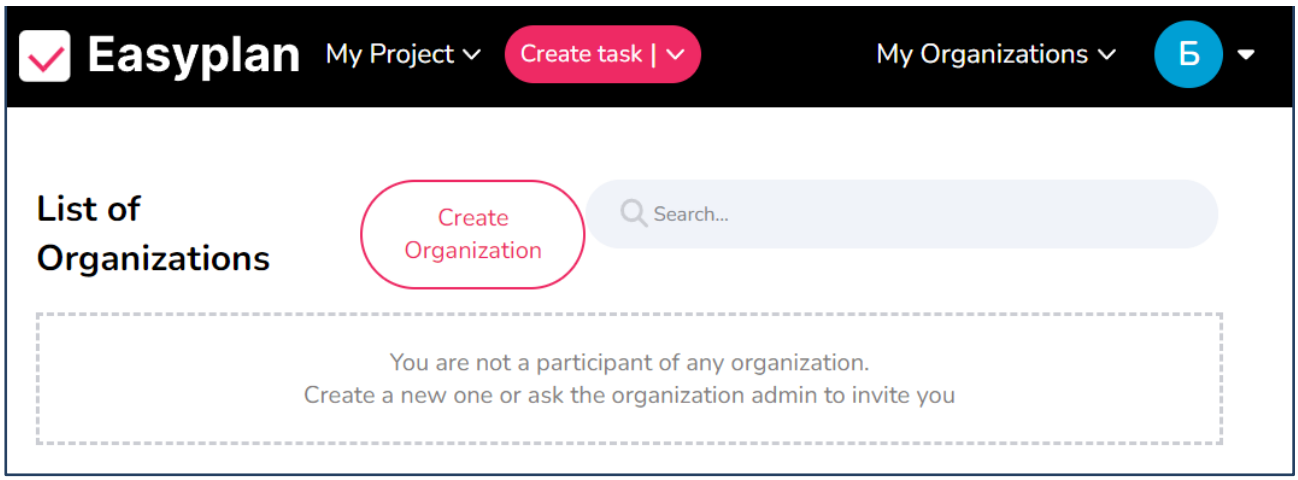


Рис. 2.40. Новий користувач не має організацій та проєктів

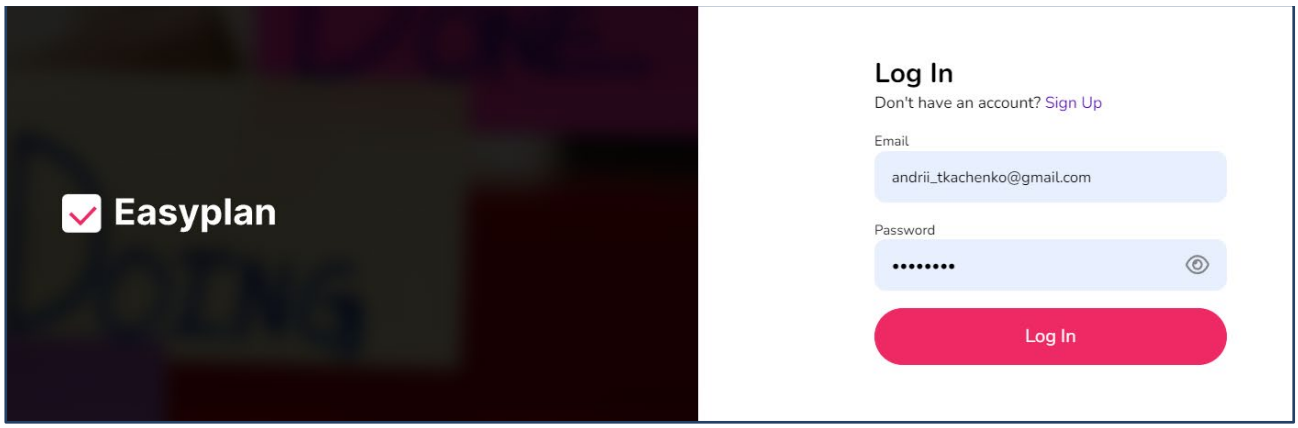


Рис. 2.41. Авторизація до аккаунту першого користувача

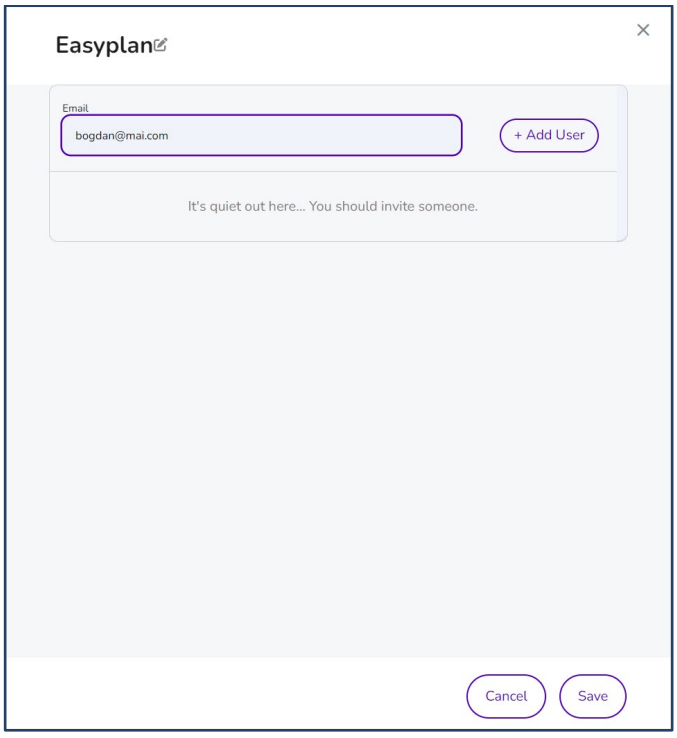


Рис. 2.42. Модальне вікно редагування організації

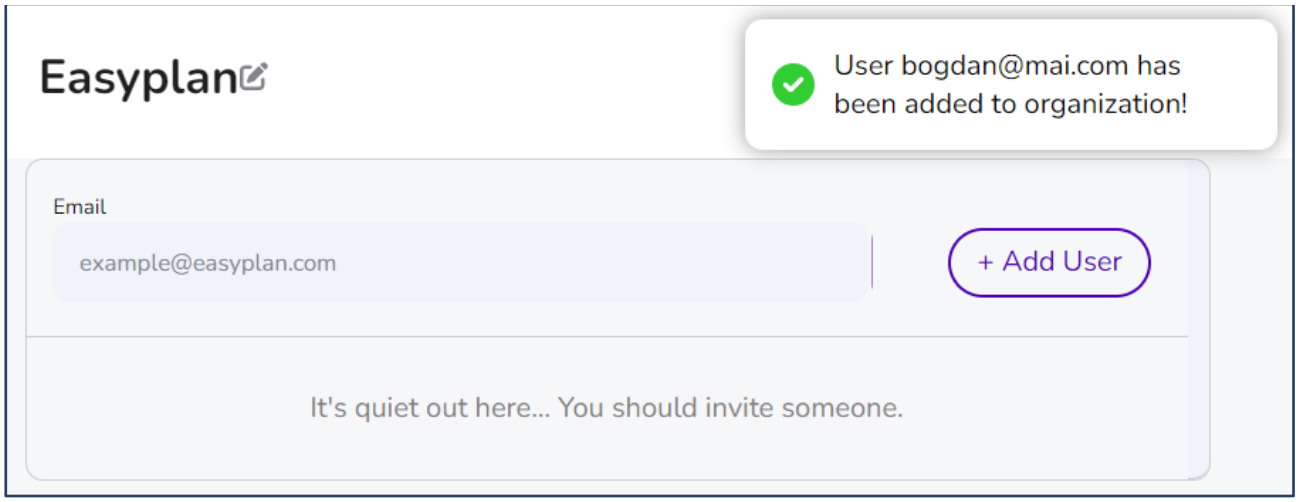


Рис. 2.43. Користувач був успішно доданий

Рис. 2.44. Організація з’явилася в списку організацій доданого користувача

Авторизувавшись під іншим юзером в іншому браузері, я побачу, що в нього є доступ до організації, але немає доступу до проєкту, тому потрібно буде додати його й до проєкту (рис. 2.45.).

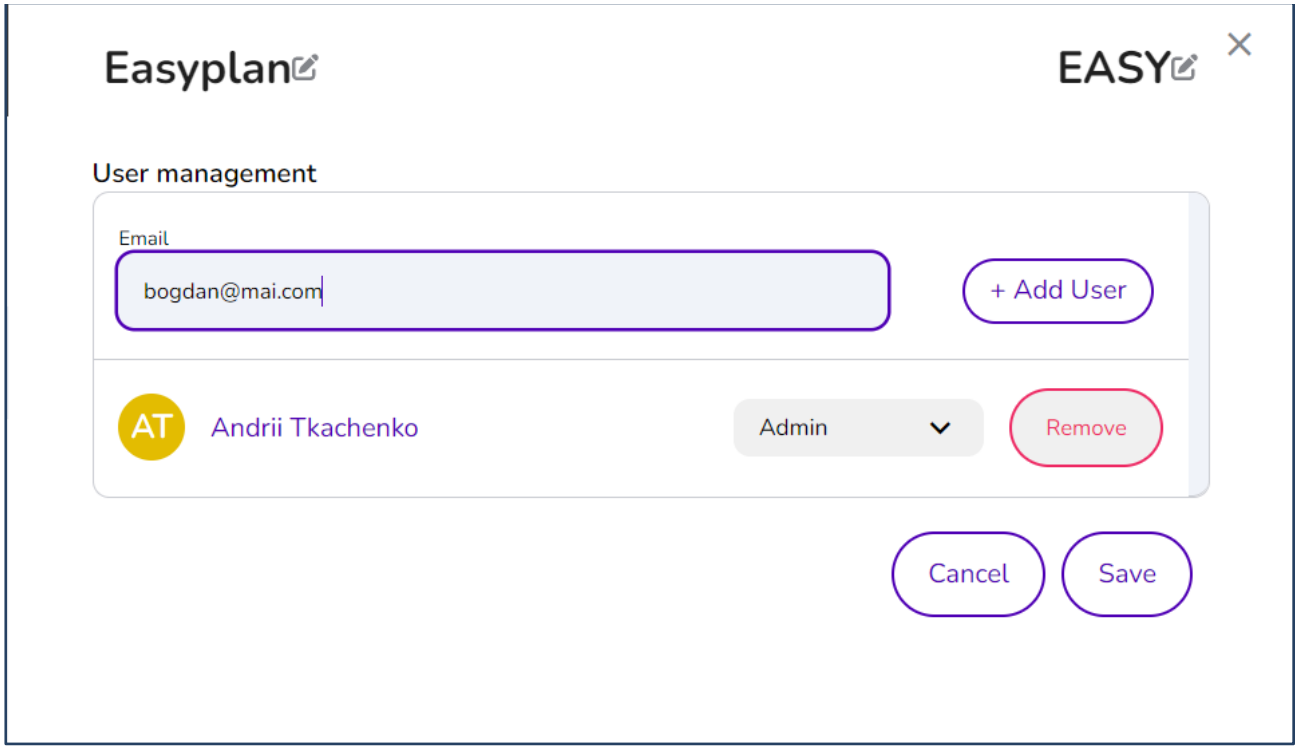


Рис. 2.45. Модальне вікно редагування проєкту

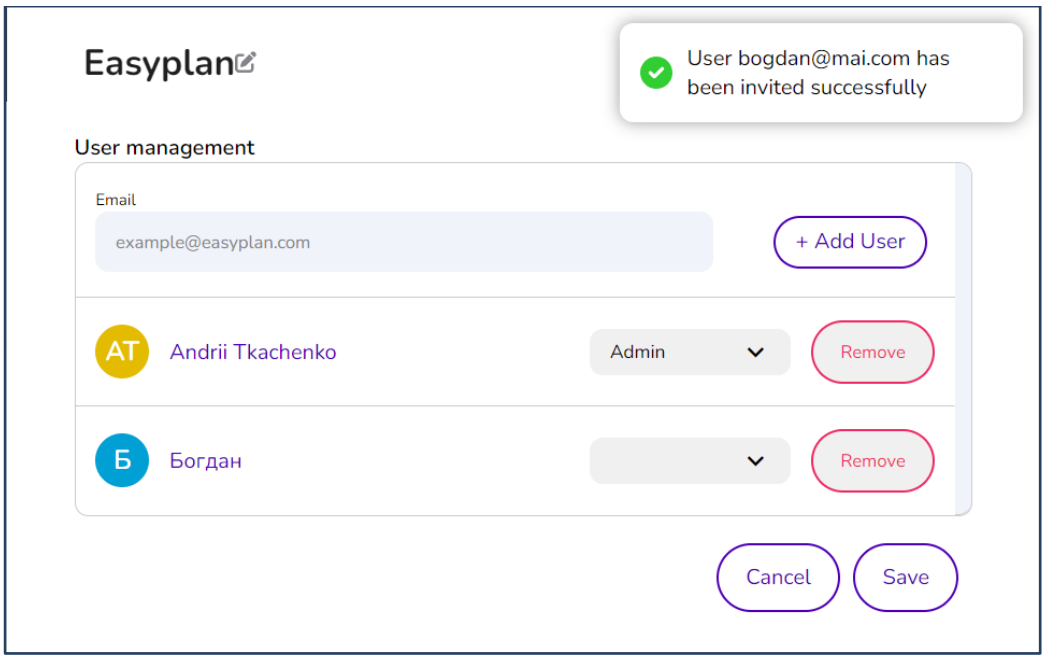


Рис. 2.46. Користувач був доданий до проєкту

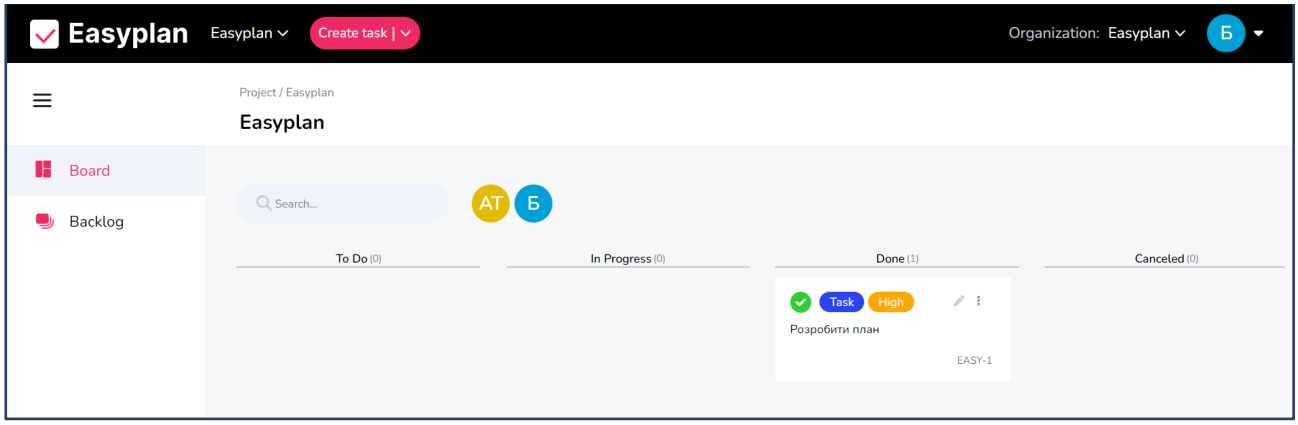


Рис. 2.47. Тепер в користувача “Богдан” є доступ до дошки

Переглянемо налаштування проєкту, натиснувши на відповідний пункт в лівому меню (рис. 2.48.). Лише адміністратор проєкту може налаштовувати параметри задач. Налаштувань є лише три: тип (рис. 2.52.), пріорітет (рис. 2.50.) та стан (рис. 2.53.). Можна додавати нові сутності, видаляти та редагувати.

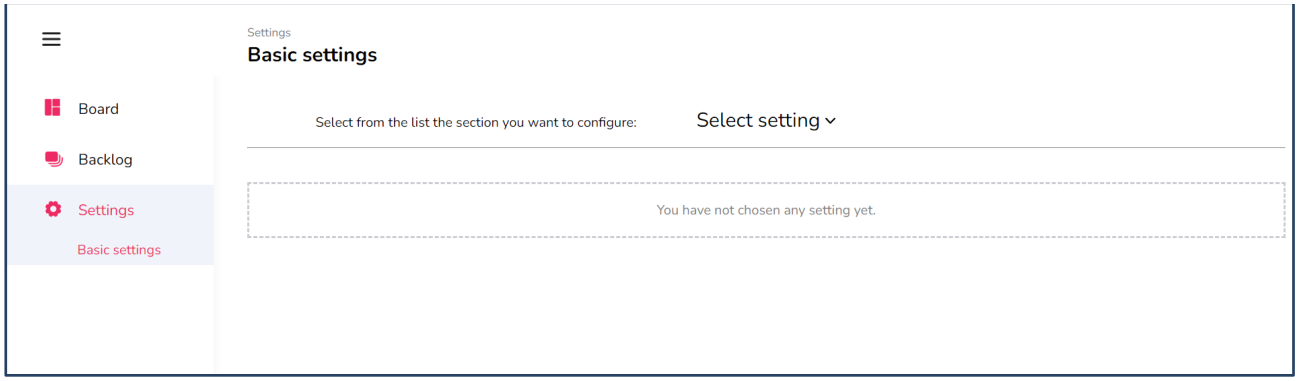


Рис. 2.48. Сторінку налаштувань проєкту

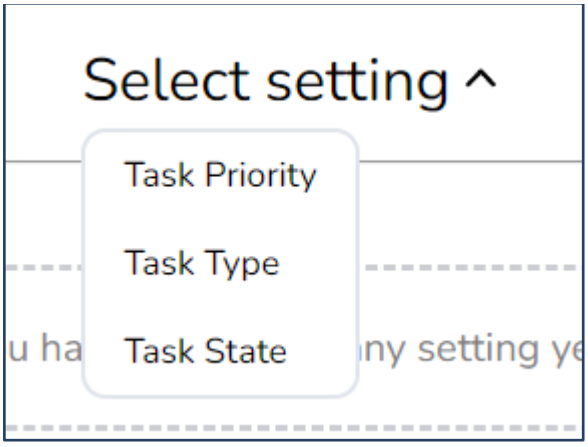


Рис. 2.49. Дропдаун вибору налаштувань

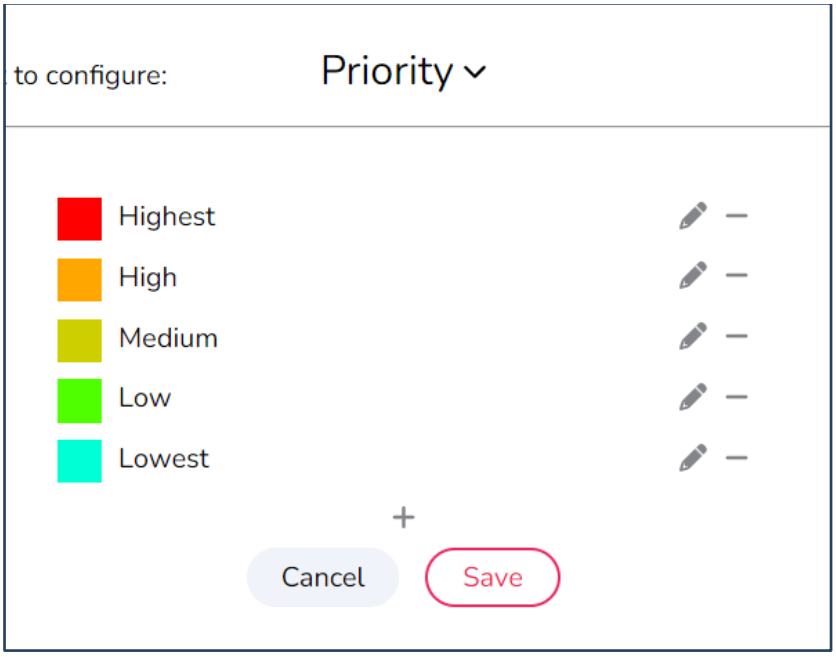


Рис. 2.50. Сторінка налаштувань пріоритетів задач

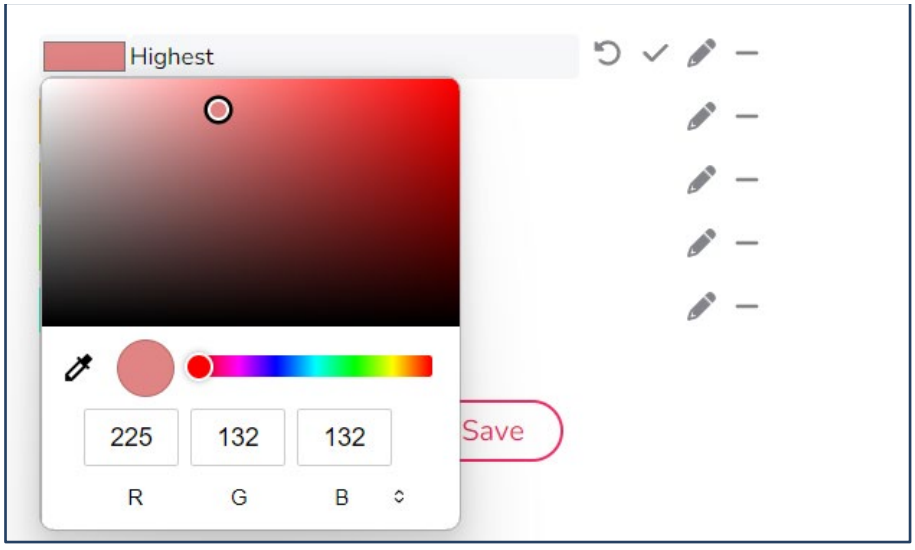


Рис. 2.51. Можливість зміни тексту та вибору кольору

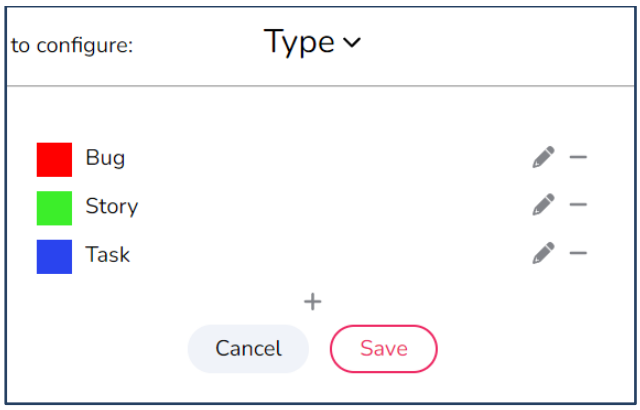


Рис. 2.52. Сторінка налаштувань типів задач

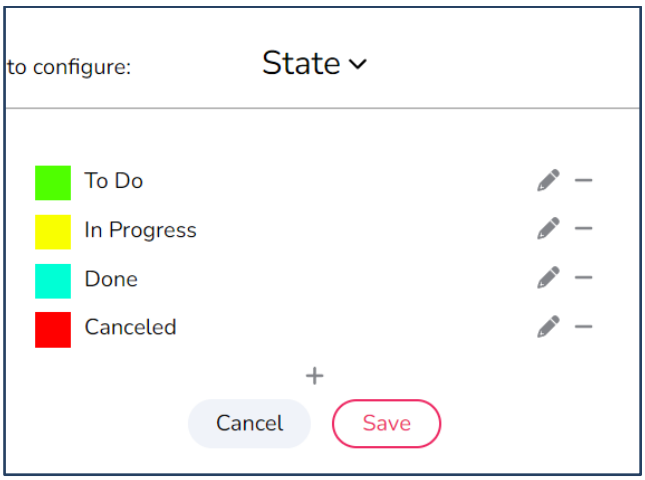


Рис. 2.53. Сторінка налаштувань станів задач

Перед тим, як завершити спрінт давайте наповнимо дошку актуальними задачами (рис. 2.54.).

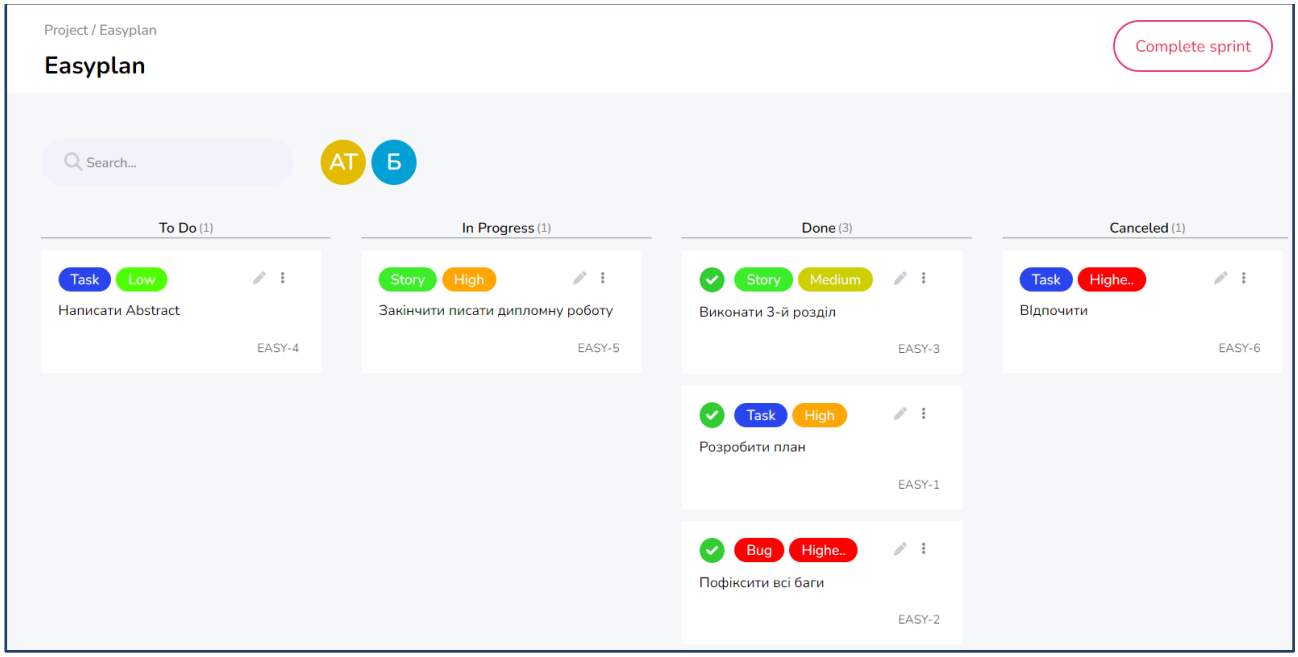


Рис. 2.54. Заповнена Kanban дошка

Натиснемо на кнопку “Завершити спрінт” (рис. 2.55.).

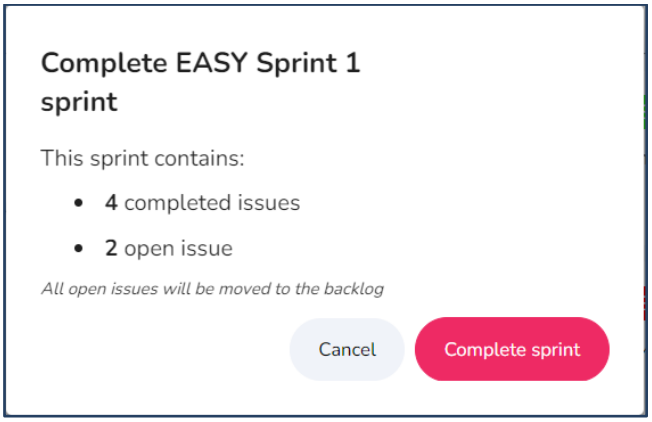


Рис. 2.55. Модальне вікно “Закінчити спрінт”

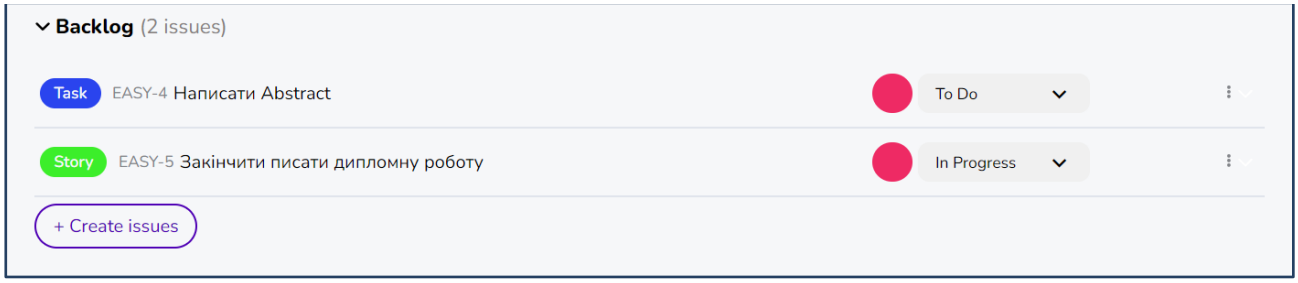


Рис. 2.56. Незавершені задачі повертаються в беклог

Інформацію про завершені спрінти можна переглянути в беклозі натиснувши на кнопку “Hystory” з замочком (рис. 2.57.).

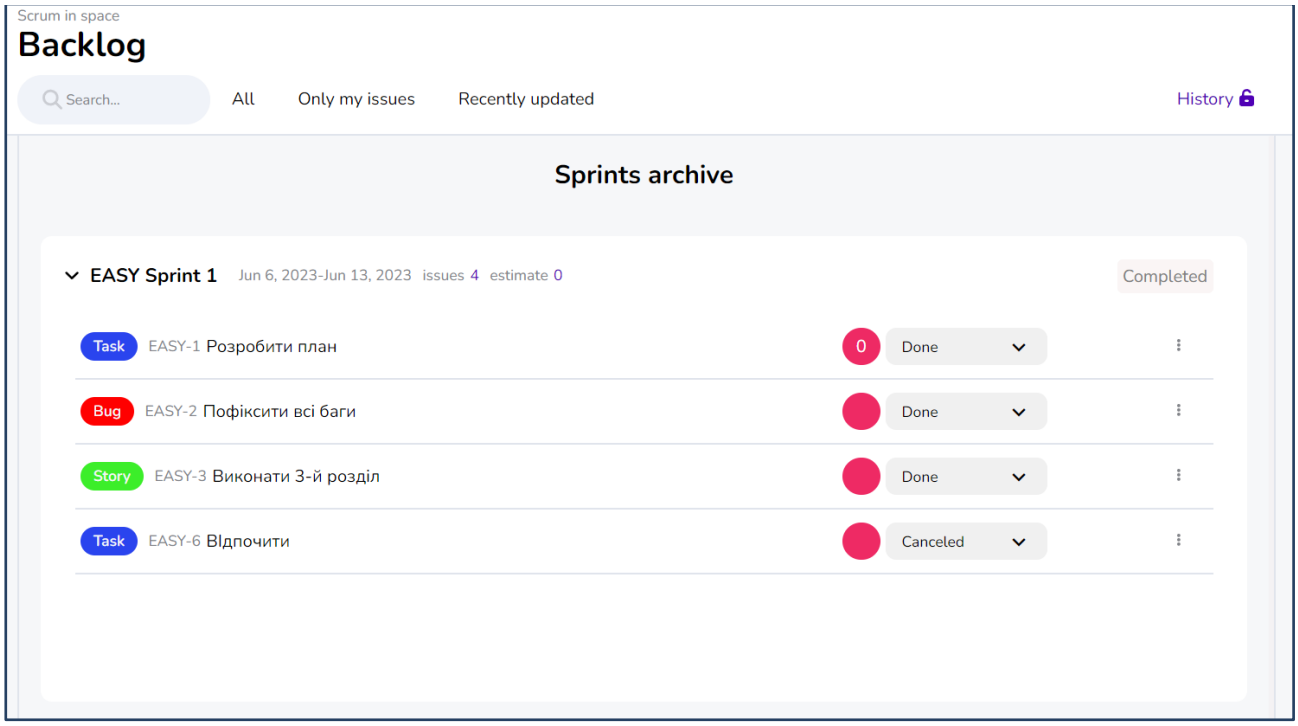


Рис. 2.57. Завершені спрінти можна переглядати в історії

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів – 4500;
- коефіцієнт корекції програми в ході її розробки – 0,1;
- коефіцієнт складності програми – 1,3;
- годинна заробітна плата програміста – 209,2 грн/год;
- коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,3;
- коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;
- вартість машино-години ЕОМ – 25 грн/год;

Заробітна плата за годину Junior .Net Software Engineer була вирахувана згідно «Української спільноти програмістів (DOU)». Станом на 2022 рік зарплата Junior .Net Engineer вар'юється від 743$ до 1275$. Вирахувавши середню заробітну плату програміста маємо плату 1009$ у місяць. При курсі валют НБУ на кінець грудня 2022 року один американський долар дорівнює 36,5 грн, тому середня зарплата в гривнях дорівнює 36 828 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 209,2 грн.

Виходячи з того, що для розробки даної системи необхідною є значна потужність ПК для безперервної роботи серверу, зберігання та підтримки великої кількості даних на сервері, вдалим рішенням буде оренда ноутбуку.

Вартість оренди ноутбуку на місяць становить 4000 грн. При стандартному графіку (160 годин/місяць) вартість машино-години ЕОМ за годину роботи буде становити 25 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (3.2)$$

де q - передбачуване число операторів (4500);

C - коефіцієнт складності програми (1,3);

p - коефіцієнт корекції програми в ході її розробки (0,1).

За формулою (3.2) умовне число операторів в програмі становить:

$$Q = 4500 * 1{,}3 * (1 + 0{,}1) = 6435 \quad (3.3)$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75..85)*k}, \quad (3.4)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,3);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 2 до 3 років він складає 1,2.

Збільшення витрат праці внаслідок недостатнього опису завдання будемо приймати не більше 50% (B = 1,3).

З урахуванням коефіцієнта кваліфікації k = 1,1 за формулою (3.4) отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{6435*1,3}{85*1,1} = 89,4, \text{ людино-годин}, \quad (3.5)$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)*k}, \quad (3.6)$$

де Q – умовне число операторів програми (6435);

k – коефіцієнт кваліфікації програміста (1,1).

Згідно формулі (3.6), отримаємо:

$$t_a = \frac{6435}{25*1,1} = 234, \text{ людино-годин}, \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)*k}, \quad (3.8)$$

Маючи формулу (3.8) витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{6435}{25*1,1} = 234, \text{ людино-годин,} \quad (3.9)$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)*k}, \quad (3.10)$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.10):

$$t_{\text{отл}} = \frac{6435}{5*1,1} = 1170, \text{ людино-годин,} \quad (3.11)$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1,5 * t_{\text{отл}}, \quad (3.12)$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.12):

$$t_{\text{отл}}^{k} = 1,5 * 1170 = 1755, \text{ людино-годин,} \quad (3.13)$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \quad (3.14)$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)*k}, \quad (3.15)$$

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 * t_{др}, \quad (3.16)$$

Маючи формули (3.15), (3.16) та (3.14) обчислюємо витрати праці на документацію:

$$t_{др} = \frac{6435}{20*1{,}1} = 292{,}5, \text{ людино-годин}, \quad (3.17)$$

$$t_{до} = 0{,}75 * 292{,}5 = 219{,}37, \text{ людино-годин}, \quad (3.18)$$

$$t_{д} = 292{,}5 + 219{,}37 = 511{,}87, \text{ людино-годин}, \quad (3.19)$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$t = 50 + 89{,}4 + 234 + 234 + 1170 + 511{,}87 = 2289{,}25$, людино-годин. (3.20)

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ КПО включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.21)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * С_{ПР}, \text{ грн,} \quad (3.22)$$

де t - загальна трудомісткість, людино-годин (2289,27);

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 209,2 грн / год, за формулою (3.22) отримуємо:

$$З_{ЗП} = 2289{,}27 * 209{,}2 = 478\,911, \text{ грн,} \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * С_{МЧ}, \text{ грн,} \quad (3.24)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год (1170 год);

$С_{МЧ}$ - вартість машино-години ЕОМ, грн/год (25 грн/год).

Підставивши в формулу (3.24) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{МВ} = 1170 * 25 = 29250, \text{ грн,} \quad (3.25)$$

Звідси, за формулою (3.21), витрати на створення програмного продукту:

$$К_{ПО} = 478911 + 29250 = 508161, \text{ грн}, \quad (3.26)$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k * F_p}, \text{ міс}, \quad (3.27)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p =176 годин).

За формулою 3.27 очікуваний період створення програмного забезпечення:

$$T = \frac{2289,25}{1*176} \approx 13 \text{ міс}. \quad (3.28)$$

Висновки: Сайт-система відстеження помилок, призначеної для управління проєктами та організації робочого процесу була розроблена для зручного та ефективного планування задач користувачів.

Розробка даного програмного забезпечення становить 508161 грн без додаткових витрат.

Згідно отриманих даних при розрахунках, очікуваний час розробки становить 2289,25 години, або 13 місяців, з урахуванням стандартного робочого графіку. Результат, що отримано, залежить від незначного коефіцієнта кваліфікації розробника, та у свою чергу включає час на виконання досліджень та розробку концепції для втілення поставленої задачі, надалі програмування за створеною концепцією, тестування програмного продукту та впровадження документації.

ВИСНОВКИ

У рамках кваліфікаційного проекту було розроблено сайт-систему відстеження помилок, яка призначена для управління проєктами та організації робочого процесу. Це програмне забезпечення призначене для планування цілей та задач користувачів з використанням зручного інтерфейсу.

Практичне застосування допомагає користувачам відстежувати їхні цілі та ефективно керувати задачами, використовуючи підхід Scrum та Kanban дошку з інтерактивники картками для візуалізації процесу.

У процесі виконання кваліфікаційного проекту були вирішені такі задачі:

- Проведено аналіз предметної області поставленої задачі.
- Розроблено дизайн додатку.
- Спроектовано базу даних.
- Реалізовано серверну частину системи.
- Реалізовано клієнтську частину системи.
- Визначено трудомісткість розробленої системи.
- Оцінено вартість додатку.

Розроблений веб-додаток надає такі можливості:

- Реєстрація та авторизація користувачів.
- Управління організаціями проєктами та задачами.
- Інтерактивна взаємодія між елементами веб-сайту для забезпечення зручного користувацького досвіду.
- Зручна навігація між різними середовищами веб-додатку.
- Збереження даних на БД.

Для реалізації продукту було необхідно розробити як серверну, так і клієнтську частини. Для створення серверної частини була використана мова програмування C# та платформа .Net, а також популярний фреймворк ASP.Net для створення веб-додатків. База даних була реалізована з використанням Entity Framework. Щодо клієнтської частини, вона була розроблена з використанням

мови програмування Typescript, фреймворка Angular, а стилізація веб-сторінки здійснювалася за допомогою препроцесора CSS - SASS.

В економічному розділі визначено трудомісткість розробленої інформаційної системи (2289 люд-год), проведений підрахунок вартості роботи по створенню програми (508161 грн.) та розраховано час на його створення (13 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. .NET | Build. Test. Deploy. / URL: https://dotnet.microsoft.com/en-us/
2. A tour of C# - Overview | Microsoft Learn. / URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
3. ASP.NET | Open-source web framework for .NET. / URL: https://dotnet.microsoft.com/en-us/apps/aspnet/
4. What is the Agile methodology? / URL: https://www.atlassian.com/agile/
5. How the kanban methodology applies to software development / URL: https://www.atlassian.com/agile/kanban/
6. What is scrum and how to get started / URL: https://www.atlassian.com/agile/scrum/
7. 14 Best Jira Alternatives to Try in 2023 (Free and Paid Competitors) / URL: https://clickup.com/blog/jira-alternatives/
8. Що таке Jira і як з нею працювати / URL: https://iampm.club/ua/blog/shho-take-jira-i-yak-z-neyu-praczyuvati/
9. What is Docker? / URL: https://aws.amazon.com/docker/
10. What is Angular? / URL: https://angular.io/guide/what-is-angular#what-is-angular/
11. Що таке ASP.NET? Принцип функціонування та моделі розробки / URL: https://highload.today/uk/shho-take-asp-net-printsip-funktsionuvannya-ta-modeli-rozrobki/
12. What is PostgreSQL ? Benefits of using PostgreSQL / URL: https://aws.amazon.com/rds/postgresql/what-is-postgresql
13. Designing an MVC Model for Rapid Web Application Development / URL: https://www.sciencedirect.com/science/article/pii/S187770581400352X
14. How to build and deploy a three-layer architecture application with C# / URL: https://enlabsoftware.com/development/how-to-build-and-deploy-a-three-layer-architecture-application-with-c-sharp-net-in-practice.html

15. Entity Framework Core with PostgreSQL — Database First / URL: https://medium.com/@ankushjain358/entity-framework-core-with-postgresql-database-first-ab03bf1079c4

16. How to Implement JWT Authentication in Web API Using .Net 6.0, Asp.Net Core / URL: https://www.c-sharpcorner.com/article/how-to-implement-jwt-authentication-in-web-api-using-net-6-0-asp-net-core/

17. What is Automapper in C# and how to use it / URL: https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/automapper-in-c-sharp

18. Best Practices & Guidelines for web applications using Angular / URL: https://blogs.halodoc.io/angular-best-practices/

19. Understanding Dockerfile. Get a in-depth look at the internals make up of a Dockerfile and the commands within it / URL: https://dzone.com/articles/understanding-dockerfile

20. What is Docker Compose? How to Use it with an Example / URL: https://www.freecodecamp.org/news/what-is-docker-compose-how-to-use-it/

ДОДАТОК А

КОД ПРОГРАМИ

easyplan-header.component.ts // компонент хедер

```
import { Component, Input, Output, EventEmitter, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import {
  faCaretDown,
  faCaretUp,
  faMagnifyingGlass,
} from '@fortawesome/free-solid-svg-icons';
import { OrganizationModel } from 'src/core/models/organization/organization-model';
import { UserModel } from 'src/core/models/user/user-model';
import { AuthService } from 'src/core/services/auth.service';
import { OpenDialogService } from 'src/core/services/open-dialog.service';
import { Observable } from 'rxjs';
import { ProjectModel } from 'src/core/models/project/project-model';
import { ProjectService } from 'src/core/services/project.service';
import { concatMap, map } from 'rxjs/operators';
import { OrganizationService } from 'src/core/services/organization.service';
import { UserRole } from 'src/core/models/user/user-roles';
import { UserProjectRoleModel } from 'src/core/models/user/user-project-roles-model';
import { BusinessRole } from '../select-users/select-user-models';
import { GetCurrentServices } from 'src/core/services/get-current-services.service';
import { GetCurrentProjectService } from 'src/core/services/get-current-project.service';
import { GetCurrentOrganizationService } from 'src/core/services/get-current-organization.service';

@Component({
  selector: 'easyplan-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.sass'],
})
export class HeaderComponent implements OnInit {
  public notifications: Notification[] = [];
  public searchIcon = faMagnifyingGlass;
  public currentUser: UserModel;
  public currentOrganizationId: number;
  public currentProjectId: number;
  public currentProject: ProjectModel;
  @Input() hasLogo = false;
  @Output() isChanged = new EventEmitter<Observable<void>>();

  public upArrowIcon = faCaretUp;
  public downArrowIcon = faCaretDown;
  public isCurrentUserAdmin = false;
  public isCurrentUserProjectAdmin = false;

  // eslint-disable-next-line max-params
  constructor(
    private authService: AuthService,
    private router: Router,
    private openDialogService: OpenDialogService,
    private getCurrentProjectService: GetCurrentProjectService,
    private getCurrentOrganizationService: GetCurrentOrganizationService,
    private projectService: ProjectService,
    private organizationService: OrganizationService,
    private scopeGetCurrentEntityService: GetCurrentServices
  ) { }


  ngOnInit(): void {
    this.subscribeToCurrentUser();
```

```
  this.subscribeToCurrentOrganization();
  this.subscribeToCurrentProject();
}
changeOrganizationId(val: number): void {
  this.currentOrganizationId = val;
}
private subscribeToCurrentOrganization(): void {
  this.getCurrentOrganizationService.currentOrganizationId$.subscribe(
    (result) => {
      this.currentOrganizationId = result;
    },
  );
}
public subscribeToCurrentUser(): void {
  this.scopeGetCurrentEntityService.getCurrentUserService.getCurrentUser();

  this.scopeGetCurrentEntityService.getCurrentUserService.currentUser$.subscribe(
    (user) => {
      if (!user) {
        return;
      }
      this.currentUser = user;


      this.scopeGetCurrentEntityService.getCurrentProjectService.currentProjectId$.subscribe(
        (project) => {
          this.currentProjectId = project;


          this.permissionToEdit();
        },
      );
    },
  );
}
public subscribeToCurrentProject(): void {
  this.getCurrentProjectService.currentProjectId$
    .pipe(
      concatMap((id) => {
        if (id > 0) {
          return this.projectService.getProjectById(id);
        }
        return [];
      }),
      map((resp) => resp.body),
    )
    .subscribe((project) => {
      if (!project) {
        return;
      }
      this.currentProject = project;
    });
}

openCreateOrganizationDialog(): void {
  this.openDialogService
    .openCreateOrganizationDialog(this.currentUser)
    .subscribe((result: OrganizationModel) => {
      if (!result) {
        return;
      }
```

```
      this.getCurrentOrganizationService.updateOrganization(
        result,
      );
    });
}

public openCreateProjectDialog(): void {
  this.openDialogService
    .openCreateProjectDialog(this.currentOrganizationId)
    .subscribe((result) => {
      if (!result) {
        return;
      }


      this.getCurrentProjectService.updateProject(result);
    });
}

public openProjectsPage(): void {
  this.router.navigate(['/projects'], { replaceUrl: true });
  window.scroll(0, 0);
}

public logoutClick(): void {
  this.authService.logout();
}

public profileSettingsClick(): void {
  this.router.navigate(['/user/profile'], { replaceUrl: true });
}

public checkUrl(): void {
  this.isChanged.emit();
}

permissionToEdit(): void {
  const organizationId =
    this.scopeGetCurrentEntityService.getCurrentOrganizationService
      .currentOrganizationId;
  this.organizationService
    .getOrganization(organizationId)
    .subscribe((resp) => {
      const currentOrganization = resp.body as OrganizationModel;
      const role = this.currentUser.organizationRoles.find(
        (r) =>
          r.organizationId === organizationId &&
          r.userId === this.currentUser.id,
      )?.role as UserRole;
      if (
        role >= UserRole.projectAdmin ||
        currentOrganization.authorId === this.currentUser.id
      ) {
        this.isCurrentUserAdmin = true;
      } else {
        this.isCurrentUserAdmin = false;
      }


      const projectRole = this.currentUser.roles?.find(
        (r) =>
          r.projectId === this.currentProjectId &&
          r.userId === this.currentUser.id,
```

```
      ) as UserProjectRoleModel;


      if (projectRole) {
        if (
          projectRole.roleId == BusinessRole.Admin ||
          this.isCurrentUserAdmin
        ) {
          this.isCurrentUserProjectAdmin = true;
        } else {
          this.isCurrentUserProjectAdmin = false;
        }
      }
    });
  }
}
```

task-editing.component.ts // компонент редагування задачі

```
import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
import { TaskModel } from 'src/core/models/task/task-model';
import { UserModel } from 'src/core/models/user/user-model';
import {
  faCheckToSlot,
  faXmark,
  faLink,
  faPaperclip,
  faShareNodes,
  faEllipsisVertical,
  faFaceSmile,
  faPen,
  faFlag,
  IconDefinition,
  faPenToSquare,
} from '@fortawesome/free-solid-svg-icons';
import { BaseComponent } from 'src/core/base/base.component';
import { AngularEditorConfig } from '@kolkov/angular-editor';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { ProjectModel } from 'src/core/models/project/project-model';
import { SprintModel } from 'src/core/models/sprint/sprint-model';
import { EditorConfig } from 'src/core/settings/angular-editor.config';
import { TaskStateModel } from 'src/core/models/task/task-state-model';
import { TaskPriorityModel } from 'src/core/models/task/task-priority-model';
import { TaskTypeModel } from 'src/core/models/task/task-type-model';
import { EasyplanDropdownOption } from '../easyplan-dropdown/dropdown.component';
import { GetCurrentUserService } from 'src/core/services/get-current-user.service';
import { ProjectService } from 'src/core/services/project.service';
import { GetCurrentOrganizationService } from 'src/core/services/get-current-organization.service';
import { SprintService } from 'src/core/services/sprint.service';
import { TaskService } from 'src/core/services/task.service';
import { takeUntil } from 'rxjs/operators';
import { TaskStorageService } from 'src/core/services/task-storage.service';
import { ToastrNotificationService } from 'src/core/services/toastr-notification.service';
import { TaskUpdateModel } from 'src/core/models/task/task-update-model';
import { BoardType, BoardViewModel, UserCardModel } from '../select-users/select-user-models';
import { UserService } from 'src/core/services/user.service';
import { Observable } from 'rxjs';

@Component({
  selector: 'easyplan-task-editing',
  templateUrl: './task-editing.component.html',
  styleUrls: ['./task-editing.component.sass'],
})
export class TaskEditingComponent extends BaseComponent implements OnInit {
```

```
@Input() public task: TaskModel;
@Input() public currentUser: UserModel;
@Input() public currentSprint: SprintModel;
@Input() public btnText = 'Edit task';
@Input() public btnClass = 'btn stroke';
@Input() public btnIcon: IconDefinition | undefined;
@Input() public isCurrentUserAdmin = false;
@Input() public isCurrentUserProjectAdmin = false;
@Output() public isChanging = new EventEmitter<boolean>();

public organizationId: number;
public editTaskForm = {} as FormGroup;
public editTaskFormDefaultValues: unknown;
public board: BoardViewModel;

public closeIcon = faXmark;
public linkIcon = faLink;
public checkIcon = faCheckToSlot;
public shareIcon = faShareNodes;
public paperClipIcon = faPaperclip;
public ellipsisIcon = faEllipsisVertical;
public faceSmileIcon = faFaceSmile;
public editIcon = faPen;
public editSquareIcon = faPenToSquare;
public flagIcon = faFlag;
public isChanged = new Observable<void>();


public descriptionEditorShow = false;

public statusOptions: EasyplanDropdownOption[] = [];
public priorityOptions: EasyplanDropdownOption[] = [];
public typeOptions: EasyplanDropdownOption[] = [];
public projectOptions: EasyplanDropdownOption[] = [];
public sprintOptions: EasyplanDropdownOption[] = [];

public priorities: TaskPriorityModel[] = [];
public states: TaskStateModel[] = [];
public types: TaskTypeModel[] = [];

public projectUsers: UserModel[] = [];

public editorConfig: AngularEditorConfig = EditorConfig;

public isOpen = false;
public isDataLoaded = false;

// eslint-disable-next-line max-params
constructor(
  private fb: FormBuilder,
  private currentUserService: GetCurrentUserService,
  private projectService: ProjectService,
  private currentOrganizationService: GetCurrentOrganizationService,
  private taskService: TaskService,
  private taskStorageService: TaskStorageService,
  private sprintService: SprintService,
  private notificationService: ToastrNotificationService,
  private userService: UserService,
) {
  super();
}
```

```
public isShow = false;


ngOnInit(): void {
  this.currentUserService.currentUser$.subscribe((user) => {
    this.currentUser = user;
  });


  this.currentOrganizationService.currentOrganizationId$.subscribe(
    (orgId) => {
      this.organizationId = orgId;


      if (!this.isDataLoaded) {
        return;
      }


      this.projectService
        .getProjectsByOrganizationId(this.organizationId)
        .pipe(takeUntil(this.unsubscribe$))
        .subscribe((resp) => {
          if (!resp.body) {
            return;
          }
          this.fillProjectOptions(resp.body);
          this.getProjectInfo(this.task.projectId);
        });
    },
  );


  this.editTaskForm = this.fb.group({
    project: [this.convertToOption(this.task.project)],
    summary: [
      this.task.summary,
      [Validators.minLength(2), Validators.maxLength(80)],
    ],
    sprint: [this.convertToOption(this.currentSprint)],
    status: [this.convertToOption(this.task.state)],
    priority: [this.convertToOption(this.task.priority)],
    type: [this.convertToOption(this.task.type)],
    description: [this.task.description, [Validators.maxLength(5000)]],
    assignees: [this.task.users],
  });


  this.editTaskFormDefaultValues = this.editTaskForm.value;


  this.editTaskForm.controls.project.valueChanges.subscribe(
    (option: EasyplanDropdownOption) => {
      this.getProjectInfo(option.id);
    },
  );


  this.board = {
    id: 1,
    type: BoardType.Board,
    users: this.task.users?.map((user) => this.convertToUserCard(user)) ?? [],
    hasRoles: true,
```

```
  };
}


public titleContent(event: Event): string {
  const input = event.target as HTMLElement;
  return input.innerText;
}

descriptionClick(): void {
  this.descriptionEditorShow = true;
}

descriptionEditorOutsideClick(): void {
  this.descriptionEditorShow = false;
}

get summaryErrorMessage(): string {
  const ctrl = this.editTaskForm.controls.summary;

  if (ctrl.errors?.['minlength']) {
    return 'Summary must be at least 2 characters';
  }
  if (ctrl.errors?.['maxlength']) {
    return 'Summary must not exceed  80 characters';
  }

  return '';
}


get descriptionErrorMessage(): string {
  const ctrl = this.editTaskForm.controls.description;

  if (ctrl.errors?.['maxlength']) {
    return 'Description must not exceed 5000 characters';
  }

  return '';
}

private convertToOption(
  item:
    | TaskStateModel
    | TaskPriorityModel
    | TaskTypeModel
    | SprintModel
    | ProjectModel
    | undefined,
): EasyplanDropdownOption {
  if (!item) {
    return {
      id: 0,
      title: '-',
      color: this.getColorById(0),
    };
  }


  if ('color' in item) {
    return {
      id: item.id,
      title: item.name,
```

```
      color: item.color,
    };
  }


  return {
    id: item.id,
    title: item.name,
    color: this.getColorById(item.id),
  };
}


private getColorById(id: number): string {
  if (id === 0) {
    return 'lightgray';
  }

  switch (id % 4) {
    case 1:
      return 'green';
    case 2:
      return 'yellow';
    case 3:
      return 'orange';
    case 0:
      return 'red';
  }


  return 'lightgray';
}


private fillProjectOptions(projects: ProjectModel[]): void {
  this.projectOptions = [];
  projects.forEach((element) => {
    this.projectOptions.push(this.convertToOption(element));
  });
}


private fillSprintOptions(sprints: SprintModel[]): void {
  this.sprintOptions = [
    {
      title: '-',
      id: 0,
      color: this.getColorById(0),
    },
  ];

  sprints.forEach((element) => {
    this.sprintOptions.push(this.convertToOption(element));
  });
}

private fillPriorityOptions(taskPriorities: TaskPriorityModel[]): void {
  this.priorityOptions = [];
  taskPriorities.forEach((element) => {
    this.priorityOptions.push(this.convertToOption(element));
  });
}
```

```
private fillStateOptions(states: TaskStateModel[]): void {
  this.statusOptions = [];
  states.forEach((element) => {
    this.statusOptions.push(this.convertToOption(element));
  });
}


private fillTypeOptions(types: TaskTypeModel[]): void {
  this.typeOptions = [];
  types.forEach((element) => {
    this.typeOptions.push(this.convertToOption(element));
  });
}

private getProjectInfo(projectId: number | undefined): void {
  if (!projectId) {
    return;
  }


  this.projectService.getProjectById(projectId)
    .pipe(takeUntil(this.unsubscribe$))
    .subscribe((resp) => {
      if (!resp.body) {
        return;
      }
      const project = resp.body;


      this.projectUsers = project.users;
      this.fillTypeOptions(project.projectTaskTypes);
      if (project.projectTaskPriorities) {
        this.fillPriorityOptions(project.projectTaskPriorities);
      }
      this.fillStateOptions(project.projectTaskStates);
    });


  this.sprintService
    .getProjectSprints(projectId)
    .pipe(takeUntil(this.unsubscribe$))
    .subscribe((sprints) => {
      if (!sprints.body) {
        return;
      }
      this.fillSprintOptions(sprints.body);
    });
}


public clearForm(): void {
  this.editTaskForm.reset(this.editTaskFormDefaultValues, {
    emitEvent: false,
  });
}

public submitForm(addUser?: boolean): void {
  if (this.editTaskForm.invalid) {
    this.notificationService.error(
      'Some values are incorrect. Follow error messages to solve this problem',
```

```
      'Invalid values',
    );
    return;
  }

  if (this.editTaskForm.pristine) {
    this.notificationService.error('Data hasn`t been changed');
    return;
  }

  const updatedTask = {
    ...this.task,
    users: this.editTaskForm.controls.assignees.value,
    description: this.editTaskForm.controls.description.value,
    summary: this.editTaskForm.controls.summary.value,
    priorityId: this.editTaskForm.controls.priority.value.id,
    stateId: this.editTaskForm.controls.status.value.id,
    typeId: this.editTaskForm.controls.type.value.id,
    projectId: this.editTaskForm.controls.project.value.id,
    sprintId:
      this.editTaskForm.controls.sprint.value.id !== 0
        ? this.editTaskForm.controls.sprint.value.id
        : undefined
  } as TaskUpdateModel;

  this.taskService
    .updateTask(updatedTask)
    .pipe(takeUntil(this.unsubscribe$))
    .subscribe((task) => {
      if (!addUser) {
        this.notificationService.success(
          'Task has been updated successfully',
        );
      }
      this.isChanged = new Observable<void>();
      if (task.body) {
        this.taskStorageService.updateTask(task.body);
        this.editTaskFormDefaultValues = this.editTaskForm.value;
        this.task = task.body;
      }
    })
    .add(() => {
      this.clearForm();
    });
  this.isChanging.emit(false);
}

toogleModal(event: boolean): void {
  this.isOpen = event;
  this.isChanging.emit(event);
  if (this.isOpen && this.task && !this.isDataLoaded) {
    this.projectService
      .getProjectsByOrganizationId(this.organizationId)
      .pipe(takeUntil(this.unsubscribe$))
      .subscribe((resp) => {
        if (!resp.body) {
          return;
        }
        this.fillProjectOptions(resp.body);
        this.getProjectInfo(this.task.projectId);
```

```
      });


    this.isDataLoaded = true;
   }
  }

  addUser(email: string): void {
   const isUserAdded = this.board.users.find((user) => user.email == email)
    ? true
    : false;


   if (isUserAdded) {
    this.notificationService.error(
     'User with given email has already been added',
    );
    return;
   }
   this.userService
    .getUserByEmail(email)
    .pipe(takeUntil(this.unsubscribe$))
    .subscribe((user) => {
     if (!user.body) {
      return;
     }
     this.editTaskForm.controls.assignees.value.push(user.body);
     this.board.users.push(this.convertToUserCard(user.body));
     this.editTaskForm.markAsDirty();
     this.submitForm(true);
     this.isChanged = new Observable<void>();
    });
  }


  deleteUser(email: string): void {
   const boardIndex = this.board.users.findIndex((x) => x.email === email);
   this.board.users.splice(boardIndex, 1);
   const index = this.editTaskForm.controls.assignees.value.findIndex(
    (x: UserModel) => x.email === email,
   );
   this.editTaskForm.controls.assignees.value.splice(index, 1);
   this.editTaskForm.markAsDirty();
   this.isChanged = new Observable<void>();
  }

  private convertToUserCard(user: UserModel): UserCardModel {
   return {
    id: user.id,
    email: user.email,
    userName: user.name,
    profileURL: '',
    role: null,
   };
  }
}
```

User.cs // модель користувача в БД

```
using Easyplan.Core.Common.Entities.Abstract;

namespace Easyplan.Core.Common.Entities;
```

```
public class User : BaseEntity
{
    public User()
    {
        OwnedProjects = new List<Project>();
        ParticipatedProjects = new List<Project>();
        OwnedTasks = new List<Task>();
        ParticipatedTasks = new List<Task>();
        Roles = new List<UserProjectRole>();
        SystemRoles = new List<UserOrganizationRole>();
        OwnedOrganization = new List<Organization>();
        ParticipatedOrganization = new List<Organization>();
    }
    public string Name { get; set; } = null!;
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
    public string Salt { get; set; } = null!;

    public virtual ICollection<Project> OwnedProjects { get; set; }
    public virtual ICollection<Project> ParticipatedProjects { get; set; }
    public virtual ICollection<Task> OwnedTasks { get; set; }
    public virtual ICollection<Task> ParticipatedTasks { get; set; }
    public virtual ICollection<UserProjectRole> Roles { get; set; }
    public virtual ICollection<UserOrganizationRole> SystemRoles { get; set; }
    public virtual ICollection<Organization> ParticipatedOrganization { get; set; }
    public virtual ICollection<Organization> OwnedOrganization { get; set; }

    public int? LastOrganizationId { get; set; }
    public Organization? LastOrganization { get; set; }
}
```

AuthController.cs // бекенд контролер для авторизації

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Easyplan.Core.Common.DTO;
using Easyplan.Core.Common.DTO.User;
using Easyplan.Core.Identity.Services;

namespace Easyplan.Core.WebAPI.Controllers
{
    [Route("api/user")]
    [AllowAnonymous]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly AuthService _service;

        public AuthController(AuthService service)
        {
            _service = service;
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] UserLoginDto loginInfo)
        {
            var user = await _service.Login(loginInfo);
            var token = _service.GetAccessToken(user.Id, user.Name, user.Email);
            return Ok(token);
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] UserRegisterDto registerInfo)
        {
```

```
            var user = await _service.Register(registerInfo);
            var token = _service.GetAccessToken(user.Id, user.Name, user.Email);
            return Ok(token);
        }
    }
}
```

organization-list.component.ts // сторінка список організацій

```
import { Component, OnInit } from '@angular/core';
import { OrganizationModel } from 'src/core/models/organization/organization-model';
import { UserModel } from 'src/core/models/user/user-model';
import {
  faMagnifyingGlass,
  faMessage,
} from '@fortawesome/free-solid-svg-icons';
import { GetCurrentUserService } from 'src/core/services/get-current-user.service';
import { BaseComponent } from 'src/core/base/base.component';
import { GetCurrentOrganizationService } from 'src/core/services/get-current-organization.service';
import { OpenDialogService } from 'src/core/services/open-dialog.service';
import { OrganizationService } from 'src/core/services/organization.service';

@Component({
  selector: 'app-organization-list',
  templateUrl: './organization-list.component.html',
  styleUrls: ['./organization-list.component.sass'],
})
export class OrganizationListComponent extends BaseComponent implements OnInit {
  public currentUser: UserModel;


  public items: OrganizationModel[] = [];


  public warningIcon = faMessage;
  public inputSearch = '';
  public itemsShow: OrganizationModel[] = [];
  public faMagnifyingGlass = faMagnifyingGlass;

  constructor(
    private currentUserService: GetCurrentUserService,
    private getCurrentOrganizationService: GetCurrentOrganizationService,
    private organizationService: OrganizationService,
    private openDialogService: OpenDialogService,
  ) {
    getCurrentOrganizationService.getLastOrganization();
    super();
  }

  ngOnInit(): void {
    this.currentUserService.currentUser$.subscribe((user) => {
      this.currentUser = user;
```

```
    this.organizationService.getUserOrganizations(this.currentUser.id).subscribe((resp) => {
      if(resp.ok){
        this.items = this.itemsShow = resp.body as OrganizationModel[];
      }
    });
  });


  this.getCurrentOrganizationService.organizations$.subscribe(
    (organizations) => {
      this.items = this.itemsShow = organizations;
    },
  );
}

filterItems(): void {
  if (this.inputSearch) {
    this.itemsShow = this.items.filter((item) => {
      return item.name.toLowerCase().includes(this.inputSearch.toLowerCase());
    });
  } else {
    this.itemsShow = this.items;
  }
}

openCreateOrganizationDialog(): void {
  this.openDialogService
    .openCreateOrganizationDialog(this.currentUser)
    .subscribe((result: OrganizationModel) => {
      if (!result) {
        return;
      }

      this.items.push(result);
      this.itemsShow = this.items;
      this.getCurrentOrganizationService.updateOrganization(result);
    });
  }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
	Пояснювальні документи
ПЗ_Ткаченко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
ПЗ_Ткаченко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
	Програма
Program.rar	Архів. Містить коди програми і скомпільовану програму
	Презентація
Презентація_Ткаченко.pptx	Презентація кваліфікаційної роботи