МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ГНУЧКОЇ СИСТЕМИ УПРАВЛІННЯ ПРОЄКТАМИ»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.П. Наумов
(ініціали та прізвище)
Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

« ____ » _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Наумову Дмитру Павловичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка гнучкої системи управління проєктами

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд існуючих методологій та засобів управління проєктами.
2. Проєктування власної системи управління проєктами.
3. Реалізація власної системи управління проєктами.
4. Рекомендації по використанню розробленої системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	13.03.2024	
2.	Збір вихідних даних.	21.03.2024	
3.	Обробка методичних та теоретичних джерел.	30.04.2024	
4.	Розробка першого та другого розділу.	24.06.2024	
5.	Розробка третього розділу.	30.09.2024	
6.	Розробка четвертого розділу.	28.10.2024	
7.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
8.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ Д.П. Наумов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка гнучкої системи управління проєктами»: 89 с., 43 рис., 5 табл., 30 джерел, 3 додатки.

EXPRESS, NODEJS, PROJECT MANAGEMENT, REALTIME COLLABORATION, WEBSOCKET, VUE.

Об’єкт дослідження – методологія Agile.

Мета роботи – розробити систему управління проєктами за методологією Agile з можливістю розширення функціоналу в форматі web-застосунку.

Методи дослідження – спостереження, аналіз, моделювання програмного забезпечення, конструювання програмного забезпечення.

У роботі проведено огляд існуючих методологій управління проєктами, розглянуто існуючі системи управління проєктами за методологією Agile, запропоновано власну систему, що має усунути недоліки та незручності існуючих систем і надати користувачам можливість розширювати функціонал системи.

Система реалізована у форматі web-застосунку з використанням мови JavaScript, технологій Express, Node.js, Objection та Vue. Взаємодія клієнта та сервера реалізована з використанням протоколів HTTP та WebSocket.

Отриманий в результаті роботи застосунок може бути використаний для управління проєктами за методологією Agile. Вихідний код застосунку та супровідна документація можуть бути використані при подальшій розробці систем управління проєктами.

SUMMARY

Master’s qualifying paper «Development of a Flexible Project Management System»: 89 pages, 43 figures, 5 tables, 30 references, 3 supplements.

EXPRESS, NODEJS, PROJECT MANAGEMENT, REALTIME COLLABORATION, WEBSOCKET, VUE.

The object of the study is the Agile methodology.

The aim of the study is to develop a project management system that implements Agile best practices and allows users to add custom features, in a web application format.

The methods of research are observation, analysis, software modeling and software construction.

This work provides an overview of existing project management methodologies and project management systems implementing Agile, provides a draft of own system that should solve inconveniences of existing systems and allow users to extend functionality.

The system was implemented in a web application format using JavaScript, Express, Node.js, Objection and Vue. Client and server communicate via HTTP and WebSocket protocols.

Software created within this study can be used to manage projects with Agile methodology. Source code and documentation can be used in further development of project management systems.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Постановка задачі 10
1.1 Методології управління проєктами 10
1.2 Існуючі системи управління проєктами 15
1.3 Вимоги до власної системи управління проєктами 21
1.3.1 Нефункціональні вимоги 21
1.3.2 Функціональні вимоги 21
1.4 Висновки до розділу 1 22
2 Проєктування системи 23
2.1 Варіанти використання системи 23
2.2 Модель даних 25
2.3 Мережева взаємодія 28
2.4 Користувацький інтерфейс 31
2.5 Висновки до розділу 2 31
3 Програмна реалізація системи 32
3.1 Перелік використаних технологій 32
3.2 Реалізація бази даних 34
3.3 Реалізація серверної частини 37
3.3.1 Реалізація взаємодії по проколу HTTP 37
3.3.2 Реалізація взаємодії по протоколу WebSocket 38
3.4 Реалізація клієнтської частини 41
3.5 Висновки до розділу 3 43
4 Використання системи 44
4.1 Розгортання системи 44

4.2 Робота з користувацьким інтерфейсом 45
4.3 Розширення користувацького інтерфейсу 50
4.4 Висновки до розділу 4 52
Висновки 53
Перелік посилань 54
Додаток А Приклади вихідного коду сервера 57
Додаток Б Приклади вихідного коду клієнта 71
Додаток В Розширення функціоналу системи 84

ВСТУП

Чи актуально створювати систему управління проєктами у 2024 році? Усім відомо, що інформаційні технології швидко розвиваються, з'являються певні новинки, тренди, як-от блокчейн або машинне навчання. Деякі напрями, як-от розробка операційних систем, мов програмування чи офісних застосунків, перебувають в стагнації через те, що предметна область була освоєна та доступно багато рішень, їх кількість перевищує попит.

Попит на певні види прикладного ПЗ є мінливим, але в будь-який момент часу існують програмні продукти у стані активної розробки або підтримки. Поки люди задіяні у розробці програмних продуктів, для забезпечення ефективного процесу розробки потрібні ефективні інструменти, що дозволять вести облік задач, планувати та відстежувати їх виконання.

Отже, засоби управління проєктами є однією з мета-галузей ринку програмних продуктів, що завжди матимуть високий попит, оскільки цей попит залежить від сумарної кількості активних програмних продуктів будь-якої галузі.

Існуючі засоби управління проєктами, як-от Atlassian Jira, GitHub Projects та інші, є досить багатофункціональними, але мають певні недоліки. В цій роботі пропонується власне бачення з точки зору користувача щодо реалізації системи управління проєктами.

Мета роботи – розробити систему управління проєктами за методологією Agile з можливістю розширення функціоналу в форматі web-застосунку.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- дослідити існуючі системи управління проєктами та сформулювати вимоги до власної системи;
- спроєктувати архітектуру системи;
- реалізувати серверну частину застосунку;

– реалізувати клієнтську частину застосунку;

– розробити керівництво користувача.

Об’єкт дослідження – методологія Agile.

Предмет дослідження – реалізація методології у форматі web-застосунку.

Методи дослідження – спостереження, аналіз, моделювання програмного забезпечення, конструювання програмного забезпечення.

Робота складається зі вступу, чотирьох розділів, висновків, переліку посилань та трьох додатків. У переліку посилань наведено 30 використаних джерел.

У першому розділі розглянуто існуючі методології управління проєктами, в тому числі методологія Agile, розглянуто існуючі системи управління проєктами, проведено порівняння функціоналу існуючих систем, сформовано вимоги до власної системи управління проєктами.

У другому розділі виконано проєктування системи управління проєктами, наведено модель варіантів використання, модель даних, модель мережевої взаємодії, модель сторінок клієнтського застосунку.

У третьому розділі описано реалізацію бази даних, серверної та клієнтської частини системи управління проєктами.

У четвертому розділі наведено рекомендації щодо розгортання системи, використання основного функціоналу клієнтського застосунку та створення розширень для клієнтського застосунку.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Методології управління проєктами

Для початку варто визначити поняття «проєкт» та «управління проєктами». В класичному розумінні проєкт – це певна послідовність дій з такими характеристиками:

– виконавці;
– ресурсні обмеження;
– часові обмеження;
– унікальність результату [1].

Управління проєктом – це застосування певних знань, умінь, практик, інструментів до проєкту для досягнення поставлених цілей [1].

Проєкти можуть бути у багатьох галузях, у тому числі в програмній інженерії. Відомо, що проєкти програмного забезпечення мають такий життєвий цикл у загальному вигляді:

– підготовка;
– виробництво;
– завершення.

Існує чимало варіацій життєвого циклу проєктів та їх окремих частин. Відомо, що до етапу підготовки можна включити збір вимог та аналіз вимог. Етап проєктування (побудова діаграм, макетів інтерфейсу тощо) є посереднім між підготовкою та виробництвом. Етап виробництва включає програмну реалізацію, тестування, виправлення дефектів. Етап завершення включає активну експлуатацію розробленого продукту та виведення з експлуатації.

Водоспадна модель (Waterfall) є однією з найбільш відомих класичних методологій управління проєктами програмного забезпечення. Тут життєвий цикл всього проєкту можна поділити на етапи з різними назвами, зазвичай це такі етапи, як збір вимог, проєктування, програмна реалізація, тестування,

впровадження (див. рис. 1.1). Зазвичай наступний етап не може починатися до завершення попереднього етапу або його значної частини [2]. Перевагою такого підходу є передбачуваність та наявність якісного підґрунтя при переході на наступний етап. Недоліками такого підходу є неефективне використання часу, складність упровадження змін, ризик втрати актуальності.

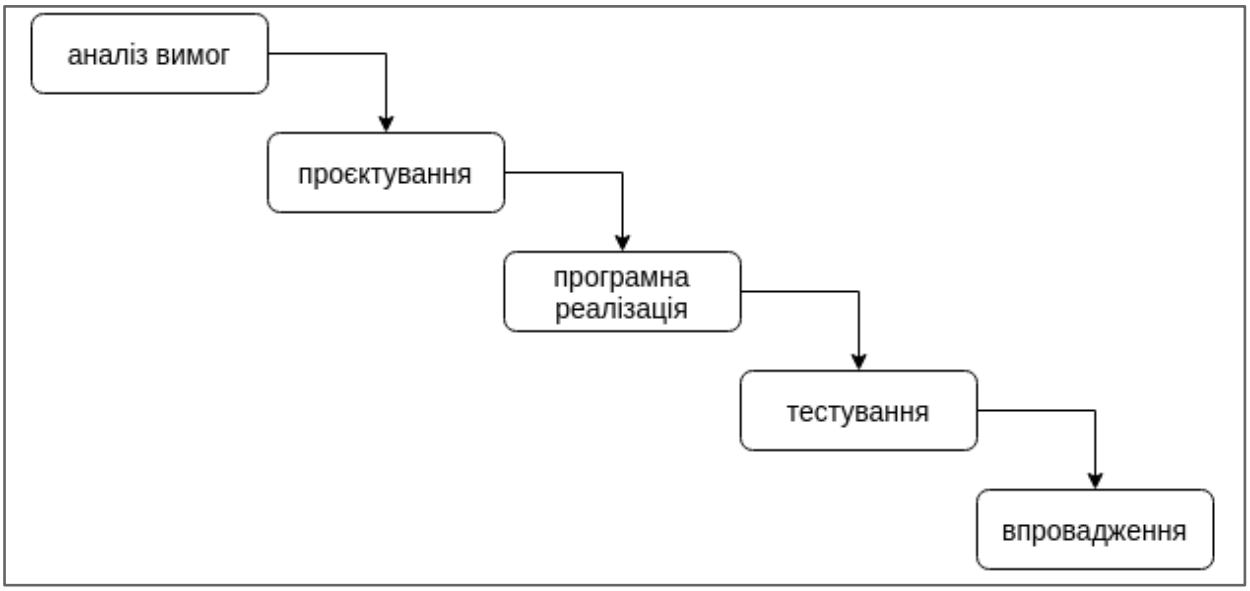


Рисунок 1.1 – Водоспадна модель

Rational Unified Process є більш прогресивною методологією в порівнянні з Waterfall. Ця методологія поділяє життєвий цикл проєкту на такі етапи: підготовка (inception), обговорення (elaboration), конструювання (construction), перехід (transition). На відміну від водоспадної методології, в одному етапі можна поєднувати різні види робіт, як-от збір вимог, проєктування, програмна реалізація, тестування, впровадження, тому перехід між етапами здійснюється більш плавно (див. рис. 1.2) [2].

Спіральна модель (Spiral) передбачає поділ життєвого циклу проєкту на ітерації приблизно однакової тривалості (див. рис. 1.3). Кожна ітерація відповідно поділяється на постановку задачі, оцінку ризиків, програмування, оцінку якості, планування наступного етапу. При правильному застосуванні така методологія дозволяє більш плавно переходити між рівнями готовності проєкту та отримати працюючий прототип за короткий проміжок часу [3].

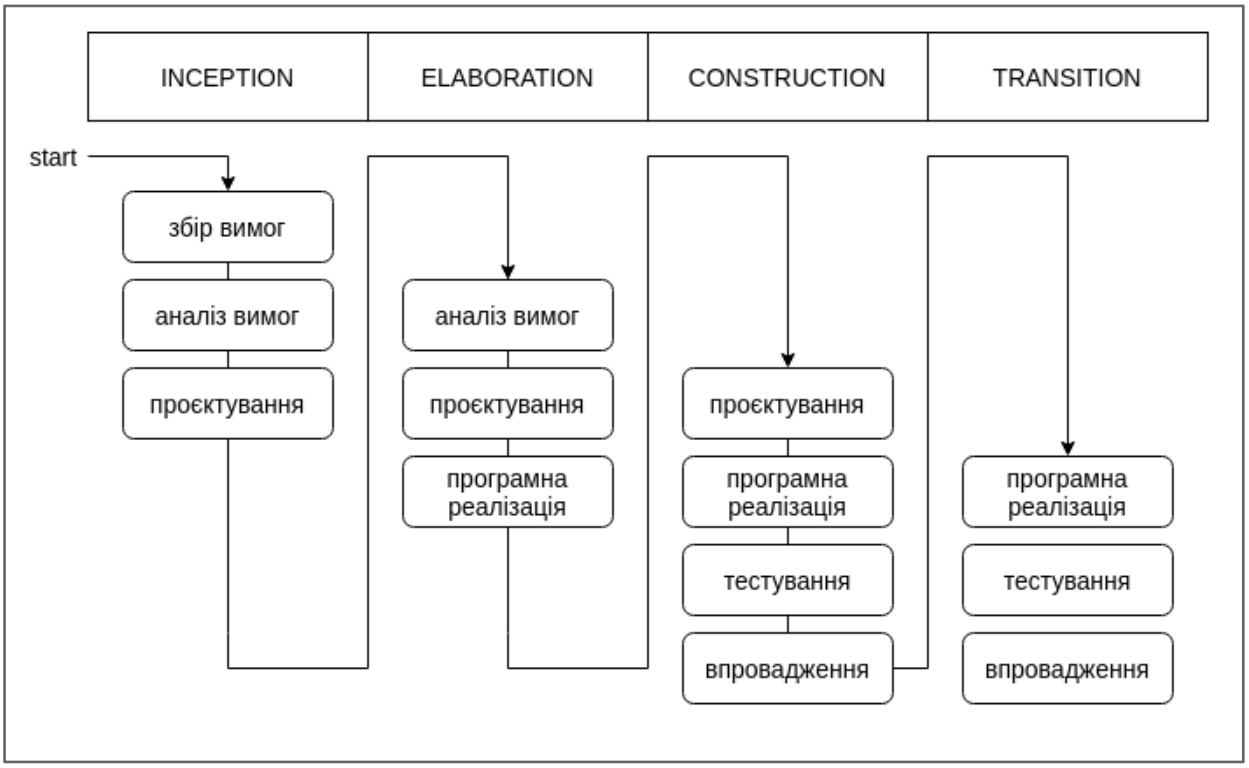


Рисунок 1.2 – Rational Unified Process

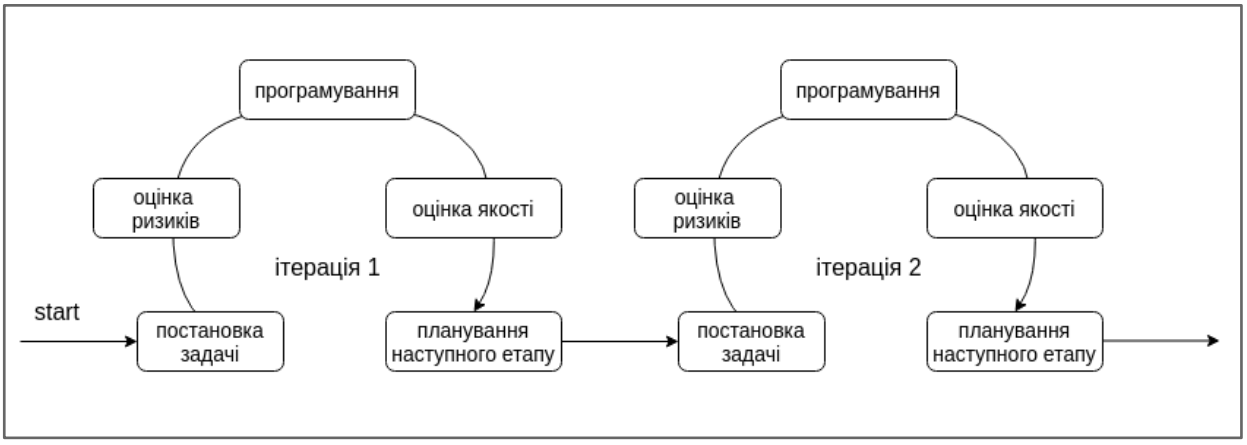


Рисунок 1.3 – Спіральна модель

Agile – це набір практик, що найчастіше використовується саме при розробці програмного забезпечення. Згідно маніфесту Agile, найвищими цінностями є:

– робоче програмне забезпечення;
– активна взаємодія розробників та замовників;
– самоорганізація команди;
– створення оптимального середовища для роботи [4].

Ця методологія включає в себе такі практики, що більшою або меншою мірою використовуються в різних проєктах:

– використання дошки задач (Kanban board);

– регулярні обговорення у форматі Scrum;

– екстремальне програмування (XP, eXtreme Programming) та ін.

Kanban – це фізична або віртуальна дошка з картками – описом задач (див. рис. 1.4). Зазвичай дошка поділяється на стовпці за статусом задачі. Метою Kanban є створити максимально зручне та зрозуміле представлення задач та їх статусу виконання [5].

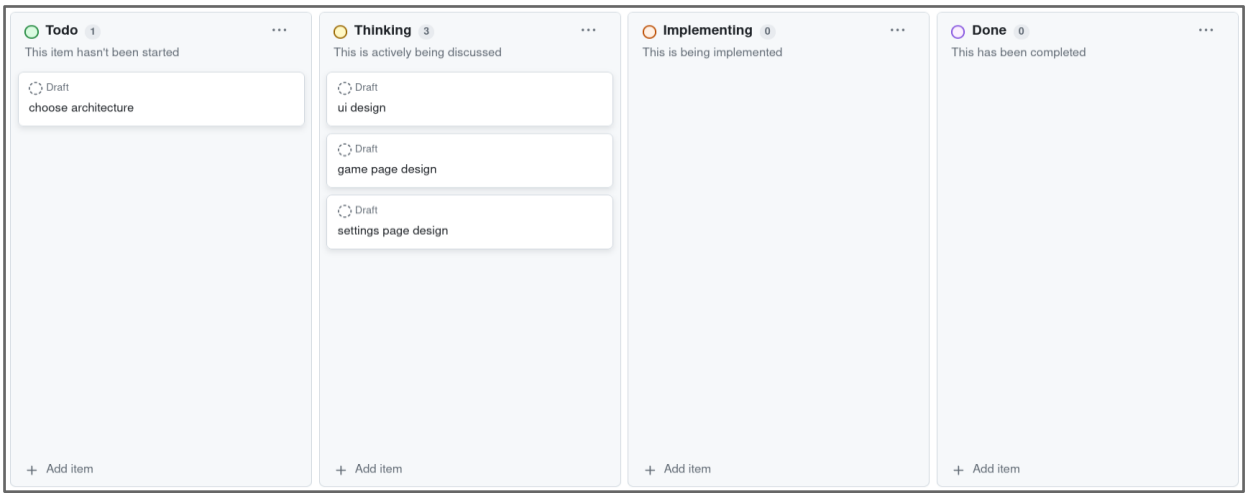


Рисунок 1.4 – Приклад дошки задач в GitHub

Scrum (див. рис. 1.5) – це підхід до управління проєктом, що передбачає поділ життєвого циклу на ітерації (спринти). Метою ітерації може бути розробка нового смислового блоку (функціоналу) продукту або виправлення помилок. Тривалість спринта зазвичай становить від одного тижня до одного місяця. До початку спринта команда виконавців обирає задачі з беклогу які мають бути виконані, оцінює тривалість виконання та можливі ризики. Упродовж спринта зазвичай проводяться короткі щоденні зустрічі для з'ясування стану проєкту та можливих проблем. Після спринта зазвичай проводиться ретроспективна зустріч для оцінки продуктивності команди, відзначення позитивного та негативного досвіду, формування рекомендацій для майбутніх спринтів [6].

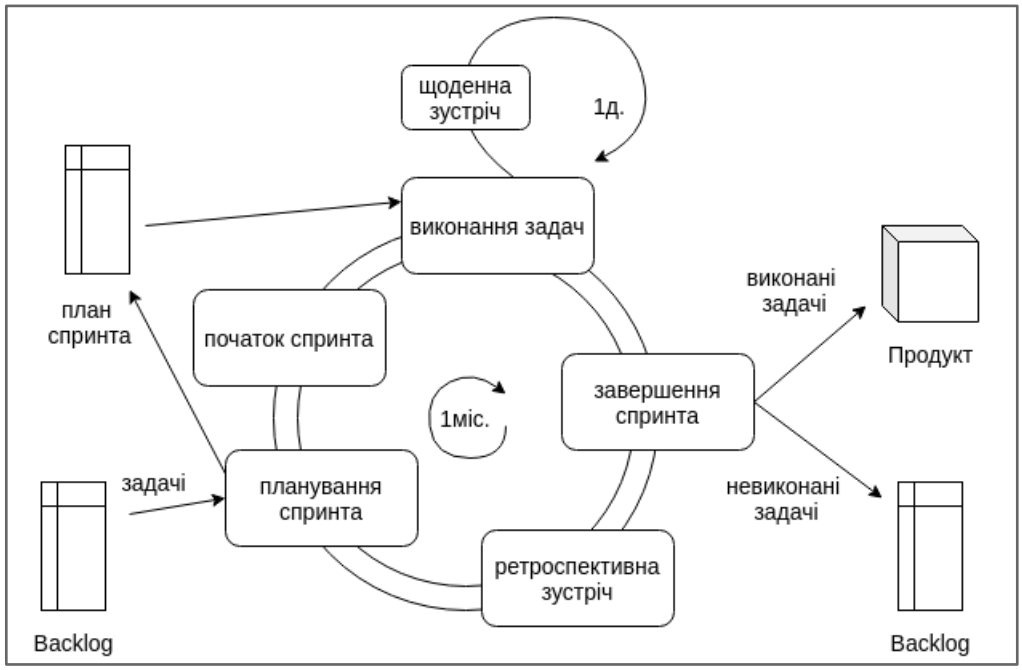


Рисунок 1.5 – Scrum

Екстремальне програмування (див. рис. 1.6) – це підхід до планування та виконання проєкту, що передбачає:

– поділ проєкту на ітерації тривалістю від 1 до 3 тижнів;
– планування в межах однієї ітерації, щоб поставити реалістичні та досяжні цілі;
– безперервну взаємодію з замовником;
– парне програмування для підвищення ефективності праці та обміну досвідом;
– розробку на основі тестів (test driven development);
– максимально просту програмну реалізацію;
– регулярну публікацію нових версій ПЗ;
– підтримку стабільної швидкості розробки та усунення перешкод для стабільності як тільки вони з’являються [7].

В цілому, Agile дозволяє перейти від фіксованих довгострокових планів, що можуть втратити актуальність, до більш ситуативного короткострокового планування з акцентом на поточний стан справ, актуальні вимоги та задачі. Для підтримки робочого середовища Agile передбачає використання різних автоматизованих засобів управління проєктами.

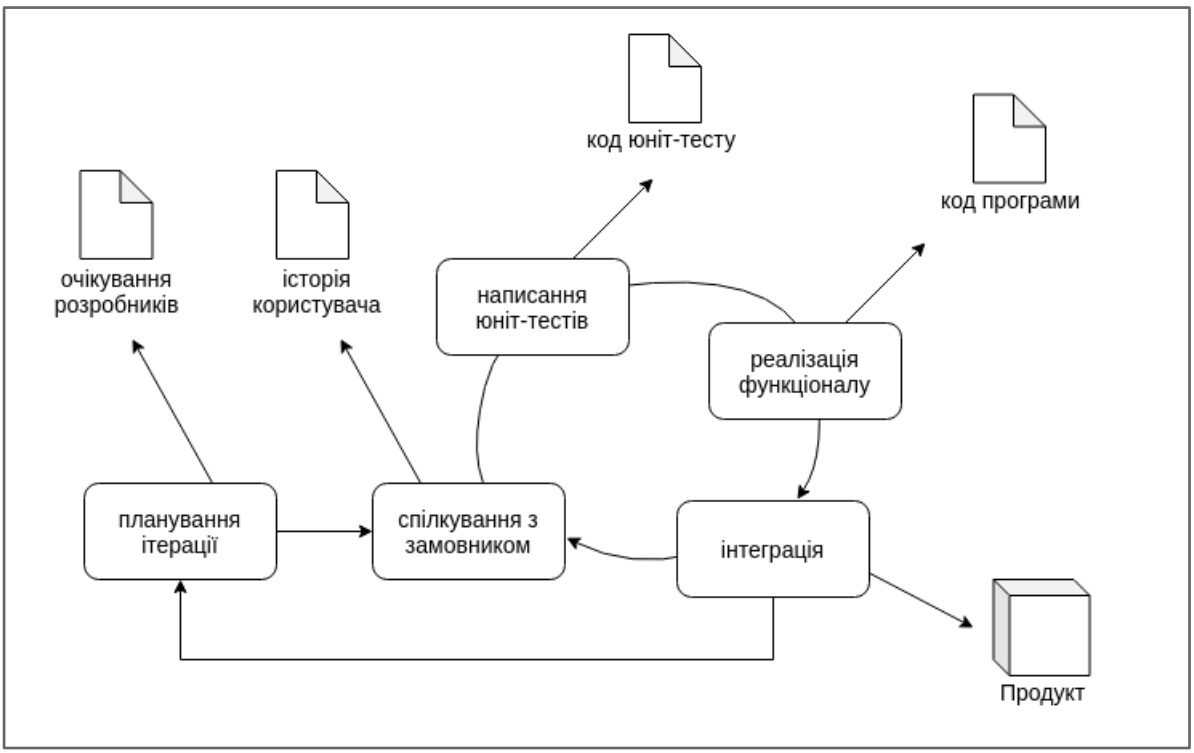


Рисунок 1.6 – Екстремальне програмування

1.2 Існуючі системи управління проєктами

Станом на 2024 рік існує багато систем для управління проєктами, кожна з яких має особливий набір функціональних можливостей та особливий підхід до такого питання: «як вести облік проєкту?».

GitLab [8] має кодо-орієнтований підхід. При створенні нового проєкту автоматично створюється репозиторій для вихідного коду та пустий список задач. У поточній версії, окрім інструментів для роботи з вихідним кодом, є меню «Plan» з пунктами «Issues», «Issue Boards» та «Milestones». На сторінці «Issues» можна переглянути існуючі задачі та створити нову задачу. Можна додавати вкладені (child) або пов’язані (linked) задачі. На сторінці «Issue Boards» можна переглянути дошки задач (див. рис. 1.7). Декілька активник дошок можуть існувати одночасно, але задачі також перебувають на всіх дошках одночасно. Є 2 фіксованих статуси задач – «Open» та «Closed». До дошки задач можна додати стовпець для задач з певною міткою. Можна

зробити припущення, що тут дошки призначені для фільтрування задач. Також можна фільтрувати задачі за значенням деякого поля. Також задачі можна додавати у персональний список «To-Do List». На сторінці «Milestones» можна переглянути існуючі віхи та створити нову віху, до віхи можна прив'язати існуючі задачі. Можна додати учасників до проєкту. Окрім цього, у преміум-версії системи доступні «Burndown Charts» (діаграми згорання) та інтеграція з Jira. Недоліком GitLab є відсутність представлення задач у часі.

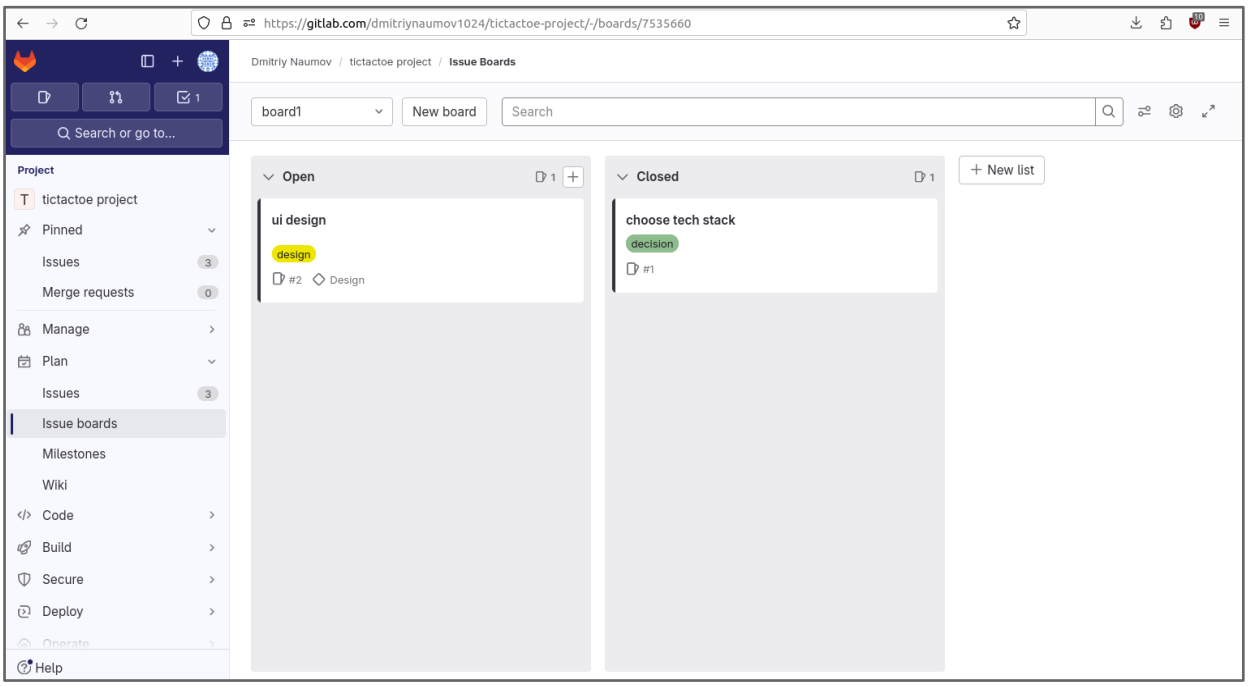


Рисунок 1.7 – GitLab (дошка задач)

GitHub [9] у свою чергу дозволяє відокремити проєкти від вихідного коду. До проєкту можна додавати довільну кількість візуальних представлень типу «Board», «Table», «Roadmap» (див. рис. 1.8). Сторінка «Board» є типовою дошкою задач, тут є можливість без обмежень редагувати існуючі стовпці та додавати нові. Стовпець відповідає статусу задачі. Можна додавати власні поля даних до задачі. У «Table» можна переглянути список задач, додавати нові задачі, групувати та фільтрувати за значенням деякого поля. У «Roadmap» можна переглянути та змінити розміщення задач у часі. GitHub не надає можливості створювати вкладені задачі. Можна створювати віхи за умови, що

проєкт прив’язаний до репозиторію з кодом. Є можливість переглядати гістограми деяких сумарних значень залежно від статусу задачі в поточний момент. Недоліком GitHub є відсутність пов’язаних та вкладених задач, відсутність часових діаграм, як-от «Burndown Charts».

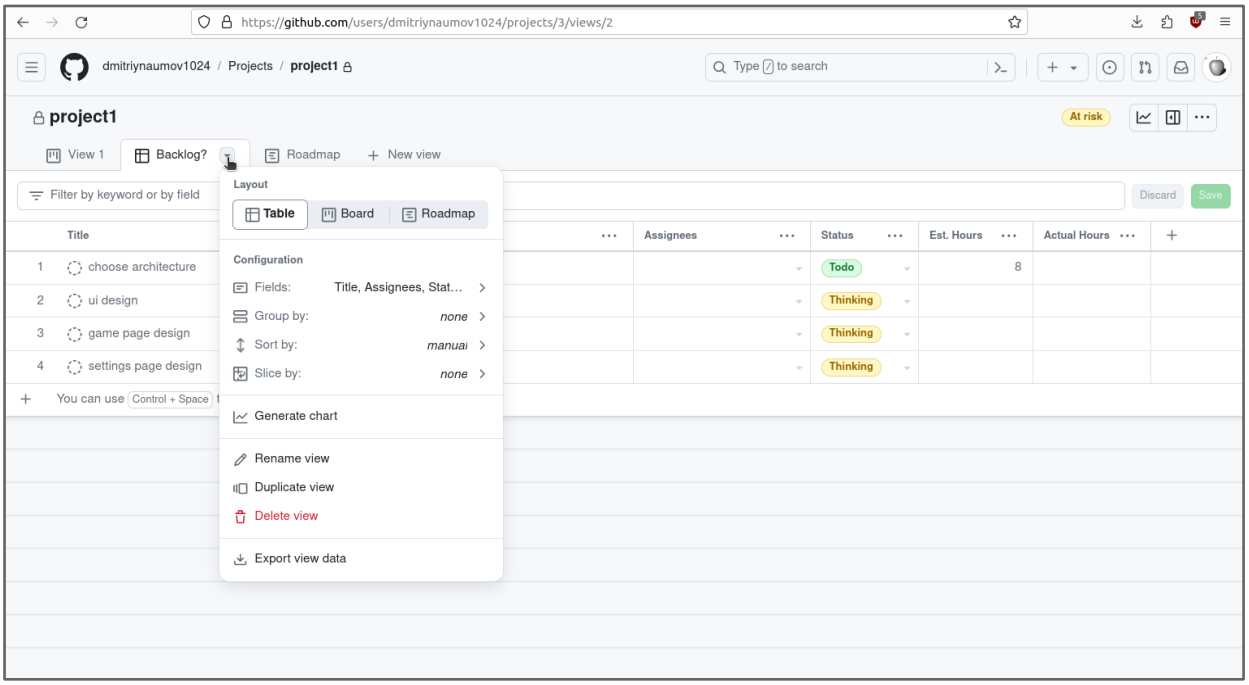


Рисунок 1.8 – GitHub (список задач)

Jira [10] є окремим сервісом, що спеціалізується на проєктах різного типу. Тут можна створити, наприклад, бізнес-проєкт або Agile-проєкт. Спільною рисою для різних типів проєктів є дошка задач. Можна створювати історії (Story), задачі (Issue) та вкладені задачі (Subtask). До дошки можна додавати власні стовпці зі статусами задач. Для бізнес-проєкту є сторінка «Timeline» що дозволяє переглядати положення задач у часі. Для Agile-проєкту є сторінка «Backlog», де відбувається створення задач та спринтів – наборів задач, що підлягають виконанню у визначений проміжок часу. Для Agile-проєкту дошка задач містить задачі поточного спринта. Можна додавати власні поля до задач, здійснювати пошук та групування по значенням полів. Є чимало корисних звітів, як-от Burndown Charts – зміна кількості виконаних задач з плином часу (див. рис. 1.9), Velocity – кількість виконаної роботи у спринтах, Cumulative

Flow – зміна кількості задач кожного статусу з плином часу. Недоліком Jira є надмірне ускладнення, користувачу доведеться витратити певний час на ознайомлення з сервісом.

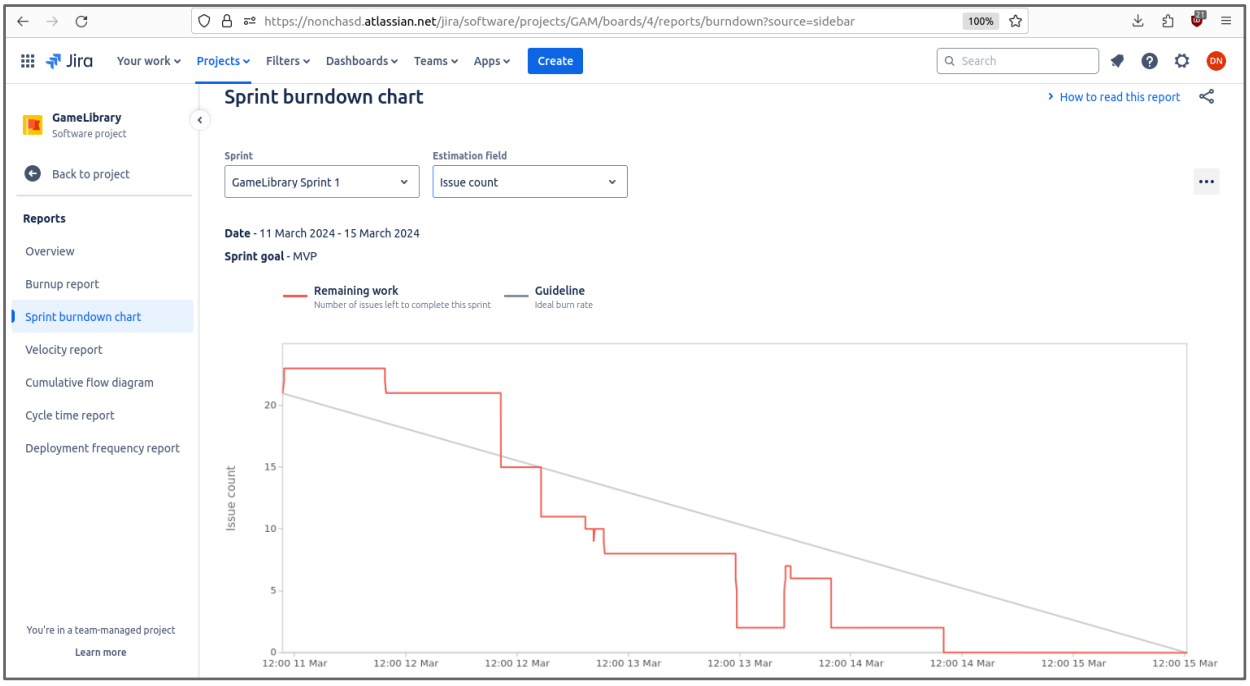


Рисунок 1.9 – Jira (Burndown chart)

Trello [11] є відносно мінімалістичним інструментом для управління проєктами, що орієнтується на роботу з дошкою задач (див. рис. 1.10). Так само як в інших системах, декілька учасників можуть працювати над одним проєктом. До дошки задач можна додавати власні стовпці. У преміум-версії доступні такі функції: таблиця задач, календар, власні поля даних.

ClickUp [12] є комплексним інструментом. Так само як у раніше розглянутих системах, можна створювати проєкти та додавати задачі, вкладені задачі та віхи. До задач можна додавати коментарі та власні поля даних. Можна переглядати проєкт у різних візуальних представленнях, як-от список, дошка задач, хронологія. Особливістю дошки задач в цій системі є гнучкість: за замовченням стовпці відповідають статусу задачі, але також можна згрупувати у стовпці за виконавцем, мітками, пріоритетом чи власним полем даних (див. рис. 1.11). Також тут є добре розвинена підсистема відстеження робочого часу.

Окрім цього, у преміум-версії доступні діаграми та звіти, як-от Burndown Charts чи Cumulative Flow.

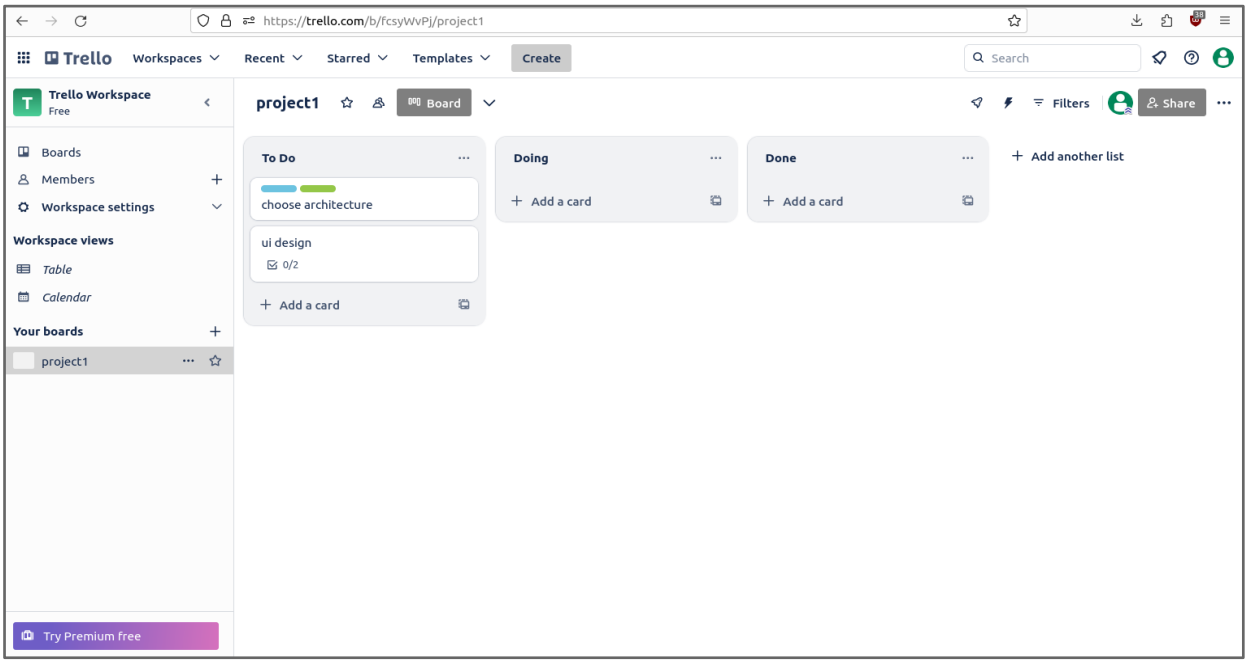


Рисунок 1.10 – Trello (дошка задач)

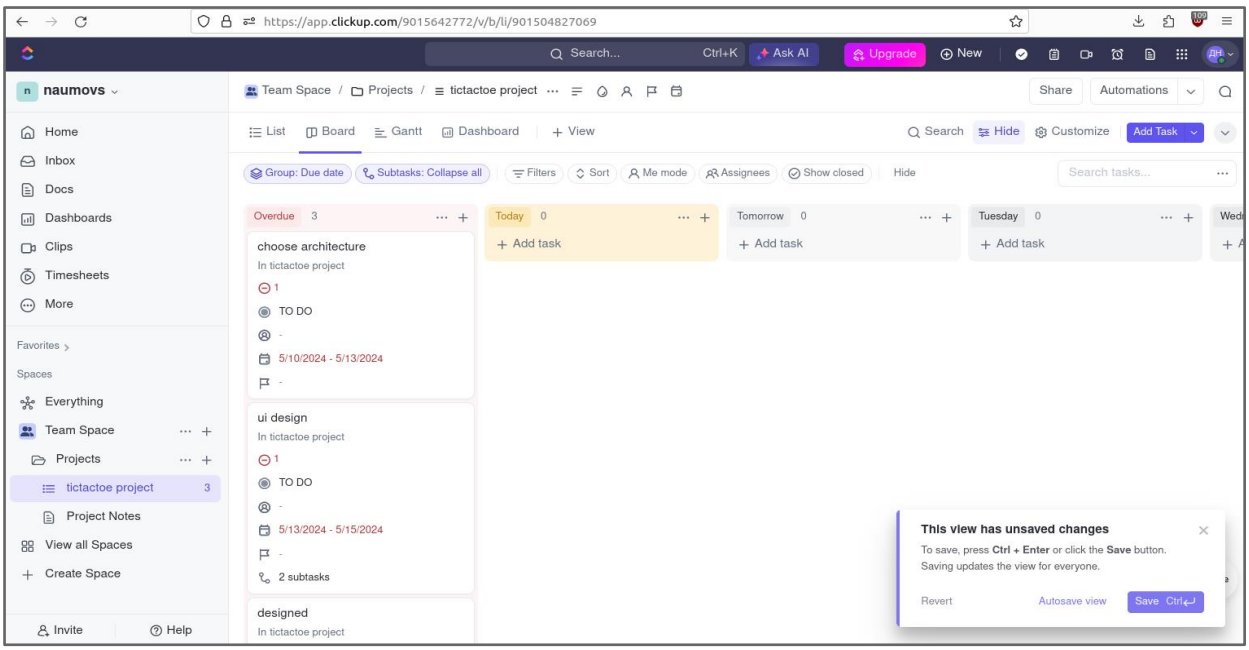


Рисунок 1.11 – ClickUp (дошка задач)

Розглянуті вище системи є вебзастосунками, що надають спільний доступ до проєктів у реальному часі. У таблиці 1.1 наведено порівняльну характеристику систем за наявністю функціоналу, як-от список/таблиця задач,

дошка задач, хронологія, спринт, віхи, діаграми, звіти, власні статуси задач, власні поля задач, пов’язані задачі, фільтрування та групування, оцінка часу, відстеження робочого часу.

Таблиця 1.1 – Функціонал деяких систем управління проєктами

№ в/п	**Назва функціоналу**	**GitLab**	**GitHub**	**Jira (business)**	**Jira (agile)**
1	Список/таблиця задач	+	+	+	+
2	Дошка задач	+	+	+	+
3	Хронологія	–	+	+	+
4	Спринт	–	–	–	+
5	Віхи	+	* (репозиторій)	–	–
6	Діаграми, звіти	* (premium)	* (не корисні)	+	+
7	Власні статуси задач	* (обмеж.)	+	+	+
8	Власні поля задач	–	+	+	+
9	Пов’язані задачі	+	–	+	+
10	Фільтрування задач	+	+	+	+
11	Групування задач	–	+	* (обмеж.)	* (обмеж.)
12	Оцінка часу	+	* (обмеж.)	* (обмеж.)	+
13	Відстеження роб.часу	–	–	–	* (обмеж.)

Продовження таблиці 1.1

№ в/п	**Назва функціоналу**	**Trello**	**ClickUp**
1	Список/таблиця задач	* (premium)	+
2	Дошка задач	+	+
3	Хронологія	* (premium)	+
4	Спринт	–	–
5	Віхи	–	+
6	Діаграми, звіти	–	* (premium)
7	Власні статуси задач	+	+
8	Власні поля задач	* (premium)	+
9	Пов’язані задачі	–	+
10	Фільтрування задач	+	+
11	Групування задач	–	+
12	Оцінка часу	–	+
13	Відстеження роб.часу	–	+

З порівняльної таблиці 1.1 можна побачити, що найкращими системами для управління проєктами є Jira (agile) та ClickUp. Для отримання найвищої оцінки системі Jira не вистачає віх та відстежування робочого часу, а системі ClickUp не вистачає спринтів.

1.3 Вимоги до власної системи управління проєктами

1.3.1 Нефункціональні вимоги

Так само як для інших вебзастосунків, тут можна сформулювати такі вимоги:

– інформативність;
– надійність;
– зрозумілий інтерфейс;
– швидкість завантаження;
– швидкість відгуку на дію користувача.

Окрім цього, додаються такі нефункціональні вимоги для системи управління проєктами:

– можливість розширення, оскільки неможливо задовольнити потреби всіх користувачів, але можна розробити певний API для розширення та дати користувачам можливість робити плагіни для цього API й використовувати їх у своїх проєктах;
– спільний доступ до проєкту, оскільки всім відомо, що значна частина проєктів виконується командами розробників;
– відображення змін у реальному часі, оскільки декілька користувачів можуть редагувати проєкт одночасно, і необхідно повідомляти інших користувачів про зміни без ініціації з боку користувача.

1.3.2 Функціональні вимоги

На основі розглянутих у п.1.1 засобів управління проєктами, можна виділити такі функціональні вимоги:

– створювати, редагувати, переглядати обліковий запис;
– створювати, редагувати, переглядати, видаляти проєкти;

– додавати, вилучати учасників проєкту;
– встановлювати рівень доступу учасників проєкту;
– змінювати власника проєкту;
– створювати, редагувати, переглядати, видаляти плагіни;
– додавати, редагувати, переглядати плагіни.

Функціональні вимоги безпосередньо проєкту:

– додавати, вилучати плагіни;
– створювати, редагувати, переглядати, видаляти задачі;
– призначати виконавців задач;
– обговорення задачі;
– вкладені, пов’язані задачі;
– додавання власних полів даних до задач;
– створювати задачі за шаблонами;
– відстежувати виконання робіт;
– переглядати журнал подій;
– переглядати історію змін проєкту.

Історія змін проєкту є важливою тому, що декілька людей можуть редагувати проєкт одночасно, використання блокуючих механізмів є витратним та незручним, але задачі є важливою частиною проєкту та випадкова втрата опису задачі може привести до значних збитків.

1.4 Висновки до розділу 1

Тут розглянуто методології управління проєктами програмного забезпечення та зазначено, що методологія Agile займає важливе місце серед інших методологій. Існує чимало систем управління проєктами, в яких певною мірою реалізовані найкращі практики методології Agile у форматі web-застосунків. На основі розглянутих систем сформовано вимоги до власної системи управління проєктами.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Варіанти використання системи

Передбачається, що всі користувачі розроблюваної системи управління проєктами будуть належати до одного класу. Єдиним актором в моделі варіантів використання буде «користувач».

На основі п.1.3 сформуємо варіанти використання системи. Можна виділити такі групи варіантів використання:

– управління обліковим записом, до якого належить створення, перегляд, редагування облікового запису, вхід в обл. запис та вихід з обл. запису;
– адміністрування плагінів, що включає в себе створення, перегляд та редагування плагінів;
– адміністрування проєкту, що включає: створення проєкту, перегляд списку проєктів, видалення проєкту, перегляд використаних плагінів, додавання та вилучення плагінів, перегляд учасників проєкту, додавання та вилучення учасників проєкту;
– огляд вмісту проєкту в реальному часі, що включає: перегляд учасників проєкту, перегляд задач, перегляд шаблонів задач, перегляд коментарів до задач, перегляд записів про виконання задач, перегляд звітів, перегляд журналу змін, перегляд журналу подій;
– редагування проєкту в реальному часі, що включає: опис проєкту, додавання власних полів задач, додавання задач, додавання коментарів до задачі, додавання запису про виконання задачі, додавання шаблону задачі, створення задачі за шаблоном.

Графічне представлення варіантів використання наведено на рисунках 2.1 – 2.2.

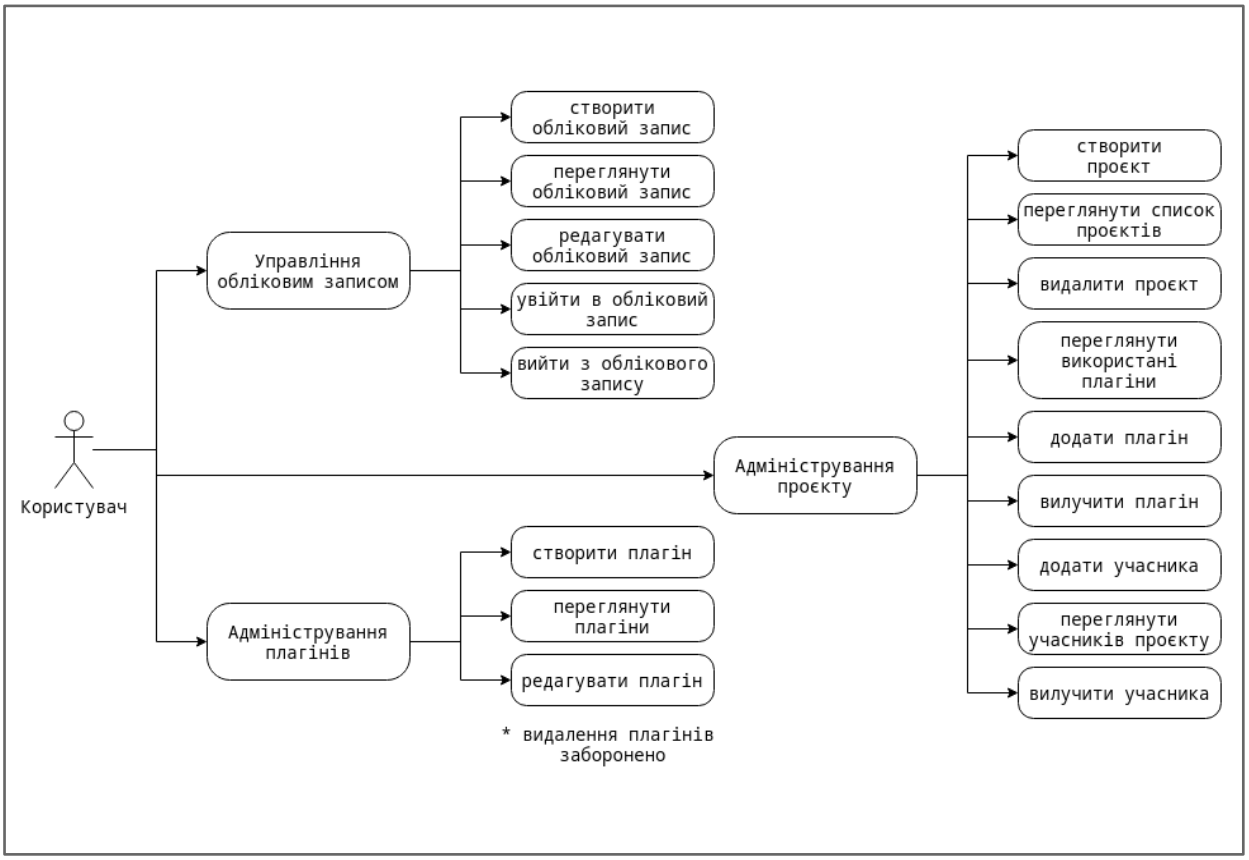


Рисунок 2.1 – Варіанти використання системи

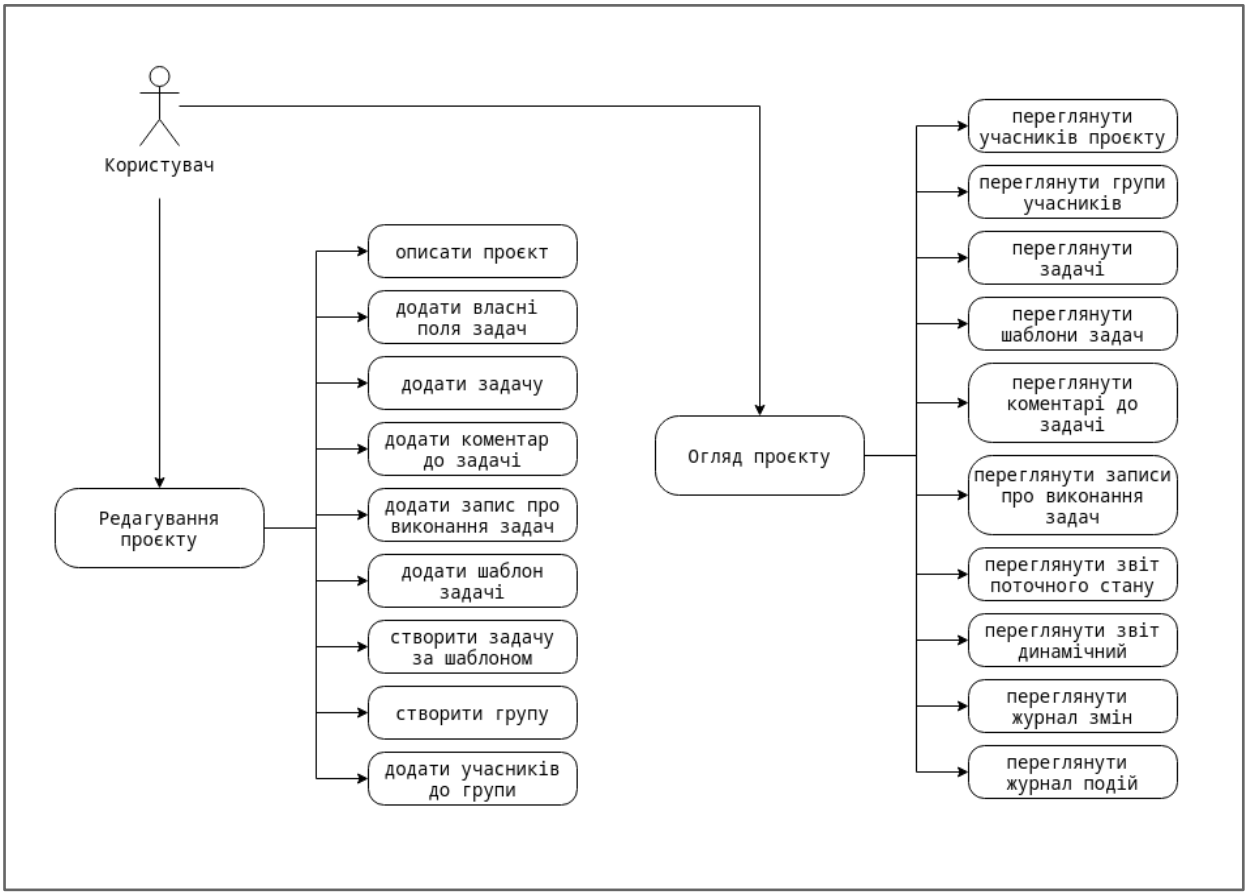


Рисунок 2.2 – Варіанти використання системи

2.2 Модель даних

Реляційна схема даних наведена на рисунку 2.3.

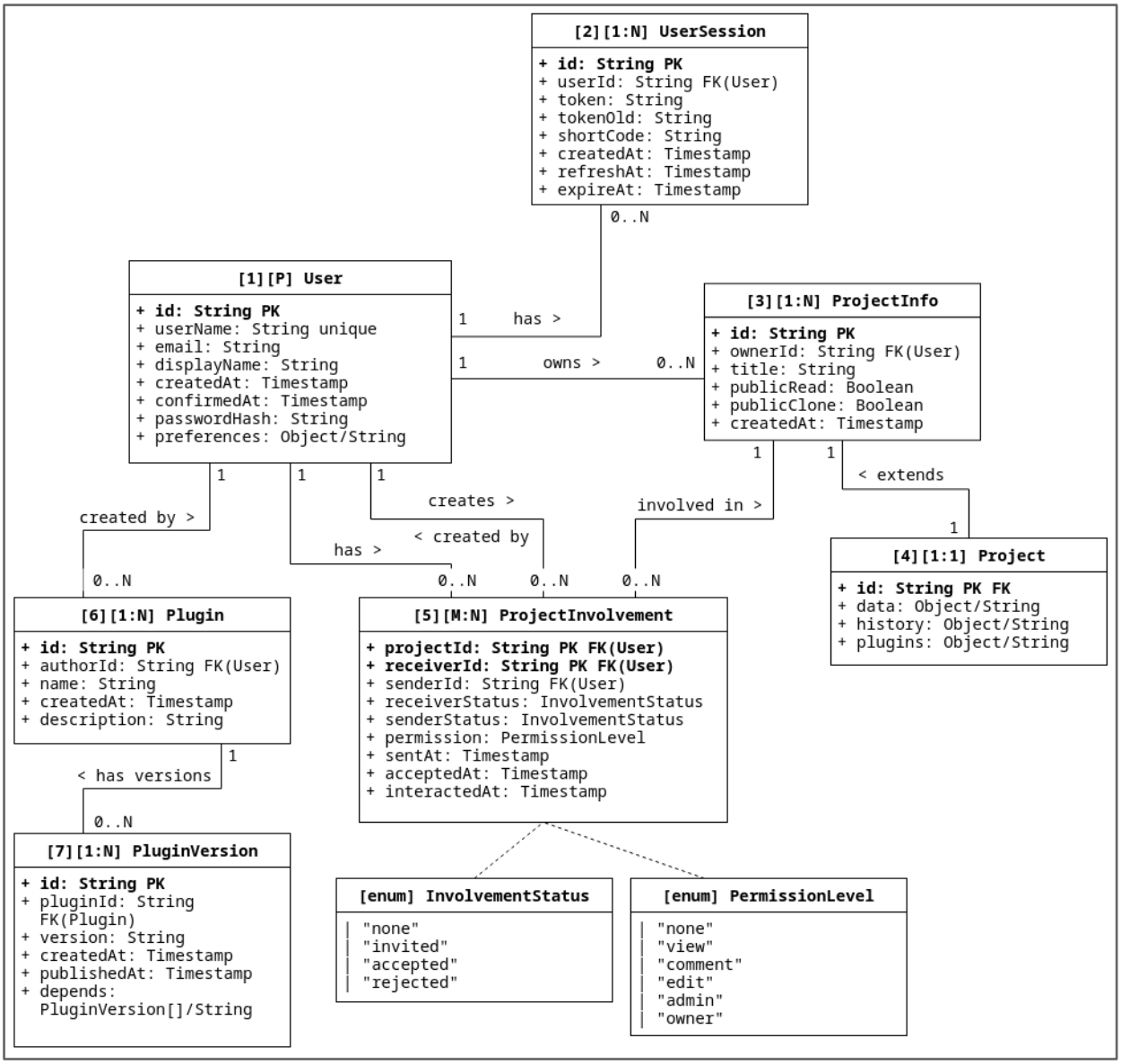


Рисунок 2.3 – Реляційна модель даних системи управління проєктами

У системі управління проєктами можна виділити такий набір сутностей:

– User – містить типову інформацію про користувача, як-от логін, електронна адреса, власне ім’я, дата реєстрації, пароль (хеш);

– Project – містить інформацію про проєкт, як-от id власника, назва проєкту, дата створення, опис проєкту, задачі, історія змін, використані плагіни;

– ProjectInvolvement – містить інформацію про учась у проєкті, що включає id проєкту, id учасника, id користувача, що сформував запрошення, дату надсилання запрошення, дату підтвердження, рівень доступу, статус запрошення;
– Plugin – містить інформацію про плагін (розширення функціоналу проєкту), що включає id автора, назву плагіна, версію, дату створення, перелік залежностей, вихідний код.
– PluginVersion – містить інформацію про версію плагіна.

Документна модель даних для внутрішньої структури проєкту наведена на рисунку 2.4.

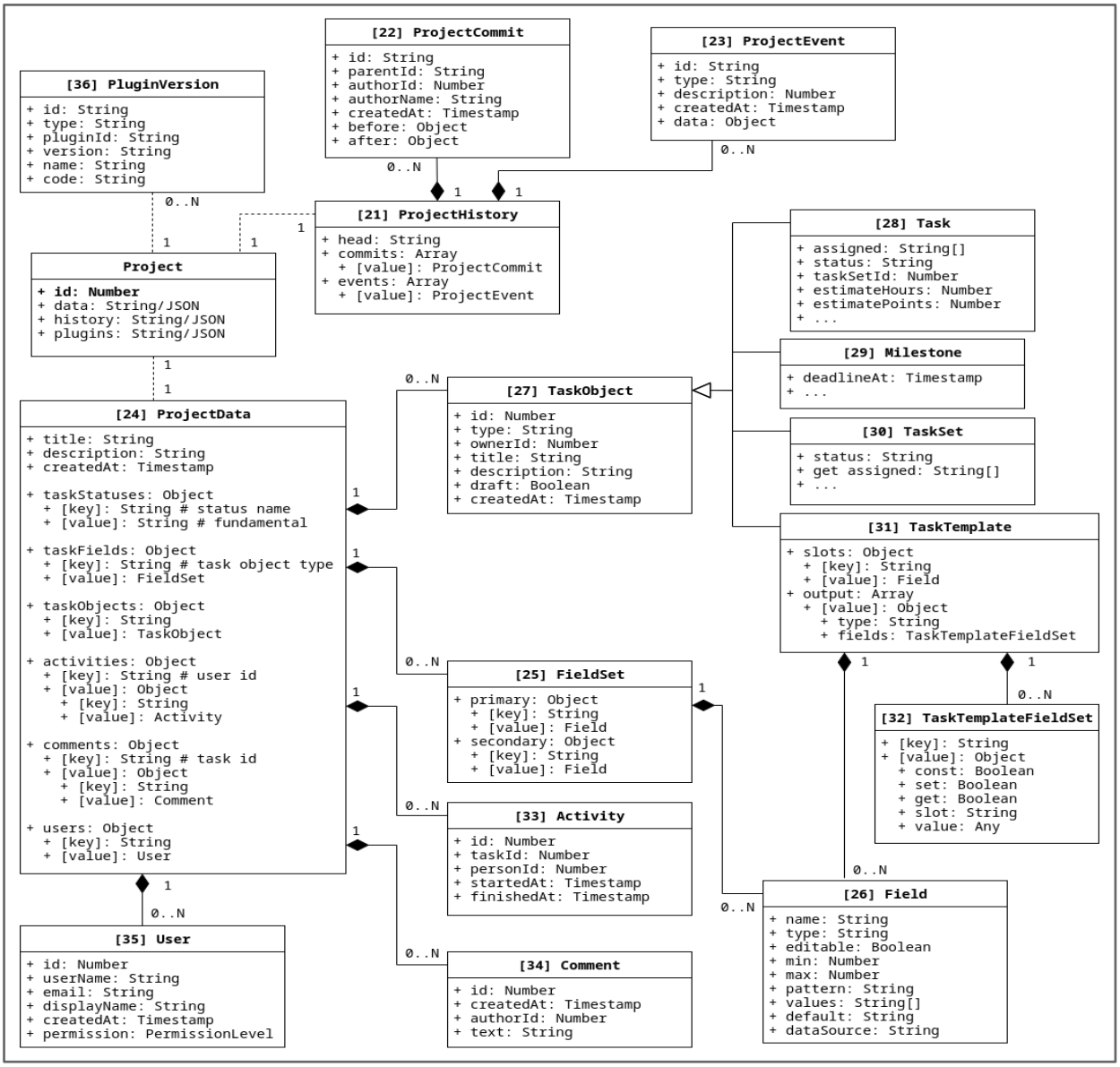


Рисунок 2.4 – Документна модель внутрішньої структури проєкту

З урахуванням особливостей використання системи, внутрішній вміст проєкту, як-от задачі, шаблони задач, коментарі та ін. буде часто змінюватися в режимі реального часу (очікується від 1 до 5 змін на хвилину), пропонується розділити сутність Project на:

– ProjectInfo, де будуть зберігатися зовнішні відомості як-от назва проєкту, id власника та дата створення;
– Project, де будуть зберігатися JSON-документи з вмістом проєкту, історією версій та переліком використаних плагінів.

У внутрішній структурі проєкту варто виділити такі типи сутностей:

– PluginVersion – містить стислу інформацію про використані плагіни що включає id плагіна, id автора, назву, версію;
– ProjectHistory – містить інформацію про історію змін задач та історію подій (приєднання нового учасника, вилучення учасника тощо);
– ProjectCommit – комміт містить інформацію про зміну в задачах проєкту, а саме: власний id, id батьківського комміту, id автора, дата створення, стан списку задач до і після змін;
– ProjectEvent – містить інформацію про подію в проєкті, а саме: тип події, опис події, дата створення;
– ProjectData – містить назву проєкту, текстовий опис проєкту, власні поля задач, власні статуси задач, задачі, коментарі до задач, відмітки про виконання роботи, учасників проєкту, групи учасників;
– TaskObject – базовий клас для всіх об’єктів у списку задач, включає ідентифікатор, тип об’єкта, id власника, заголовок, текстовий опис, дату створення;
– Task – задача, походить від TaskObject, також включає в себе id виконавців, статус, очікувану тривалість виконання тощо;
– Milestone – віха, походить від TaskObject, також включає дату дедлайну та користувацькі поля даних;
– TaskSet – набір задач, походить від TaskObject, також включає в себе статус та id виконавців;

– TaskTemplate – шаблон задачі, походить від TaskObject, включає слоти (комірки для даних) та шаблони вихідних задач, де вказано з якого слота треба взяти дані;
– Field – поле даних, включає назву, тип даних, можливість редагування даних, мінімальне/максимальне значення, шаблон для текстового рядка, допустимі значення, значення за замовченням;
– Activity – запис про виконання задачі, включає id задачі, id виконавця, дату та час початку, дату та час завершення виконання;
– Comment – коментар до задачі, включає id автора та текст.

2.3 Мережева взаємодія

Враховуючи формат web-застосунку та необхідність відображення змін у реальному часі, пропонується реалізувати мережеву взаємодію з використанням протоколів HTTP [13] та WebSocket [14].

Групи варіантів використання «Управління обліковим записом», «Адміністрування плагінів» та «Адміністрування проєкту» не потребують взаємодії в реальному часі, тому їх можна реалізувати з використанням HTTP. У таблиці 2.1 наведено перелік обробників HTTP-запитів для зазначених варіантів використання.

Таблиця 2.1 – обробники HTTP-запитів

№ в/п	Метод, URI	Опис
1	GET /api/utils/info	Інформація про API
2	GET /api/utils/time	Час на сервері
3	POST /api/auth/begin	Почати вхід в обліковий запис за іменем користувача
4	POST /api/auth/complete	Підтвердити право на користувача за допомогою пароля та коду надісланого на електронну пошту
5	POST /api/auth/forgotpass	Почати скидання паролю обл. запису
6	POST /api/auth/resetpass	Скинути пароль за допомогою коду надісланого на ел. пошту

Продовження таблиці 2.1

№ в/п	Метод, URI	Опис
7	POST /api/auth/ping	Підтвердити свіжість сесії користувача, оновити токен за необхідності
8	POST /api/user/create	Створити обліковий запис користувача
9	POST /api/user/update	Змінити обліковий запис
10	GET /api/user/whoami	Отримати інформацію про власний обліковий запис
11	GET /api/user/find	Отримати інформацію про користувача за id або username
12	GET /api/user/in-project	Отримати перелік користувачів у проєкті
13	POST /api/project/create	Створити проєкт
14	POST /api/project/update	Змінити опис проєкту
15	POST /api/project/clone	Клонувати проєкт
16	GET /api/project/find	Отримати проєкт за id
17	GET /api/project/own-list	Отримати список власних проєктів та проєктів до яких є доступ
18	GET /api/project/foreign-list	Отримати список публічних проєктів деякого користувача за його id
19	DELETE /api/project/delete	Видалити проєкт та всі пов’язані дані
20	POST /api/project/involvement/create	Додати користувача до проєкту
21	POST /api/project/involvement/update	Оновити участь у проєкті (змінити рівень доступу)
22	POST /api/project/involvement/accept	Прийняти запрошення на участь у проєкті
23	POST /api/project/involvement/reject	Відхилити запрошення
24	POST /api/project/involvement/invite	Повторно надіслати запрошення
25	POST /api/project/involvement/kick	Виключити користувача з проєкту
26	POST /api/plugin/create	Створити плагін
27	POST /api/plugin/update	Оновити опис плагіна
28	GET /api/plugin/list	Отримати перелік плагінів за іменем власника або за ключовим словом
29	GET /api/plugin/find	Отримати інформацію про плагін за id
30	POST /api/plugin/version/create	Створити нову версію плагіна
31	POST /api/plugin/version/update	Змінити версію плагіна, якщо версія ще не була опублікована
32	POST /api/plugin/version/publish	Опублікувати версію плагіна
33	GET /api/plugin/version/list	Отримати список версій плагіна
34	GET /api/plugin/version/find	Отримати повну інформацію про версію плагіна

Групи варіантів використання «Огляд проєкту» та «Редагування проєкту» передбачають надсилання великої кількості повідомлень від сервера

до клієнтів та у зворотньому напрямку упродовж тривалого часу, тому тут доречно використати WebSocket [14]. У таблиці 2.2 наведено перелік повідомлень, що надсилаються від клієнта до сервера. У таблиці 2.3 наведено перелік відповідей від сервера до клієнта.

Таблиця 2.2 – Повідомлення «клієнт > сервер»

№ в/п	Назва, параметри	Опис
1	Connect (session, project)	Підключитися до проєкту як деякий користувач, облікові дані записані в сесії
2	UpdateProjectInfo (project)	Змінити опис проєкту
3	UpdateProjectTaskStatuses (taskStatuses)	Змінити перелік статусів задач у проєкті
4	UpdateProjectTaskFields (type, taskFields)	Змінити поля даних для певного типу задач у проєкті
5	UpdatePlugins (update, reorder, toggle, remove)	Змінити, додати, вилучити плагін у проєкті
6	UpdateTasks (add, update, remove)	Змінити, додати, вилучити задачі у проєкті

Таблиця 2.3 – Повідомлення «сервер > клієнт»

№ в/п	Назва, параметри	Опис
1	Connect.BadRequest	Неправильний формат запиту при спробі підключення до проєкту
2	Connect.NotAuthorized	Сесія користувача не є дійсною
3	Connect.NotFound	Не вдалося знайти проєкт, до якого користувач намагався підключитися
4	Connect.Forbidden	Проєкт існує, але користувач не має права на доступ
5	Connect.Ok	Успішне підключення до проєкту
6	Connect.NeedsRestart	В проєкті відбулися «breaking changes» і користувач має перепідключитися до проєкту
7	Action.NotAuthorized	Користувач намагався внести зміни до проєкту, але сесія користувача не є дійсною, або користувач не підключений до проєкту
8	Action.Forbidden	Користувач не має дозволу на виконання такого виду маніпуляцій у проєкті
9	Project.Data	Початковий стан даних проєкту
10	Project.DataPatch	Внесені зміни до опису проєкту, статусів задач або полів задач
11	Project.PatchTaskObject	Внесені зміни до переліку задач у проєкті

2.4 Користувацький інтерфейс

Система буде реалізована у форматі web-застосунку. Користувацький інтерфейс визначається набором сторінок з URI (або шаблоном URI) та описом сторінки (див. табл. 2.4).

Таблиця 2.4 – Сторінки користувацького інтерфейсу

№ в/п	URI	Опис сторінки
1	/, /index	Головна сторінка з описом системи управління проєктами. Посилання: /login, /signup, /me
2	/login	Вхід в існуючий обл. запис. Посилання: /signup
3	/signup	Створення нового обл. запису. Посилання: /login
4	/me	Власний профіль та нещодавно переглянуті проєкти. Посилання: /projects?page=1
5	/projects?page={page}	Власні проєкти та проєкти зі спільним доступом, кнопка для створення нового проєкту. Параметри: page – номер сторінки
6	/project/info/{id}	Інформація про проєкт, використані плагіни та учасників, редагування списку плагінів та учасників. Параметри: id – ідентифікатор проєкту
7	/project/workspace/{id}	Робочий простір проєкту. Параметри: id – ідентифікатор проєкту
8	/plugins?query={query} &authorId={authorId} &page={page}	Перелік плагінів. Параметри: query – пошуковий запит, authorId – id автора, page – номер сторінки
9	/plugin/info/{id}	Детальна інформація про плагін, редагування (для власника). Параметри: id – ідентифікатор плагіна

2.5 Висновки до розділу 2

На основі вимог до власної системи управління проєктами побудовано модель варіантів використання, модель даних, модель мережевої взаємодії та модель користувацького інтерфейсу. Це є підґрунтям для подальшої програмної реалізації системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Перелік використаних технологій

Загальними вимогами до мов програмування та технологій, використаних у цій роботі, є безкоштовність, наявність у відкритому доступі, актуальність версій, наявність зрозумілої документації, підтримка Unix-подібних операційних систем.

Для управління базою даних розроблюваної системи обрано Postgres. Це безкоштовна СУБД, що підтримує реляційну модель даних та мову запитів SQL, регулярно оновлюється, має якісну документацію [15]. Окрім цього, для локальної розробки системи обрано СУБД Sqlite. Це мінімалістична СУБД, що зазвичай зберігає базу даних у файлі, а її виконавчий файл запускається разом з застосунком-користувачем БД [16].

Для реалізації сервера обрано мову програмування JavaScript, середовище Node.js та фреймворк Express. Переваги JavaScript для сервера:

– можливість спільного використання частин вихідного коду на сервері та на клієнті;
– простий та зрозумілий синтаксис;
– достатня стандартна бібліотека [17];
– наявність пакетного менеджера npm, в якому можна знайти пакети для будь-якої задачі [18].

JavaScript є інтерпретованою мовою програмування та має недоліки:

– низька швидкість виконання;
– неможливість виявляти помилки до виконання коду програми (вирішується використанням статичного аналізатора як-от Eslint) [19].

Спочатку мова JavaScript розроблялася для використання у web-переглядачах. Середовище Node.js дозволяє виконувати програми, написані

на мові JavaScript, з доступом до інтерфейсів операційної системи, як-от файлова система, криптографічна система та сокети [20].

Фреймворк Express дозволяє створювати серверні застосунки, оперуючи більш високорівневими об’єктами, як-от Router, Request, Response. Це значно прискорює розробку застосунку в порівнянні з використанням вбудованого низькорівневого модуля http в Node.js [21].

Для забезпечення взаємодії серверного застосунку з базою даних обрано ORM Objection. Перевагами Objection є гнучкість, підтримка багатьох реляційних СУБД (в тому числі Postgres та Sqlite), всеосяжна документація [22]. Недоліком Objection є необхідність ручного створення таблиць бази даних [23, 24], для усунення цього недоліку використано розширення Better-Objection, що дозволяє створювати таблиці на основі полів, оголошених у класі моделі [25].

Станом на 2024 рік існує багато фреймворків, метою яких є полегшити розробку клієнтських частин вебзастосунків, і вибір фреймворку зазвичай залежить від існуючого вихідного коду застосунку, а в разі його відсутності – від знань та вмінь розробника. Оскільки ця система управління проєктами розробляється «з нуля», тобто не має попередніх версій, то довільним чином було обрано фреймворк Vue.

Vue – це сучасний фреймворк, який дозволяє створювати вебзастосунки на основі компонентів [26]. Перевагами Vue є:

– модульність – застосунок можна поділити на компоненти, компоненти можуть передавати дані до «дітей» за допомогою властивостей (props);
– реактивність – будь-який об’єкт JavaScript можна зробити реактивним, при внесенні змін залежні від нього об’єкти та елементи HTML будуть автоматично оновлюватися;
– декларативність – необхідно лише описати зовнішній вигляд компонента, а фреймворк бере на себе всю роботу з HTML-документом;

– повторне використання компонентів;
– гнучкість – можна описувати компоненти у форматі Vue SFC, з використанням рендер-функцій або з використанням JSX як у React;
– регулярне оновлення;
– всеосяжна документація [26].

3.2 Реалізація бази даних

Для прикладу розглянемо сутність «ProjectInfo» (див. рис. 3.1).

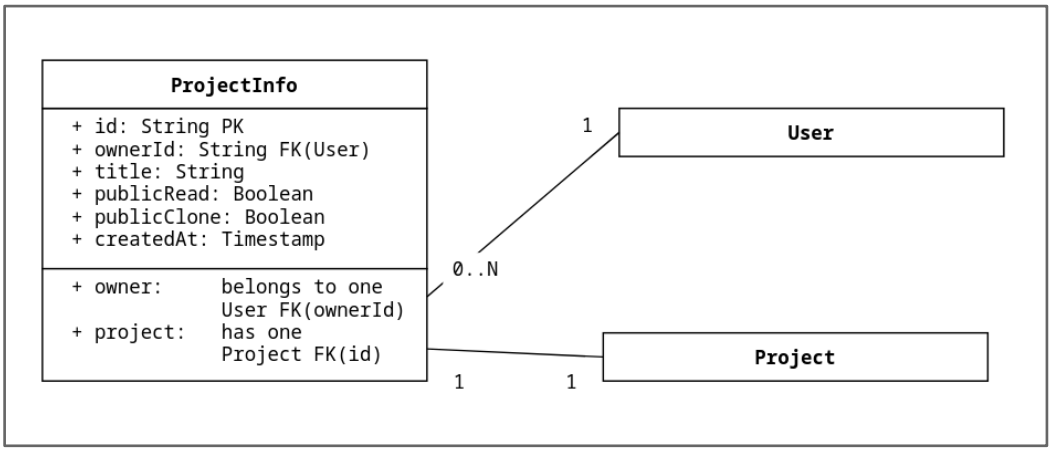


Рисунок 3.1 – Сутність ProjectInfo

З використанням мови JavaScript та ORM Objection створюється клас моделі «ProjectInfo» (див. рис. 3.2), що наслідує базовий клас Model з бібліотеки Better-Objection. Мова JavaScript не вимагає типізації, але це потрібно для реляційної моделі даних. В класі моделі обов’язково має бути визначено поле props, де зазначаються поля реляційної моделі даних та пов’язані сутності. Різниця між визначенням поля таблиці та пов’язаної сутності полягає в наступному: якщо type є класом, що наслідує клас Model – це сприймається як пов’язана сутність, якщо type відповідає назві примітивного типу даних – це поле таблиці, в іншому випадку це сприймається як помилка [25]. Так само створюються класи для інших сутностей.

```
import { Model, type } from "better-obj"
import { pk, fk, cascade, max, belongsToOne, hasOne, unique, json } from "better-obj"

const custom = { Timestamp: type.Integer }

export class ProjectInfo extends Model
{
    static get props() {
        return {
            id: { type: type.String, rules: [ pk() ] },
            ownerId: { type: type.String, rules: [ fk(User) ] },
            title: { type: type.String, rules: [ max(128) ] },
            publicRead: { type: type.Boolean, rules: [ ] },
            publicClone: { type: type.Boolean, rules: [ ] },
            createdAt: { type: custom.Timestamp, rules: [ ] },
            owner: { type: User, rules: [ belongsToOne() ] },
            project: { type: Project, rules: [ hasOne() ] }
        }
    }

    static rules = {
        title: {
            min: 0,
            max: 128,
            default: "Untitled Project",
            clamp (value) {
                return value.slice(0, 128)
            }
        }
    }
}
```

Рисунок 3.2 – Клас моделі «ProjectInfo»

Для прикладу роботи з складними структурами даних, розглянемо сутність «Project». Щоб автоматично серіалізувати поле у формат JSON при записі у БД та десеріалізувати при читанні з БД, потрібно задати правило «json» у визначенні цього поля у класі сутності [25]. Вихідний код класу моделі «Project» наведено на рисунку 3.3.

```
import { Model, type } from "better-obj"
import { pk, fk, cascade, max, belongsToOne, hasOne, unique, json } from "better-obj"

export class Project extends Model
{
    static get props() {
        return {
            id: { type: type.String, rules: [ pk(), fk(ProjectInfo), cascade() ] },
            version: { type: type.Integer, rules: [ ] },
            data: { type: type.Object, rules: [ json() ] },
            history: { type: type.Object, rules: [ json() ] },
            plugins: { type: type.Object, rules: [ json() ] }
        }
    }
}
```

Рисунок 3.3 – Клас моделі «Project»

Для надання централізованого доступу до бази даних створюється клас «ProjDbAdapter» (див. рис. 3.4), що наслідує базовий клас DbAdapter з бібліотеки Better-Objection [25]. При запуску застосунку, що використовує цю базу даних, потрібно підключити адаптер до бази даних та створити таблиці за необхідності (див. рис. 3.5). Запити до таблиць бази даних здійснюються так само, як в оригінальній ORM Objection [22]. Приклад запиту, що знаходить проєкт за його id, наведено на рисунку 3.6.

```
import { User, UserSession, ProjectInfo, Project,
         ProjectInvolvement, Plugin, PluginVersion } from "./models.js"

import { DbAdapter } from "better-obj"

export class ProjDbAdapter extends DbAdapter
{
    constructor() {
        super({
            user: User,
            userSession: UserSession,
            projectInfo: ProjectInfo,
            project: Project,
            projectInvolvement: ProjectInvolvement,
            plugin: Plugin,
            pluginVersion: PluginVersion
        })
    }
}
```

Рисунок 3.4 – Адаптер бази даних

```
import { connections } from "./database/connect.js"
let dbConnection = connections[env.DB_PROVIDER]()

import { ProjDbAdapter } from "./database/database.js"
let dbAdapter = new ProjDbAdapter()
await dbAdapter.connect(dbConnection)
await dbAdapter.createDb()
```

Рисунок 3.5 – Підключення до бази даних

```
let theProject = await dbAdapter.projectInfo.query()
    .withGraphFetched("project")
    .where("id", projectId)
    .where("ownerId", userId)
    .first()
```

Рисунок 3.6 – Приклад запиту до бази даних

3.3 Реалізація серверної частини

3.3.1 Реалізація взаємодії по проколу HTTP

Розглянемо реалізацію варіанту використання «Переглянути список власних проєктів». Цей варіант використання вимагає, щоб клієнт був авторизований. Взаємодія клієнта та сервера здійснюється через API Endpoint /api/project/own-list.

Для реалізації сервера створимо основну систему, яка буде складатися з іменованих модулів (тут мається на увазі клас, а не модуль npm) для груп варіантів використання (як-от User, Project), та буде працювати незалежно від HTTP.

До іменованого модуля Project додається метод getOwnList, що приймає номер сторінки, id власника проєкту, статус запрошення до проєкту, контекст з підтвердженим id користувача (див. додаток А.1).

Далі з використанням Express створюється роутер для /api/project, до роутера додається обробник запиту /api/project/own-list. Цей обробник спочатку звертається до методу getUser модуля Auth. Отриманий результат передається в якості контексту до метода getOwnList модуля Project. Отриманий результат з останнього методу надсилається клієнтові у форматі JSON (див. додаток А.2).

З використанням Express створюється новий HTTP-сервер, до нього додається раніше створений роутер. Також створюється екземпляр основної системи з усіма залежностями, як-от база даних та логгер.

За допомогою спеціального middleware (проміжного обробника HTTP-запитів) екземпляр основної системи передається до роутера через об’єкт запиту (див. додаток А.3).

Об’єкти, задіяні у розглянутому вище варіанті використання, зображені на рисунку 3.7.

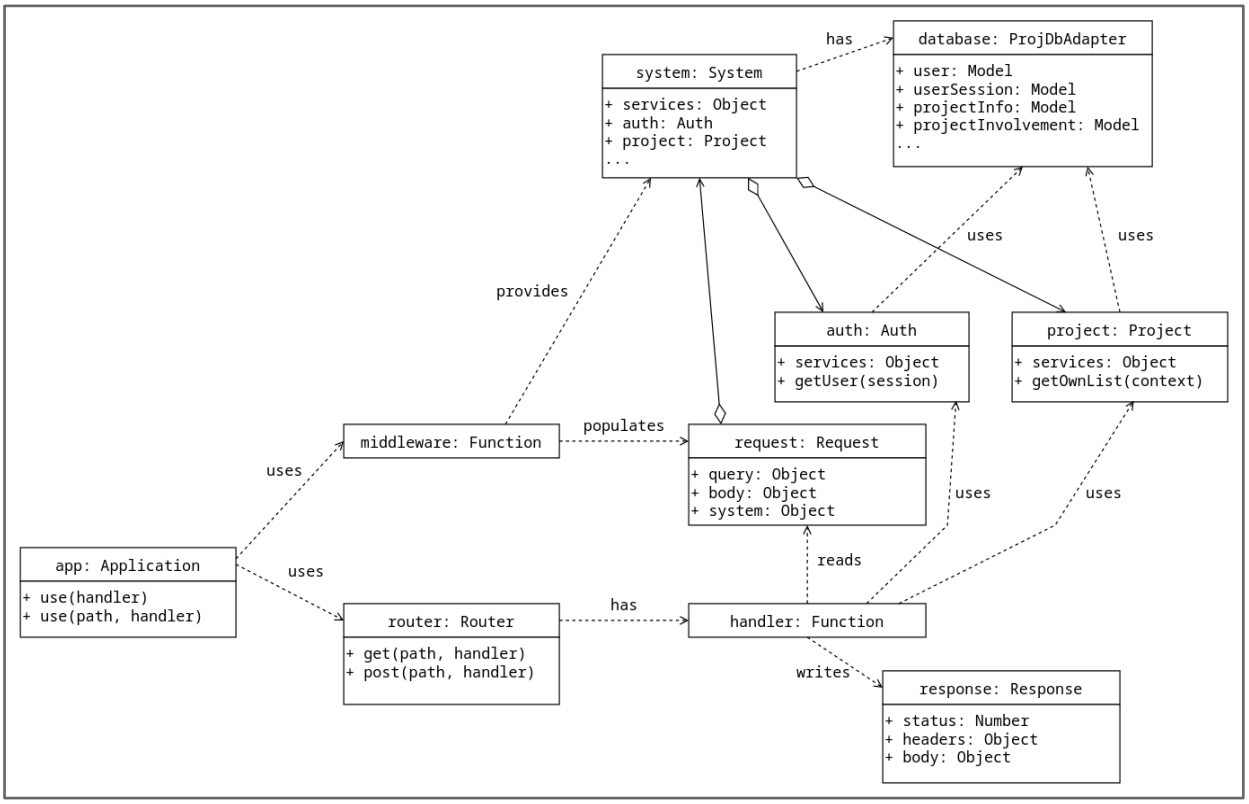


Рисунок 3.7 – Об’єкти у перегляді списку власних проєктів

3.3.2 Реалізація взаємодії по протоколу WebSocket

Взаємодія з робочим простором проєкту відбувається за протоколом WebSocket та вимагає, щоб користувач був авторизований та мав право на доступ до проєкту.

До розробленої раніше основної системи додається іменований модуль ProjectWorkspace. До сервісу events, що відповідає за створення та передачу подій, додається обробник події, що спрацьовує при додаванні екземпляру сервера WebSocket та прив’язує обробника підключень у модулі ProjectWorkspace до події «connection» сервера WebSocket. Об’єкти, задіяні при налаштуванні WebSocket, показані на рисунку 3.8.

При новому підключенні головний обробник повідомлень прив’язується до події «message» екземпляру підключення. Екземпляр підключення додається до списку активних підключень. Об’єкти, задіяні при підключенні, наведені на рисунку 3.9.

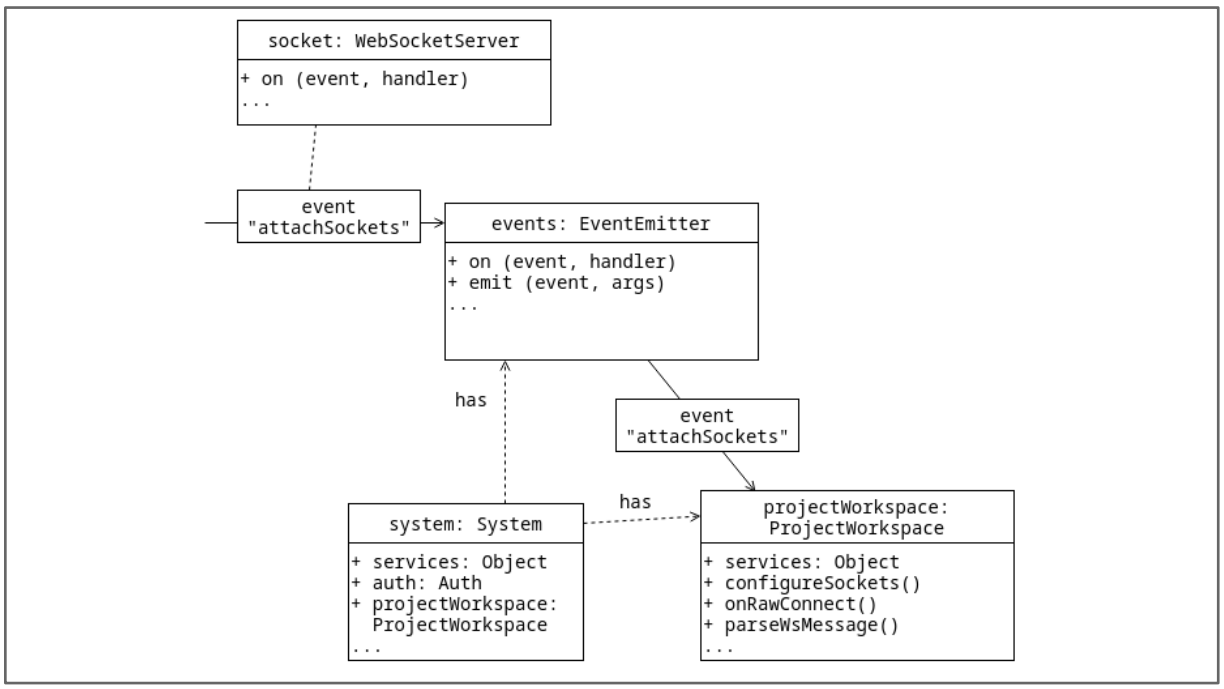


Рисунок 3.8 – Налаштування WebSocket

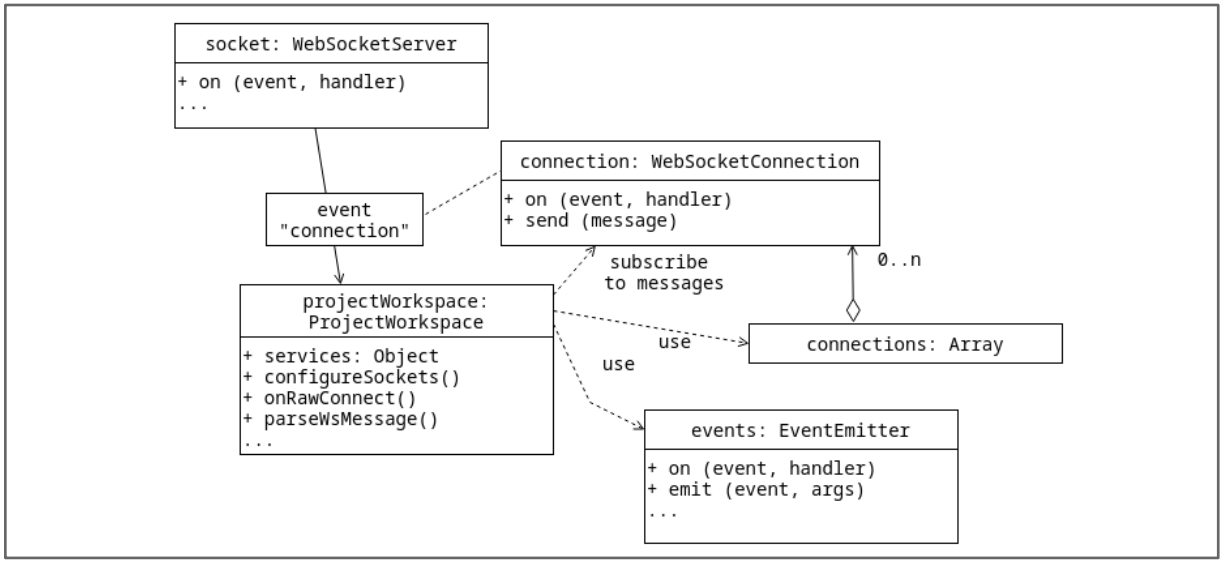


Рисунок 3.9 – Обробка нового підключення

Головний обробник повідомлень десеріалізує повідомлення з формату JSON та опосередковано передає їх через сервіс events до метода-обробника з назвою, що відповідає типу повідомлення, а якщо обробник не знайдений – повідомлення ігнорується.

Очікується, що першим повідомленням від клієнта буде «Connect». При обробці цього повідомлення метод onWspConnect перевіряє дійсність сесії користувача та право доступу цього користувача до проєкту. Якщо перевірка

пройшла успішно, обробник записує id користувача, id проєкту та рівень доступу в контекст підключення, та записує проєкт, отриманий з бази даних, у кеш проєктів. Обробник надсилає клієнтові повідомлення «Connect.Ok» та повідомлення «Project.Data» з внутрішнім вмістом проєкту. Об’єкти, що задіяні при обробці повідомлення «Connect», наведені на рисунку 3.10.

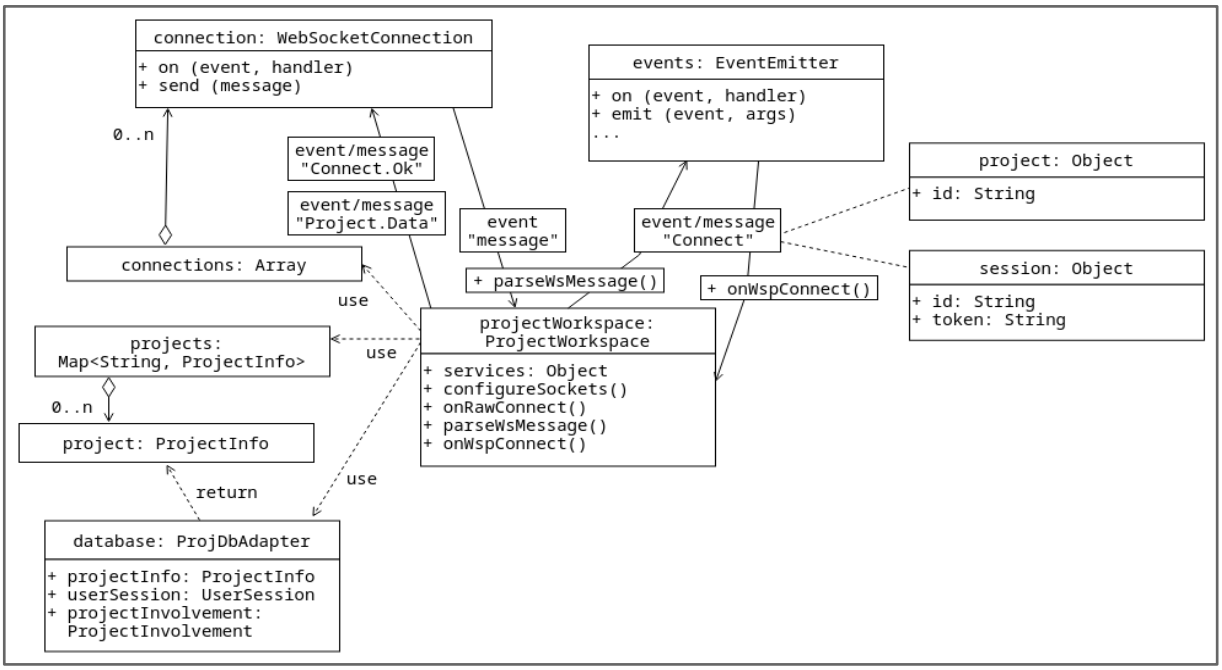


Рисунок 3.10 – Обробка повідомлення «Connect»

При подальшій взаємодії підтвердження «особи» клієнта здійснюється на основі id користувача, id проєкту та рівня доступу в контексті підключення. Якщо ці значення не встановлені, клієнт отримає повідомлення «Action.NotAuthorized».

Для прикладу розглянемо редагування опису проєкту. Клієнт надсилає повідомлення «UpdateProjectInfo». Якщо перевірка «особи» пройшла успішно, обробник повідомлення знаходить екземпляр проєкту в кеші, оновлює відповідні поля, помічає проєкт як змінений. Внесені зміни до опису проєкту розсилаються всім клієнтам, що мають id цього проєкту в контексті підключення (тобто підключені до цього проєкту). Об’єкти, що задіяні при обробці повідомлення «UpdateProjectInfo», наведені на рисунку 3.11. Вихідний код іменованого модуля ProjectWorkspace наведено у додатку А.4.

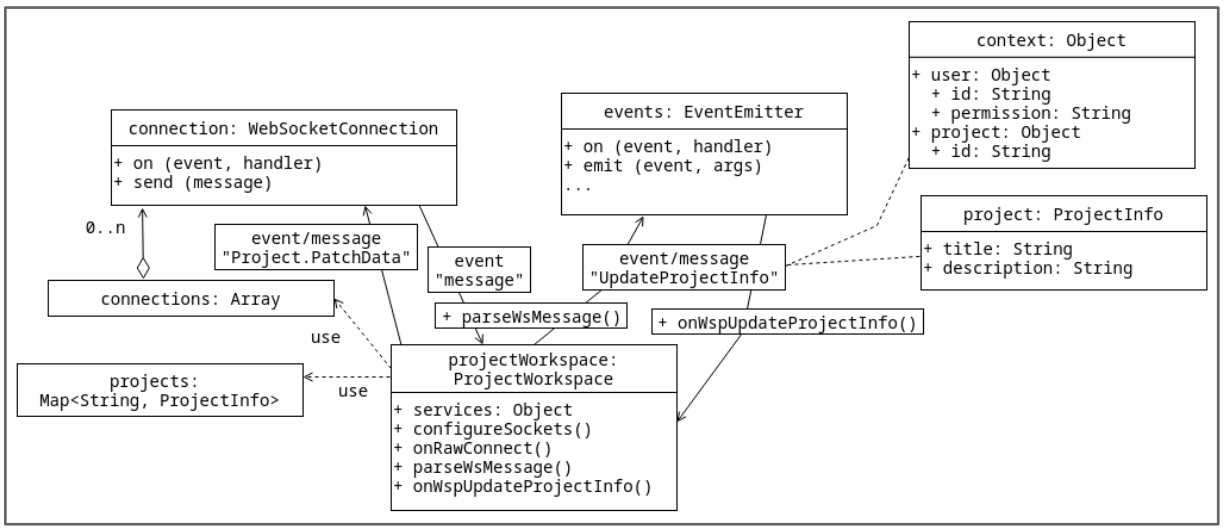


Рисунок 3.11 – Обробка повідомлення «UpdateProjectInfo»

3.4 Реалізація клієнтської частини

Клієнтська частина реалізована у вигляді односторінкового web-застосунку з використанням фреймворку Vue. Для прикладу розглянемо реалізацію сторінки «project/info» для відображення інформації про проєкт та його учасників. Сторінка реалізовується у вигляді компонента Vue. Сторінка має один параметр – id проєкту. При переході на сторінку, спрацьовує спостерігач за параметром id, який викликає метод для отримання проєкта з /api/project/find. Якщо проєкт знайдено, викликається метод для отримання учасників цього проєкту з /api/user/in-project. Вихідний код сторінки наведено в додатку Б.1.

Сторінка «project/workspace» містить в собі вкладений застосунок Vue. При переході на цю сторінку створюється підключення до сервера по протоколу WebSocket. Клієнт надсилає повідомлення «Connect» в якому зазначається сесія користувача та ідентифікатор проєкту. В разі отримання від сервера повідомлення «Connect.Ok» та «Project.Data» виконується ініціалізація проєкту та плагінів, відображається головна сторінка робочого простору проєкту, що містить меню та опис проєкту. В разі отримання «Connect.NotAuthorized» або інших повідомлень про помилку, відповідна

інформація відображається на сторінці. Вихідний код сторінки наведено в додатку Б.2.

Завантажувальник плагінів спрацьовує при отриманні переліку плагінів у повідомленні «Project.Data» та використовує можливості вбудованого інтерпретатора мови JavaScript. Вихідний код плагіна отримується з CDN-сервера для external-плагінів, або з самого опису плагіна для inline-плагінів. На основі коду плагіна створюється об'єкт Function (функція), що приймає такі параметри: imports, exports, require. Щоб плагін можна було встановити у вкладеному застосунку, плагін має експортувати об'єкт «default» з такими методами: install(app), mount(app). Створена функція викликається з такими параметрами: imports – об'єкт з усіма наявними імпортами, require (name) – функція що повертає значення з об'єкта imports за вказаним ключем name, exports – пустий об'єкт. Якщо отриманий об'єкт exports.default має методи install або mount, ці методи викликаються з параметром app – екземпляр застосунку, що містить проєкт, плагіни, перелік сторінок та пунктів меню, екземпляр підключення WebSocket. Плагін має право вносити зміни до отриманих об'єктів. Вихідний код завантажувальника плагінів включено до коду сторінки «project/workspace» у додатку Б.2.

Для прикладу розглянемо вкладену сторінку «settings.project». Сторінка відображає текстові поля для вводу назви та опису проєкту. При натисканні кнопки «Save» клієнт надсилає повідомлення «UpdateProjectInfo» по протоколу WebSocket. Відповідь від сервера «Project.DataPatch» обробляється не цією сторінкою, а раніше розглянутим вкладеним застосунком, що керує відображенням усіх сторінок. Вихідний код сторінки «settings.project» наведено в додатку Б.3.

Навігація між сторінками основного застосунку здійснюється за допомогою бібліотеки Vue-Router [27]. Роутер визначається списком об'єктів, в яких визначено path – URI сторінки, component – компонент що буде відображатися, props – параметри з URI що мають бути передані до сторінки. Вихідний код налаштувань роутера для застосунку наведено в додатку Б.4.

3.5 Висновки до розділу 3

Тут описано програмну реалізацію бази даних, серверної частини та клієнтської частини системи управління проєктами на мові програмування JavaScript. Для реалізації бази даних засобами ORM Objection та Better-Objection створено класи сутностей та адаптер бази даних. Серверна частина реалізована з використанням Node.js, Express та WebSocket. Клієнтська частина реалізована з використанням фреймворку Vue.

4 ВИКОРИСТАННЯ СИСТЕМИ

4.1 Розгортання системи

Система може бути розгорнута на фізичній або віртуальній машині з таким середовищем та технологіями:

– Linux, версія ядра ≥ 5.15;
– OpenSSL, версія ≥ 3.0;
– Node.js, версія ≥ 18.9;
– Npm, версія ≥ 8.19.

Для розгортання потрібно клонувати репозиторій, редагувати файл back/.env за зразком back/.env.example [28] та виконати такі команди (див. рис. 4.1).

```
cd ./var/openssl/ && ./create-dev-cert.sh && cd ../../
cd ./@common && npm run ci && cd ..
cd ./front && npm run ci && cd ..
cd ./back && npm run ci && cd ..
cd ./back && npm run serve && cd ..
```

Рисунок 4.1 – Перелік команд для розгортання системи

Система також може бути розгорнута у середовищі Docker з використанням Docker-Compose [29]. Для цього потрібно клонувати репозиторій, редагувати файл back/.env за зразком back/.env.example [28] та виконати такі команди (див. рис. 4.2)

```
# use this command for sqlite or external postgres instance
docker-compose up

# use this for dockerized postgres instance
docker-compose --file compose.yaml --file database/postgres.compose.yaml up
```

Рисунок 4.2 – Перелік команд для розгортання з Docker-Compose

У разі успішного запуску сервера, в терміналі мають з’явитися такі повідомлення (див. рис. 4.3).

```
dmitriy@debby:~/Code/proj/back$ npm run serve

> proj-back@0.0.1 serve
> node src/index.js

[13:04:44][i] Using stub/fake email sender
[13:04:44][i] wsp: using handler onWspConnect
[13:04:44][i] wsp: using handler onWspUpdateProjectInfo
[13:04:44][i] wsp: using handler onWspUpdateProjectTaskStatuses
[13:04:44][i] wsp: using handler onWspUpdatePlugins
[13:04:44][i] wsp: using handler onWspUpdateTask
[13:04:44][i] wsp: using handler onWspTypelessMessage
[13:04:44][i] wsp: using handler onWspBadMessage
[13:04:44][i] wsp: using handler onWspPing
[13:04:44][i] Using sockets
[13:04:44][i] Using static server for ./dist
[13:04:44][i] Using static fallback ./dist/index.html
[13:04:44][i] Server listening to port 4000
```

Рисунок 4.3 – Результат запуску сервера

4.2 Робота з користувацьким інтерфейсом

Після запуску сервера, для початку роботи з застосунком необхідно відкрити у web-переглядачі https://<ip>:<port>/, де ip – IP-адреса сервера, на якому було розгорнуто застосунок, port – номер порту на якому працює застосунок. Номер порту зазначається у файлі конфігурації сервера та відображається у стандартному виводі сервера (див. рис. 4.3). При нормальній роботі застосунку має відображатися така сторінка (див. рис. 4.4).

Для створення нового облікового запису потрібно перейти на сторінку /signup, ввести у відповідні поля ім’я користувача, електронну адресу та власне ім’я (див. рис. 4.5).

При успішній реєстрації має відображатися повідомлення «You have successfully signed up».

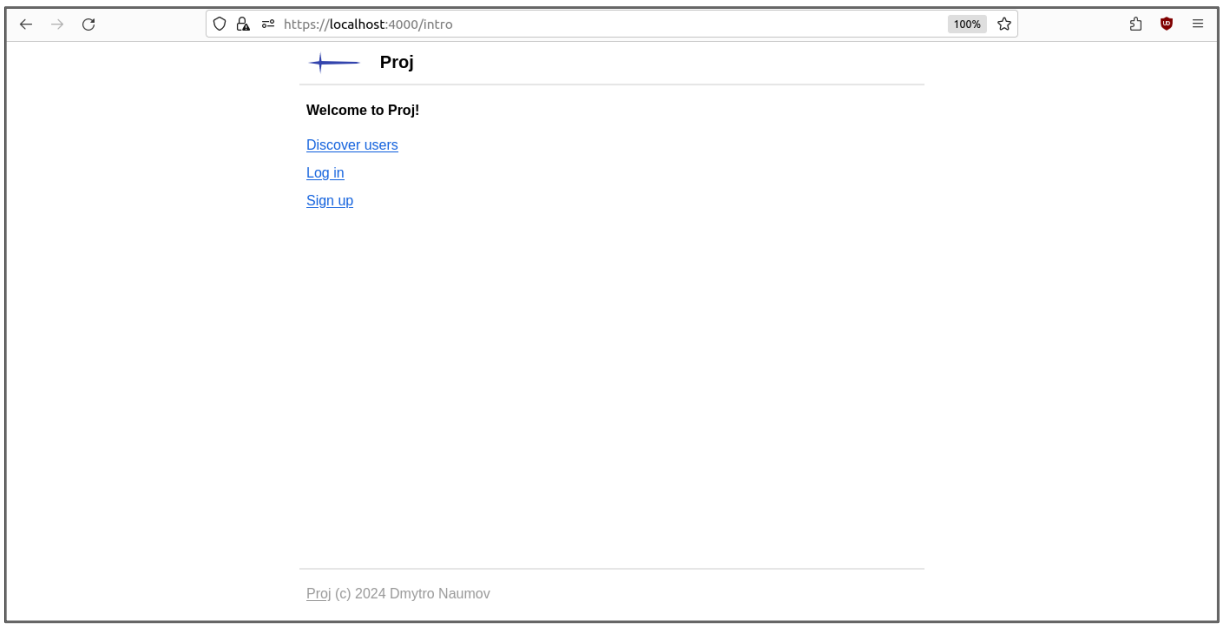


Рисунок 4.4 – Головна сторінка застосунку

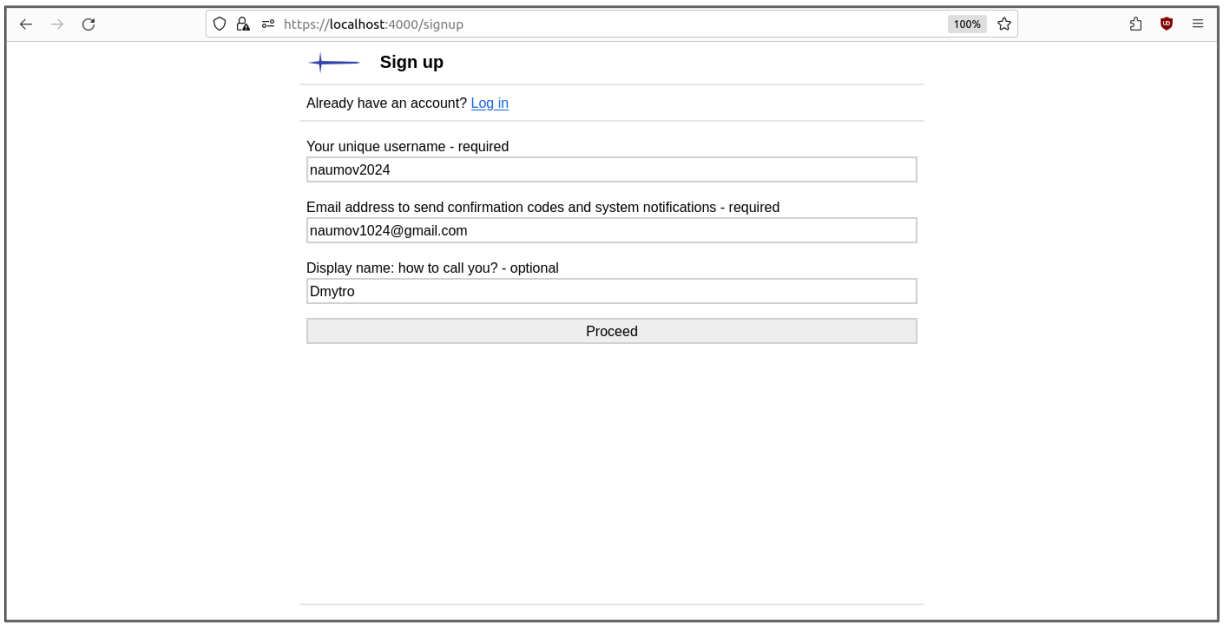


Рисунок 4.5 – Створення облікового запису

Для входу в обліковий запис необхідно перейти на сторінку /login та ввести ім’я користувача у відповідне поле.

Якщо користувач існує, система має надіслати код підтвердження на електронну адресу, що вказана у обліковому записі. Необхідно ввести отриманий код у відповідне поле та за потреби ввести пароль (див. рис. 4.6). При успішному вході буде здійснено перехід на сторінку /me з обліковим записом користувача (див. рис. 4.7).

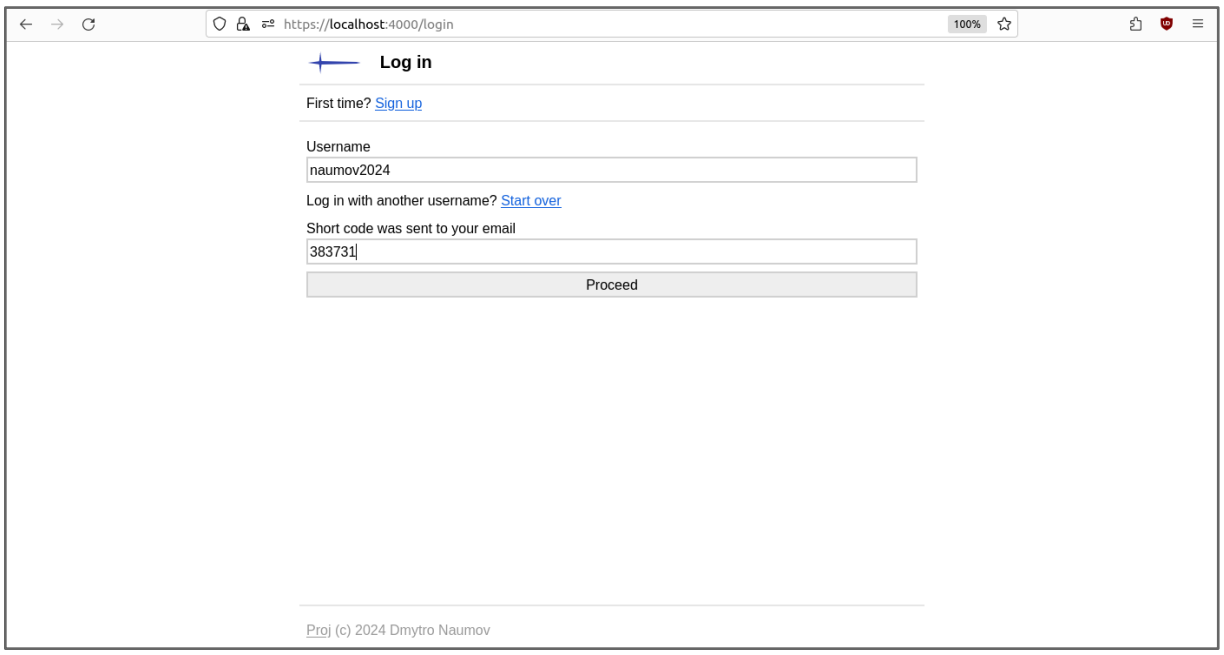


Рисунок 4.6 – Вхід в обліковий запис

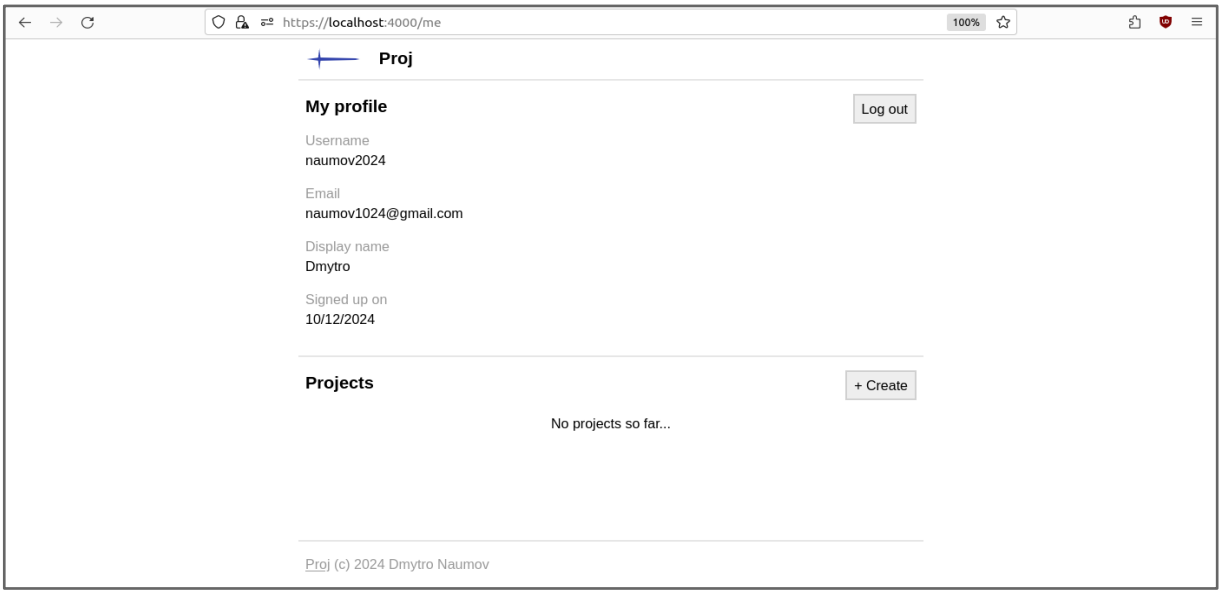


Рисунок 4.7 – Новий обліковий запис користувача

Для того щоб створити проєкт, потрібно натиснути кнопку «+ Create» у переліку власних проєктів. Щоб переглянути загальну інформацію про створений проєкт, потрібно натиснути на картку цього проєкту (див. рис. 4.8). Сторінка з загальною інформацією про проєкт наведена на рисунку 4.9.

Щоб підключитися до робочого простору проєкту, потрібно натиснути на посилання «Go to Workspace». Вигляд робочого простору проєкту наведено на рисунках 4.10 – 4.15.



Рисунок 4.8 – Відображення створеного проєкту

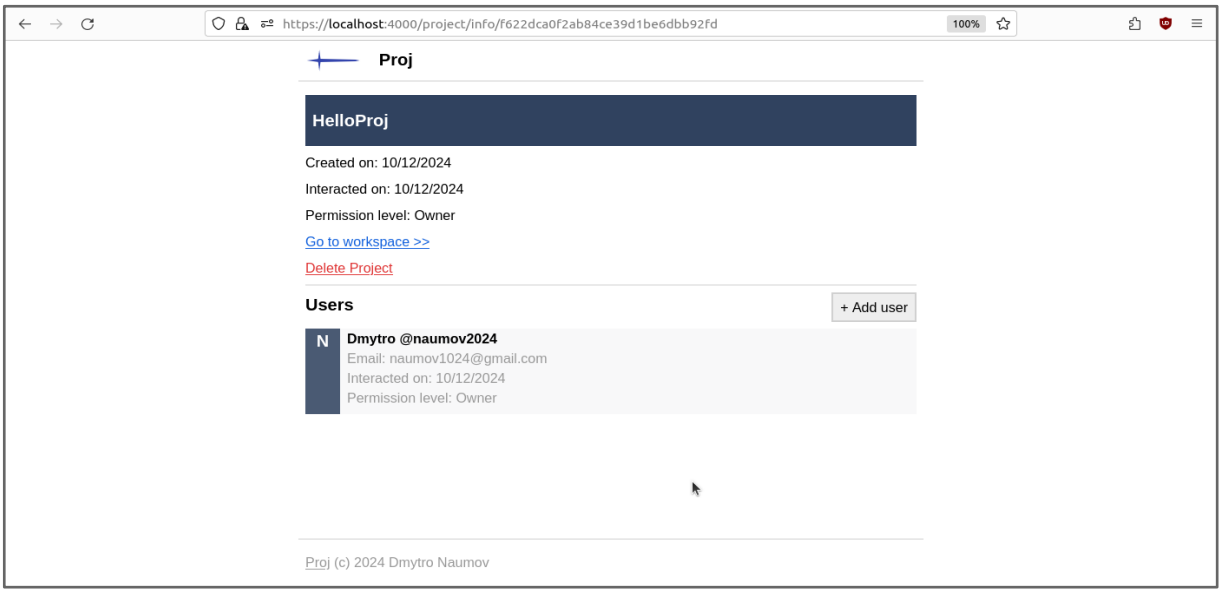


Рисунок 4.9 – Перегляд загальної інформації про проєкт

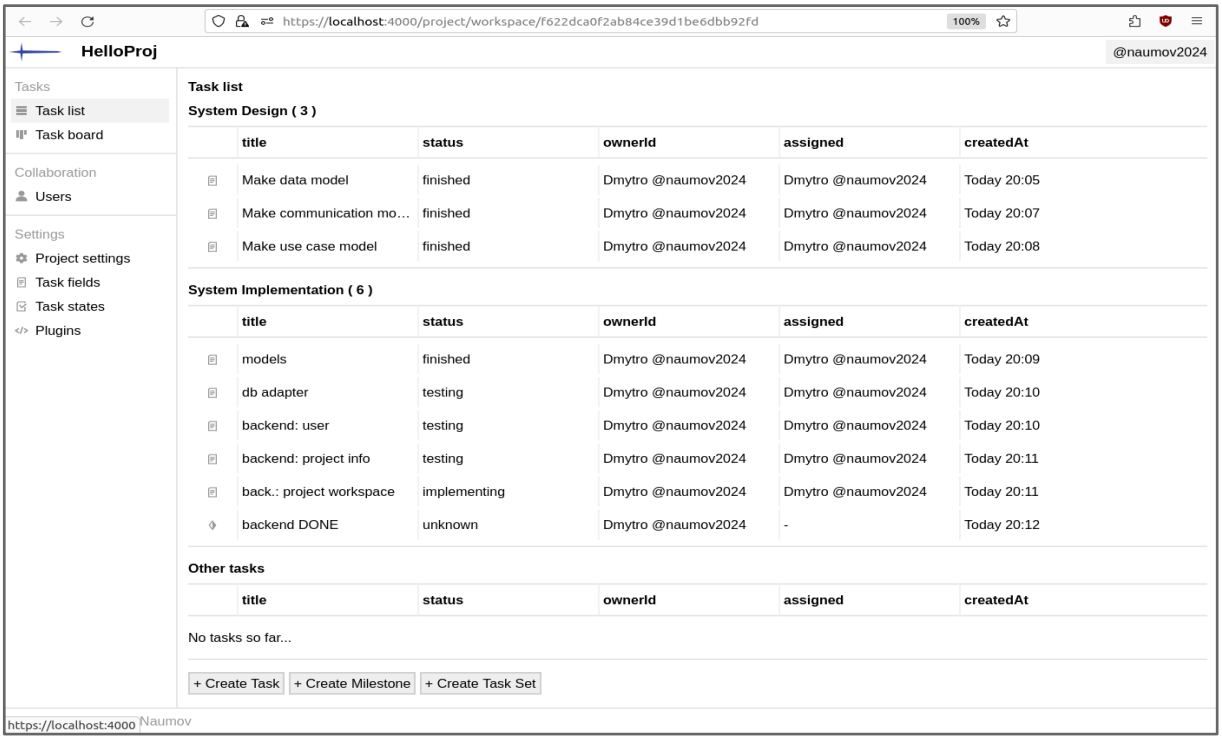


Рисунок 4.10 – Список задач

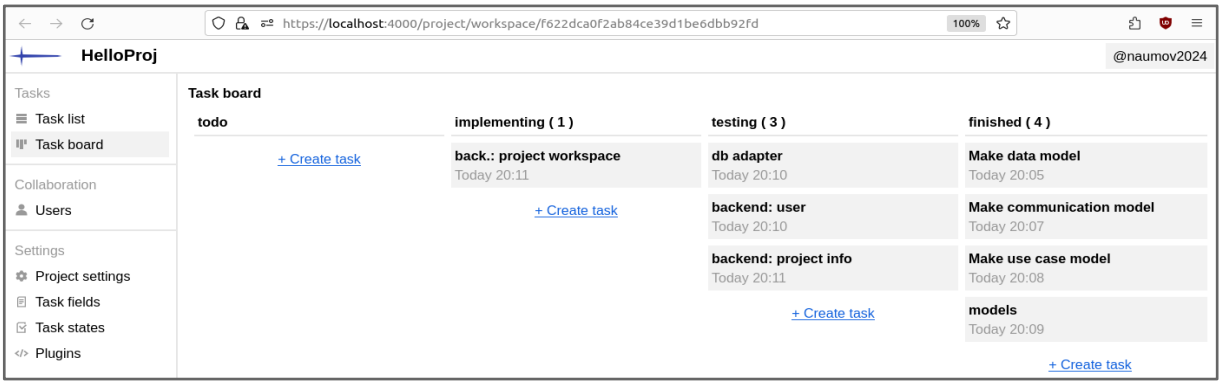


Рисунок 4.11 – Дошка задач



Рисунок 4.12 – Список учасників проєкту

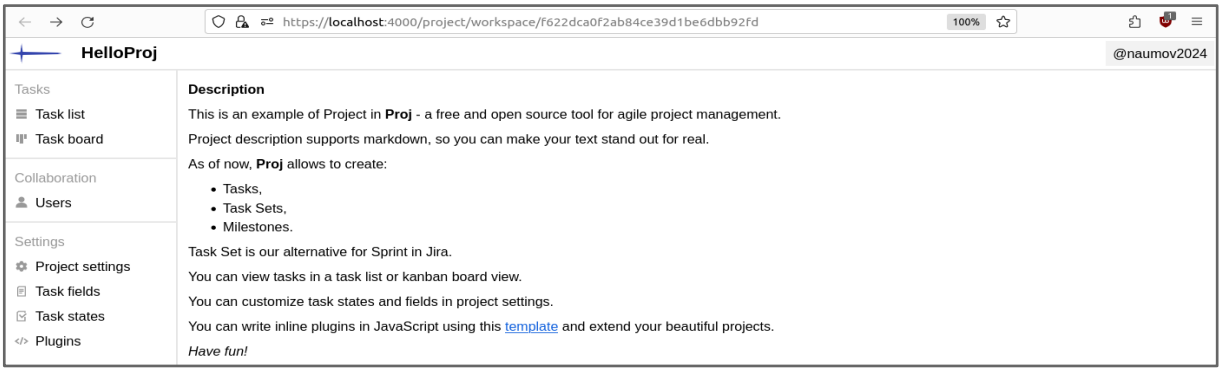


Рисунок 4.13 – Опис проєкту

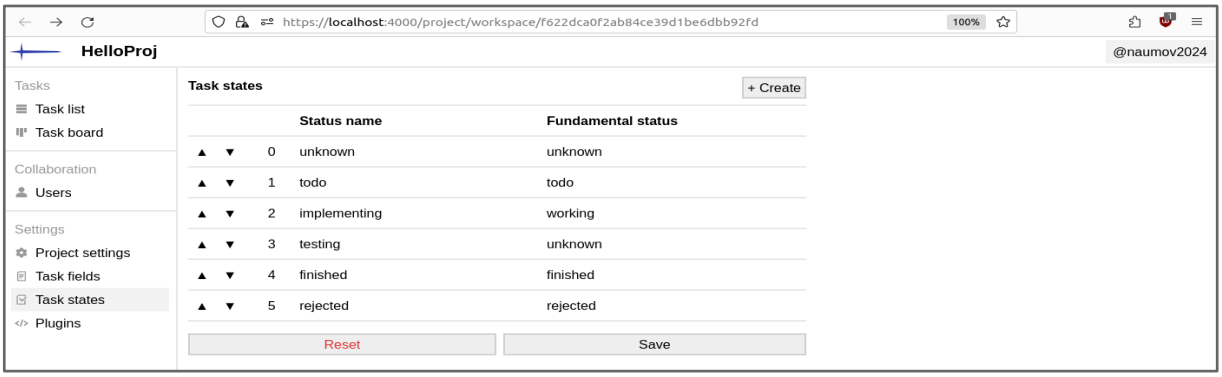


Рисунок 4.14 – Налаштування статусів задач

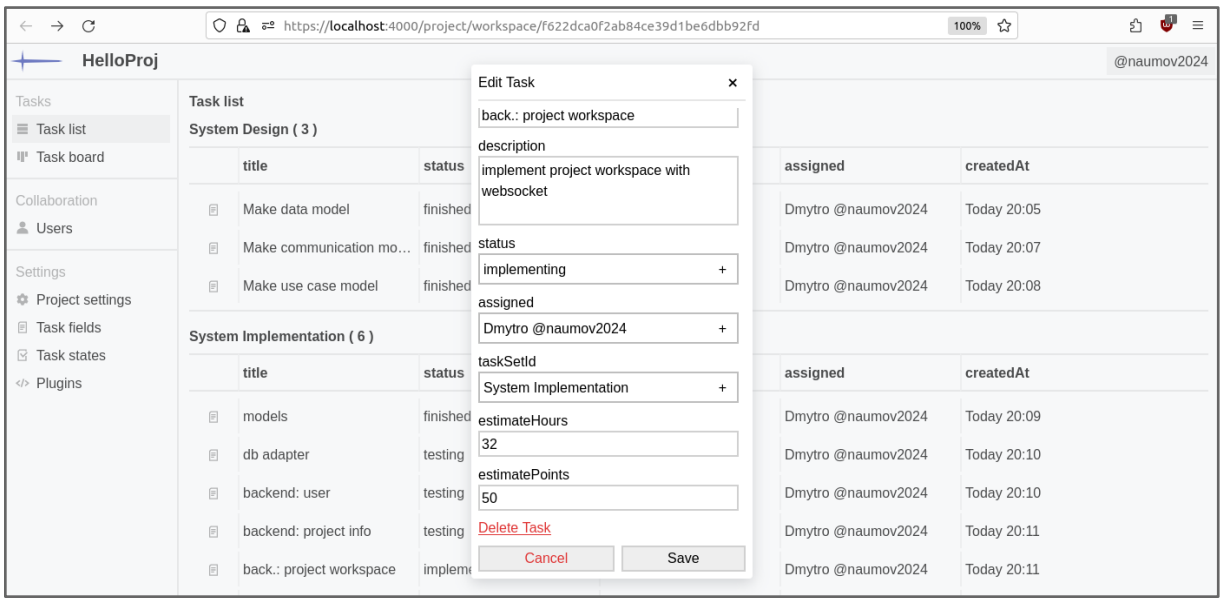


Рисунок 4.15 – Редагування задачі

4.3 Розширення користувацького інтерфейсу

Розроблена система дозволяє створювати плагіни з використанням фреймворку Vue та двох принципово різних підходів:

– «straight» коли компоненти Vue пишуться з використанням чистого JavaScript та рендер-функцій та весь код має бути в єдиному файлі;

– «compiled» з використанням Vue SFC, Vite та інших інструментів за бажанням розробника.

Налаштувати розробку нового плагіна можна з використанням шаблону [30] що запускається командами:

– mkdir {path};

– npx init github:dmitriynaumov1024/proj-plugin {path}{pluginname}.

Розглянемо приклад створення плагіна для відображення розподілу задач за статусом у поточний момент. Вихідний код плагіна наведено в додатку В.1. Для того, щоб скомпілювати плагін, потрібні такі інструменти:

– Node.js, версія 18 або вище;

– Npm, версія 8.19 або вище.

Для компіляції плагіна необхідно виконати такі команди:

– npm install;

– npm run build.

В результаті отримаємо директорію dist з файлами plugin.js та style.css. Далі потрібно перейти на сторінку «Plugins» у робочому просторі проєкту, створити новий плагін, натиснувши на кнопку «+ Create», та відкрити щойно створений плагін. На сторінці редагування плагіна потрібно вставити вміст раніше отриманого файлу plugin.js у текстове поле «Code», а вміст файлу style.css – у текстове поле «Style», та натиснути кнопку «Save» (див. рис. 4.16). За необхідності перезавантажити сторінку.

Після додавання плагіна, в робочому просторі проєкту має з’явитися меню «View» з пунктом «Dashboard». Якщо відкрити цей пункт меню, має відображатися така сторінка (див. рис. 4.17).

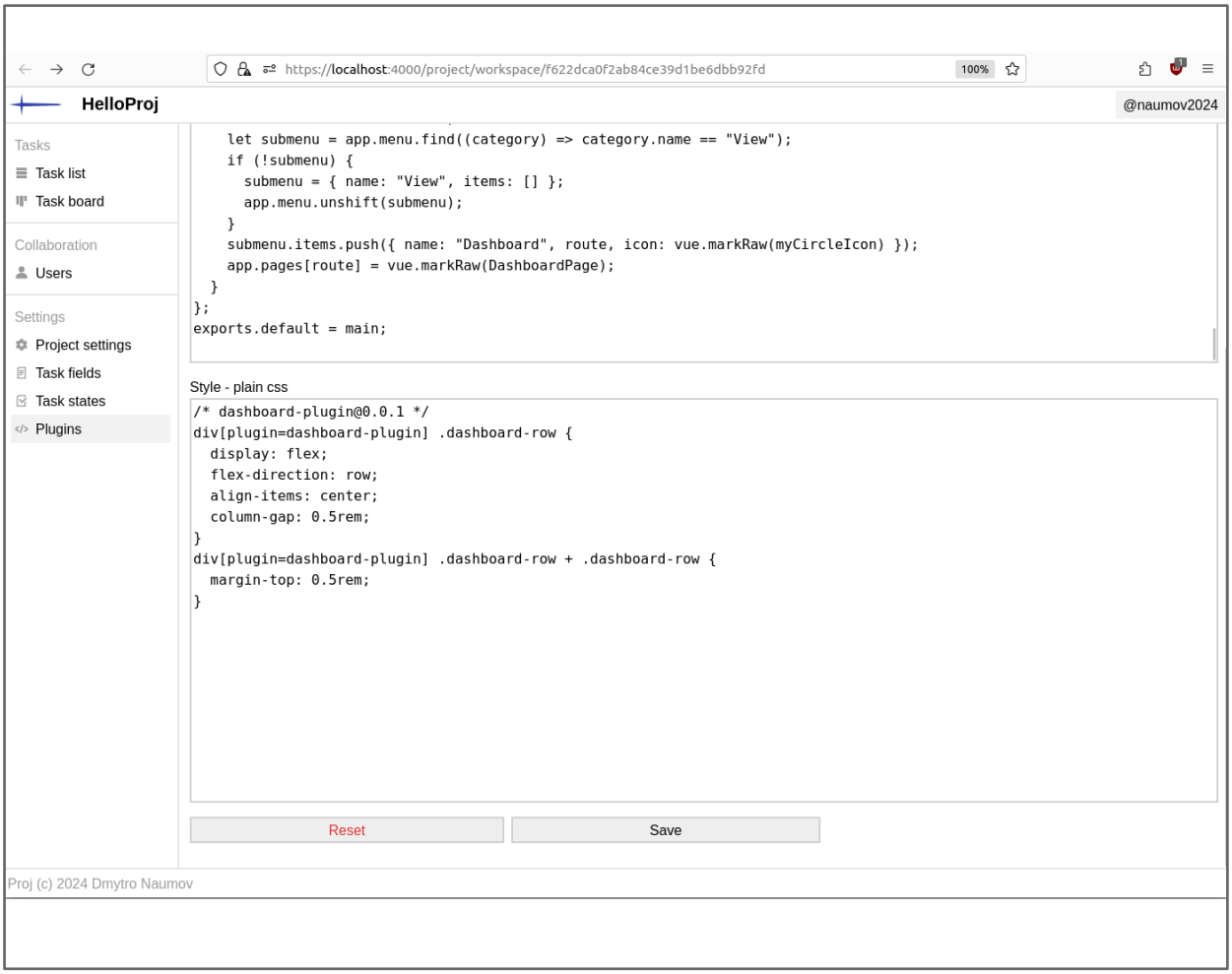


Рисунок 4.16 – Додавання плагіна до проєкту

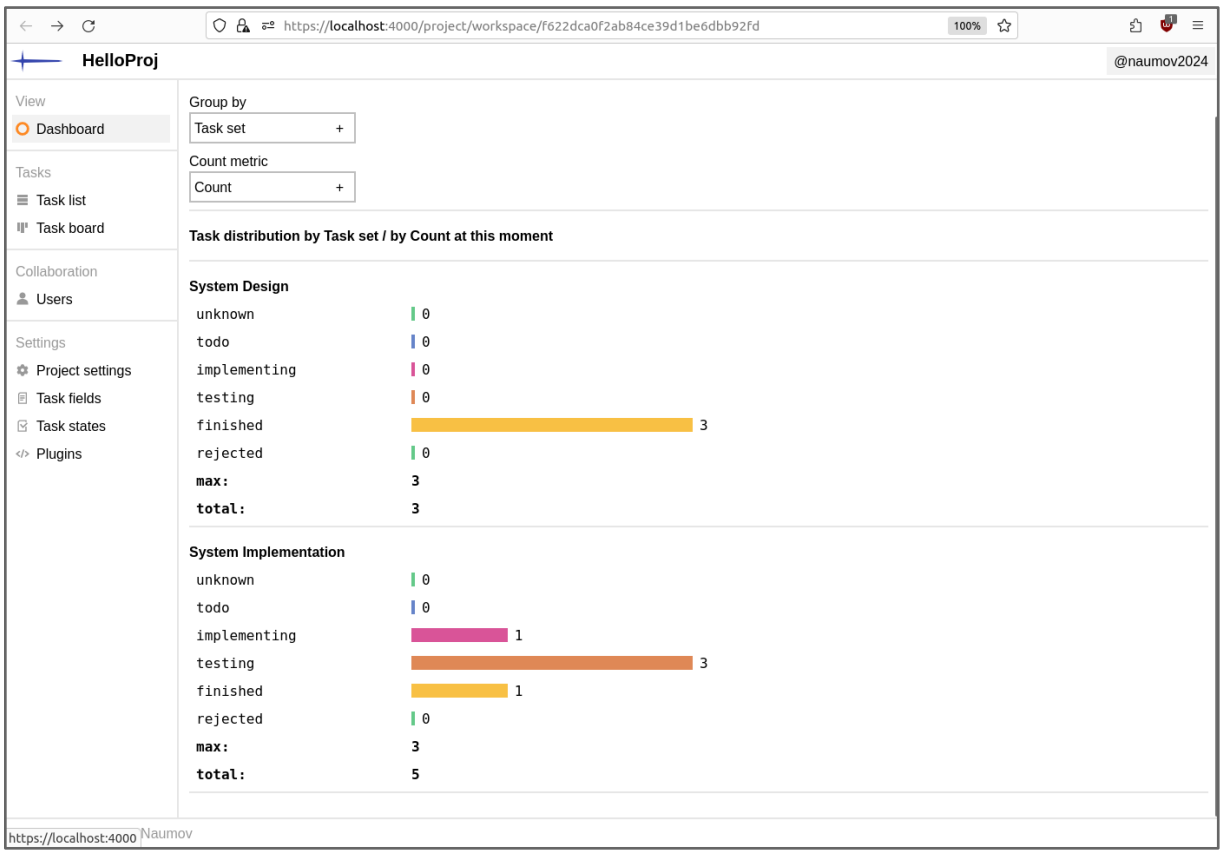


Рисунок 4.17 – Сторінка, що була додана за допомогою плагіна

4.4 Висновки до розділу 4

В цьому розділі наведено рекомендації щодо розгортання системи, наведено інструкції щодо найбільш часто виконуваних дій у клієнтському застосунку. Також наведено приклад розширення функціоналу клієнстького застосунку з використанням плагінів, що підтверджує працездатність та можливість розширення розробленого застосунку.

ВИСНОВКИ

В результаті роботи створено систему управління проєктами за методологією Agile у форматі web-застосунку.

Серверна частина застосунку реалізована з використанням JavaScript, Node.js та Express. Взаємодія сервера з реляційною базою даних реалізована з використанням ORM Objection.

Клієнтська частина застосунку реалізована з використанням JavaScript та фреймворку Vue. Взаємодія між клієнтом та сервером реалізована з використанням WebSocket для робочого простору проєкту, та з використанням HTTP у всіх інших випадках.

Розроблена система дозволяє створювати облікові записи користувачів, створювати проєкти, переглядати та редагувати проєкти разом з іншими користувачами в реальному часі, розширювати функціонал за допомогою плагінів.

У відповідності з поставленими задачами були виконані такі етапи створення застосунку:

- огляд існуючих систем управління проєктами та формування вимог до власної системи;
- проєктування архітектури, що включає моделювання варіантів використання, схеми даних, мережевої взаємодії та сторінок клієнтського застосунку;
- програмна реалізація сервера системи;
- програмна реалізація клієнтської частини системи;
- створення керівництва користувача, що включає рекомендації щодо розгортання, використання системи та розширення функціоналу.

Розроблена система представлена у вигляді вихідного коду сервера та клієнта, документації типів даних та API, шаблону плагіна для розширення користувацького інтерфейсу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Duncan W. R. A Guide to the Project Management Body of Knowledge. PA : Project Management Institute, 1996. 182 p.

2. Awad M. A. A Comparison between Agile and Traditional Software Development Methodologies. University of Western Australia, 2005. 84 p.

3. Boehm B. W. A spiral model of software development and enhancement. *IEEE Computer*. 1988. V. 21, № 5. P. 61–72.

4. Principles behind the Agile Software Development. URL: https://agilemanifesto.org/principles.html (дата звернення: 08.05.2024).

5. Jira Canban Boards. *Atlassian*. URL: https://www.atlassian.com/software/jira/features/kanban-boards (дата звернення: 08.05.2024).

6. Project Management & Agile Methodologies. *CPrime*. URL: https://www.cprime.com/resources/blog/project-management-agile-methodologies/ (дата звернення: 08.05.2024).

7. Extreme Programming Rules. *Extreme Programming*. URL: http://www.extremeprogramming.org/rules.html (дата звернення: 08.05.2024).

8. GitLab: The most-comprehensive AI-powered DevSecOps platform. *GitLab*. URL: https://about.gitlab.com/ (дата звернення: 09.05.2024).

9. GitHub: Let’s build from here. *GitHub*. URL: https://github.com/ (дата звернення: 09.05.2024).

10. Unlock your team’s best work with Jira Software. *Atlassian*. URL: https://jira.atlassian.com/ (дата звернення: 10.05.2024).

11. Manage your team’s projects from anywhere. *Trello*. URL: https://trello.com/en/ (дата звернення: 11.05.2024).

12. One app to replace them all. *ClickUp*. URL: https://clickup.com/ (дата звернення: 11.05.2024).

13. An overview of HTTP. *Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (дата звернення: 21.05.2024).

14. The WebSocket API. *Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 03.09.2024).

15. PostgreSQL: Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 03.09.2024).

16. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 04.09.2024).

17. JavaScript Reference. *Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference (дата звернення: 11.09.2024).

18. About Npm. *Npm Docs*. URL: https://docs.npmjs.com/about-npm (дата звернення: 12.09.2024).

19. Getting Started with ESLint. URL: https://eslint.org/docs/latest/use /getting-started (дата звернення: 13.09.2024).

20. Index – Node.js v18.20.5 Documentation. URL: https://nodejs.org /docs/latest-v18.x/api/index.html (дата звернення: 17.09.2024).

21. Express – Node.js Web Application Framework. URL: https://expressjs.com/ (дата звернення: 18.09.2024).

22. Objection.js. URL: https://vincit.github.io/objection.js/ (дата звернення: 19.09.2024).

23. Getting Started – Objection.js. URL: https://vincit.github.io /objection.js/guide/getting-started.html (дата звернення: 19.09.2024).

24. Schema builder – Knex.js. URL: https://knexjs.org/guide/schema-builder.html (дата звернення: 19.09.2024).

25. Dmitriynaumov1024/lib-better-obj: better objection.js. *GitHub*. URL: https://github.com/dmitriynaumov1024/lib-better-obj (дата звернення: 20.09.2024).

26. Introduction – Vue.js. URL: https://vuejs.org/guide/introduction.html (дата звернення: 25.09.2024).

27. Getting Started – Vue Router. URL: https://router.vuejs.org/guide/ (дата звернення: 26.09.2024).

28. Dmitriynaumov1024/proj: Project Management System. *GitHub*. URL: https://github.com/dmitriynaumov1024/proj (дата звернення: 22.10.2024).

29. Docker Compose. *Docker Docs*. URL: https://docs.docker.com/compose/ (дата звернення: 23.10.2024).

30. Dmitriynaumov1024/create-proj-plugin: Create a plugin for Proj. *GitHub*. URL: https://github.com/dmitriynaumov1024/create-proj-plugin (дата звернення: 24.10.2024).

ДОДАТОК А

Приклади вихідного коду сервера

А.1 Іменований модуль Project

```
import { SystemUnit } from "./__base.js"
import { base32id, base16id } from "common/utils/id"
import { filterFields } from "common/utils/object"
import { hash } from "common/utils/hash"
import { ProjectInfo as ProjectInfoModel, Project as ProjectModel } from "../database/models.js"
import { PermissionLevel as PL, InvolvementStatus as IS } from "../database/enums.js"

const pageSize = 20

export class Project extends SystemUnit
{
  /*
  Create project
  assume client logged in
  client does:
    sends POST ./project/create {
      session { id, token },
      project { title? }
    }
  server does:
    if (session not valid):
      return { success: false, notAuthorized: true }
    else if (user can not create project):
      return { success: false, forbidden: true }
    else:
      return {
        success: true,
        project: { id }
      }
  project.create
    + id = uuid()
    + createdAt = Date.now()
    + ownerId = context.user.id
    + title = request.project.title ?? "Untitled Project"
    + publicRead = false
    + publicClone = false
  */
  async create ({ project }, context) {
    let userOk = !!(context?.user?.id)
    if (!userOk) {
      return { success: false, notAuthorized: true }
    }

    let { database } = this.infrastructure
    let userCanCreateProject = true
    if (!userCanCreateProject) {
```

```
        return { success: false, forbidden: true }
    }

    let now = Date.now()
    let theProjectInfo = await database.projectInfo.query().insert({
        id: base16id(24) + (now & 0xffffff | 0x800000).toString(16),
        ownerId: context.user.id,
        title: ProjectInfoModel.rules.title.clamp(
            project?.title ?? ProjectInfoModel.rules.title.default
        ),
        createdAt: now,
        publicRead: false,
        publicClone: false
    })

    let theProject = ProjectModel.create().setParent(theProjectInfo)
    theProject = await database.project.query().insert(theProject)

    let { projectInvolvement } = await this.parent.projectInvolvement.create({
        project: { id: theProject.id },
        receiver: { id: context.user.id, permission: "owner" },
        sender: { id: context.user.id },
        verified: true,
        autoAccept: true
    })

    return {
        success: true,
        project: { id: theProjectInfo.id },
        projectInvolvement: projectInvolvement
    }
}

/*
Get list of own or related projects
assume project = { ownerId?, ownerUserName?, receiverStatus?, page? }
assume client logged in
client does:
    sends POST ./project/own-list {
        session { id, token },
        project { ownerId?, ownerUserName?, receiverStatus?, page? }
    }
server does:
    if (user not logged in):
        return { success: false, notAuthorized: true }
    let projects = get projects where
        sender status is not "rejected"
        and receiver id is user.id
        and optional receiverStatus = project.receiverStatus
        and optional ownerId = project.ownerId
                or ownerUserName = project.ownerUserName
        order by interactedAt descending
        paginate.
    return {
        success: true,
        projects: projects,
        page: page
```

```
    }
  */
  async getOwnList ({ project }, context) {
    project ??= { }
    if (!(project.page > 0)) project.page = 1

    let userOk = context?.user?.id
    if (!userOk) return { success: false, notAuthorized: true }

    let { database } = this.infrastructure

    let query = database.projectInvolvement.query()
      .withGraphJoined("[project.[owner]]")
      .whereNot("senderStatus", "=", IS.rejected)
      .where("receiverId", context.user.id)

    if (project.receiverStatus) {
      query = query.where("receiverStatus", "=", project.receiverStatus)
    }

    if (project.ownerId) {
      query = query.where("project.ownerId", "=", project.ownerId)
    }
    else if (project.ownerUserName) {
      query = query.where("project:owner.userName", "=", project.ownerUserName)
    }

    let projects = await query.orderBy("interactedAt", "desc", "last")
      .offset(pageSize*(project.page-1)).limit(pageSize)

    for (let p of projects) {
      // hide sensitive information
      if (p.project.owner) p.project.owner =
        filterFields(p.project.owner, ["id", "email", "displayName", "userName"])
    }

    return {
      success: true,
      projects: projects,
      page: project.page
    }
  }

}
```

A.2 Роутер /api/project

```
import { createRouter } from "better-express"
import { defineHandler } from "./__base.js"
import { InvolvementStatus as IS } from "../database/enums.js"

let route = createRouter()
```

```
route.post("/create", defineHandler(async (context)=> {
   let { system, body } = context
   let innerContext = { user: await system.auth.getUser({ session: body.session }) }
   let result = await system.project.create({ project: body.project }, innerContext)
   return result
}))

route.post("/own-list", defineHandler(async (context)=> {
   let { system, body } = context
   let innerContext = { user: await system.auth.getUser({ session: body.session }) }
   let result = await system.project.getOwnList({ project: body.project }, innerContext)
   return result
}))

export default { route }
```

А.3 Точка входу в серверний застосунок

```
import "dotenv/config"
const env = process.env

import fs from "node:fs"
fs.mkdirSync("./var", { recursive: true })

import { ConsoleLogger } from "better-logging"
let logger = new ConsoleLogger({
   color: env.BACKEND_LOGGER_COLOR=="true"
})

import { connections } from "./database/connect.js"
let dbConnection = connections[env.DB_PROVIDER]()

import { ProjDbAdapter } from "./database/database.js"
let dbAdapter = new ProjDbAdapter()
await dbAdapter.connect(dbConnection)
await dbAdapter.createDb()

import { mailSenders } from "./email/sender.js"
let createMailSender = mailSenders[env.MAIL_PROVIDER] || mailSenders.default
let mailer = createMailSender({
   apiKey: env.MAIL_API_KEY,
   secretKey: env.MAIL_API_SECRET,
   logger: logger
})

let familiar = {
   promo: {
      name: env.FAMILIAR_PROMO_NAME,
      email: env.FAMILIAR_PROMO_EMAIL
   },
   system: {
      name: env.FAMILIAR_SYSTEM_NAME,
```

```
        email: env.FAMILIAR_SYSTEM_EMAIL
    },
    app: {
        name: env.FAMILIAR_APP_NAME ?? "Proj"
    },
    name: env.FAMILIAR_NAME,
    email: env.FAMILIAR_EMAIL
}

let sockets = {
    connections: null
}

let cache = { }

import { System } from "./system/system.js"
let system = new System({
    services: {
        logger: ()=> logger,
        database: ()=> dbAdapter,
        cache: ()=> cache,
        mailer: ()=> mailer,
        familiar: ()=> familiar,
        sockets: ()=> sockets
    }
})

import { createServer } from "better-express"
let server = createServer({
    websocket: true,
    https: env.BACKEND_HTTPS=="true",
    key: env.BACKEND_HTTPS_KEY,
    cert: env.BACKEND_HTTPS_CERT
})

sockets.server = server.socket
system.events.emit("attachSockets", null)

import { staticServer } from "better-express"

let staticRootPath = "./dist"
try {
    server.http.use(staticServer("./dist", { maxAge: 60*1000 }))
    logger.log("Using static server for " + staticRootPath)
}
catch (error) {
    logger.log("Failed to set up static server for " + staticRootPath)
}

import { crossOrigin } from "better-express"
server.http.use(crossOrigin({ origins: "*" }))

import { requestLogger } from "./middleware/request-logger.js"
server.http.use(requestLogger(()=> logger))

let api = server.http.subpath("/api")
```

```
import { bodyParser } from "better-express"
api.use(bodyParser.text({ type: "application/json" }))

import { bodyTextToJson } from "./middleware/body-text-to-json.js"
api.use(bodyTextToJson())

import { requestItemProvider } from "./middleware/item-provider.js"
api.use(requestItemProvider(p=> {
   p.provide("system", ()=> system)
}))

import utils from "./webapi/utils.js"
api.use("/utils", utils.route)

import auth from "./webapi/auth.js"
api.use("/auth", auth.route)

import user from "./webapi/user.js"
api.use("/user", user.route)

import project from "./webapi/project.js"
api.use("/project", project.route)

// api fallback
api.use((request, response)=> {
   response.status(404).json({
      success: false,
      notFound: true,
      message: request.originalUrl+" not found"
   })
})

let indexHtml = null
try {
   let staticIndexPath = "./dist/index.html"
   indexHtml = fs.readFileSync(staticIndexPath)
   // static server fallback
   server.http.use((request, response, next)=> {
      if (request.method == "GET") {
         response.status(200)
         .append("Content-Type", "text/html")
         .send(indexHtml)
      }
      else next()
   })
   logger.log("Using static fallback " + staticIndexPath)
}
catch (error) {
   logger.log("Not using any static fallback")
}

import { errorCatcher } from "better-express"
server.http.app.use(errorCatcher(()=> logger))

let port = Number(env.BACKEND_PORT)

server.listen(port)
logger.log(`Server listening to port ${port}`)
```

А.4 Іменований модуль ProjectWorkspace

```
import { SystemUnit } from "./__base.js"
import { base32id, base16id } from "common/utils/id"
import { filterFields, nestedClone } from "common/utils/object"
import { PermissionLevel as PL, InvolvementStatus as IS } from "../database/enums.js"

export class ProjectWorkspace extends SystemUnit
{
   constructor (options) {
      super(options)
      this.initCache()
      this.registerWspHandlers()
      this.registerSystemHandlers()
      this.scheduleSocketConfiguration()
      this.schedulePersistProjects()
   }

   initCache () {
      this.services.cache.project ??= new Map()
   }

   registerWspHandlers () {
      let { logger, sockets } = this.services
      const wspHandlerPrefix = "onWsp"
      const wspEventPrefix = "wsp:"
      let handlerNames = Object.getOwnPropertyNames(this.constructor.prototype)
                        .filter(name=> name.startsWith(wspHandlerPrefix))
      for (let name of handlerNames) {
         logger.debug("wsp: using handler " + name)
         this.events.on(wspEventPrefix+name.slice(wspHandlerPrefix.length),                    (...args)=>
this[name](...args))
      }
   }

   registerSystemHandlers () {
      const systemHandlerPrefix = "onSystem"
      const systemEventPrefix = "system:"
      let handlerNames = Object.getOwnPropertyNames(this.constructor.prototype)
                        .filter(name=> name.startsWith(systemHandlerPrefix))
      for (let name of handlerNames) {
         this.events.on(systemEventPrefix+name.slice(systemHandlerPrefix.length),              (...args)=>
this[name](...args))
      }
   }

   schedulePersistProjects () {
      const projectPersistIntervalMs = 5000
      setTimeout(()=> this.events.emit("system:PersistProjects"), projectPersistIntervalMs)
   }

   scheduleSocketConfiguration () {
      this.events.on("attachSockets", ()=> this.configureSockets())
   }
```

```
  configureSockets () {
    let { logger, sockets } = this.services
    logger.log("Using sockets")
    sockets.server.on("connection", (connection)=> this.onRawConnect(connection))
  }

  /*
  assume connection = websocket connection
  */
  onRawConnect (connection) {
    let { sockets, logger } = this.infrastructure
    logger.log("wsp: new connection")
    sockets.connections ??= new Map()
    let wrapper = ({
      id: base32id(20),
      raw: connection,
      data: { },
      send (type, data) {
        this.raw.send(JSON.stringify({
          type, data,
          requestId: this.requestId,
          timestamp: Date.now()
        }))
      }
    })
    sockets.connections.set(wrapper.id, wrapper)
    connection.on("close", ()=> {
      logger.log("wsp: someone disconnected")
      this.events.emit("wsConnectionClosed", wrapper)
      sockets.connections.delete(wrapper.id)
    })
    connection.on("message", (data)=> {
      let parsedMessage = this.parseWsMessage(data, wrapper)
      logger.debug("wsp: "+parsedMessage.type)
      this.events.emit("wsp:"+parsedMessage.type, parsedMessage.data, Object.assign({ requestId:
parsedMessage.requestId }, wrapper))
    })
  }

  /*
  assume data = String
  */
  parseWsMessage (data, connWrapper) {
    try {
      let result = JSON.parse(data)
      if (!result.type) result.type = "TypelessMessage"
      return result
    }
    catch (error) {
      return { type: "BadMessage", data: data }
    }
  }

  /*
  convenient method to send message to
  every client connected to the project workspace
  */
```

```
broadcastForProject (projectId, type, data) {
    if (!projectId) return
    setTimeout(()=> {
        this.services.sockets.connections.forEach(con=> {
            if (con.data.project?.id != projectId) return
            con.send(type, data)
        })
    }, 0)
}

/*
assume session = { id, token }, project = { id }
*/
async onWspConnect ({ session, project }, context) {
    // validate request
    let requestOk = session?.id && session?.token && project?.id
    if (!requestOk) {
        return context.send("Connect.BadRequest", { })
    }

    // get user by session
    let user = await this.parent.auth.getUser({ session, wholeUser: true })
    if (!user) {
        return context.send("Connect.NotAuthorized", { })
    }

    // init user. later we can just check context.user without having to query database.
    user = filterFields(user, ["id", "displayName", "userName", "email"])
    context.data.user = user

    // get project
    let { cache, database } = this.services
    let theProject = cache.project.get(project.id)
    if (!theProject) {
        theProject = await database.projectInfo.query()
            .withGraphFetched("project")
            .where("id", project.id).first()
        if (!theProject) {
            return context.send("Connect.NotFound", { project })
        }
        cache.project.set(project.id, theProject)
        this.events.emit("system:SyncProjectUsers", { project: { id: theProject.id } })
    }

    // get project involvement
    if (!context.data.involvement) {
        context.data.involvement = await database.projectInvolvement.query()
            .where("projectId", project.id)
            .where("receiverId", user.id)
            .first()
        if (!context.data.involvement && theProject.publicRead) {
            context.data.involvement = { projectId: theProject.id, permission: PL.view }
        }
    }
    if (!context.data.involvement || context.data.involvement.permission == PL.none) {
        return context.send("Connect.Forbidden", { })
    }
```

```
    context.data.project = { id: theProject.id }

    context.send("Connect.Ok", { })

    context.send("User.Data", { user })

    return context.send("Project.Data", {
      id: theProject.id,
      title: theProject.title,
      data: theProject.project.data,
      plugins: theProject.project.plugins,
      // we don't send history here
    })
  }

  /*
  update project info like title and description.
  assume request project = { title, description }
  */
  async onWspUpdateProjectInfo ({ project }, context) {
    let pl = context.data.involvement?.permission
    let canUpdateInfo = pl == PL.admin || pl == PL.owner
    if (!canUpdateInfo) return context.send("Action.Forbidden", { })

    let { cache, sockets } = this.services
    let theProject = cache.project.get(context.data.project.id)
    if (!theProject) return context.send("Action.BadRequest", { })

    theProject.changedAt = Date.now()
    if (Object.hasOwn(project, "title")) theProject.title = project.title
    if (Object.hasOwn(project, "description")) theProject.project.data.description = project.description

    this.broadcastForProject(theProject.id, "Project.DataPatch", {
      title: theProject.title,
      data: {
        description: theProject.project.data.description
      }
    })
  }

  async onWspUpdateProjectTaskStatuses ({ taskStatuses }, context) {
    let pl = context.data.involvement?.permission
    let canUpdateInfo = pl == PL.admin || pl == PL.owner
    if (!canUpdateInfo) return context.send("Action.Forbidden", { })

    let { cache, sockets } = this.services
    let theProject = cache.project.get(context.data.project.id)
    if (!theProject) return context.send("Action.BadRequest", { })

    theProject.project.data.taskStatuses = nestedClone(taskStatuses)
    theProject.changedAt = Date.now()

    this.broadcastForProject(theProject.id, "Project.DataPatch", {
      data: {
        taskStatuses: {
          $rewrite: true,
          ...taskStatuses
```

```
            }
        }
    })

    this.broadcastForProject(theProject.id, "Connect.NeedsRestart", { })
}

async onWspUpdatePlugins ({ update, reorder, toggle, remove }, context) {
    let pl = context.data.involvement?.permission
    let canUpdateInfo = pl == PL.admin || pl == PL.owner
    if (!canUpdateInfo) return context.send("Action.Forbidden", { })

    let { cache, sockets } = this.services
    let theProject = cache.project.get(context.data.project.id)
    if (!theProject) return context.send("Action.BadRequest", { })

    if (update) {
        let plugins = update
        for (let key in plugins) {
            theProject.project.plugins[key] = plugins[key]
        }
        theProject.changedAt = Date.now()
    }
    else if (reorder) {
        let oldPlugins = theProject.project.plugins
        theProject.project.plugins = []
        for (let i in reorder) theProject.project.plugins[i] = oldPlugins[reorder[i]]
        theProject.changedAt = Date.now()
    }
    else if (toggle) {
        for (let p of toggle) {
            let thePlugin = theProject.project.plugins.find(plugin=> plugin.id==p.id)
            if (thePlugin) thePlugin.enabled = p.enabled
        }
        theProject.changedAt = Date.now()
    }
    else if (remove) {
        // assume remove = { id }
        theProject.project.plugins = theProject.project.plugins.filter(p=> p.id != remove.id)
        theProject.changedAt = Date.now()
    }

    this.broadcastForProject(theProject.id, "Connect.NeedsRestart", { })
}

async onWspUpdateTask ({ add, update, remove }, context) {
    let pl = context.data.involvement?.permission
    let canUpdateInfo = pl == PL.edit || pl == PL.admin || pl == PL.owner
    if (!canUpdateInfo) return context.send("Action.Forbidden", { })

    let { cache, sockets } = this.services
    let theProject = cache.project.get(context.data.project.id)
    if (!theProject) return context.send("Action.BadRequest", { })

    let before = { }, after = { }

    if (add) {
```

```
      after[add.id] = add
    }
    else if (update) {
      before[update.id] = theProject.project.data.taskObjects[update.id]
      after[update.id] = update
    }
    else if (remove) {
      before[remove.id] = theProject.project.data.taskObjects[remove.id]
    }
    else {
      return false
    }

    theProject.changedAt = Date.now()

    this.events.emit("system:CommitTaskObjects", {
      project: theProject.project,
      commit: {
        before, after,
        createdAt: Date.now()
      },
    })
}

onSystemCommitTaskObjects ({ commit, project }) {
    project.history.commits ??= [ ]
    project.history.commits.push(commit)

    for (let key in commit.before) {
      if (!commit.after[key]) delete project.data.taskObjects[key]
    }
    for (let key in commit.after) {
      project.data.taskObjects[key] = commit.after[key]
    }

    this.broadcastForProject(project.id, "Project.PatchTaskObjects", commit)
}

async onSystemSyncProjectUsers ({ project }) {
    let { logger, cache, database, sockets } = this.services
    let theProject = cache.project.get(project.id)
    if (!theProject) return

    let involvements = await database.projectInvolvement.query()
      .withGraphJoined("receiver")
      .where("projectId", project.id)

    let users = involvements.map(u=> ([u.receiver.id, {
      id: u.receiver.id,
      userName: u.receiver.userName,
      displayName: u.receiver.displayName,
      email: u.receiver.email,
      permission: u.permission
    }]))

    theProject.project.data.users = Object.fromEntries(users)
    theProject.changedAt = Date.now()
```

```
    this.broadcastForProject(theProject.id, "Project.DataPatch", {
      data: {
        users: {
          $rewrite: true,
          ...theProject.project.data.users
        }
      }
    })
  }

  onSystemDeleteProject ({ project }) {
    let cache = this.services.cache
    cache.project.delete(project.id)
    this.broadcastForProject(project.id, "Connect.NeedsRestart", { })
  }

  onSystemCreateInvolvement ({ project }) {
    // re-emit event
    this.events.emit("system:SyncProjectUsers", { project })
  }

  onSystemUpdateInvolvement ({ project }) {
    // re-emit event
    this.events.emit("system:SyncProjectUsers", { project })
  }

  async onSystemPersistProjects () {
    let { cache, database, logger } = this.services
    let count = 0
    for (let [key, project] of cache.project.entries()) {
      let projectChangedAt = project.changedAt?? 0,
        projectPersistedAt = project.persistedAt?? 0
      if (projectChangedAt > projectPersistedAt) {
        await database.projectInfo.query().where("id", project.id)
          .patch(filterFields(project, ["title", "publicRead", "publicClone"]))
        await database.project.query().where("id", project.id)
          .patch(project.project)
        if (project.changedAt > Date.now()) project.changedAt = Date.now() - 1
        project.persistedAt = Date.now()
        count++
      }
    }
    if (count > 0) logger.log(`Persisted ${count} projects`)
    this.schedulePersistProjects()
  }

  onWspTypelessMessage (data) {
    let { logger } = this.infrastructure
    logger.debug("wsp: typeless message")
  }

  onWspBadMessage (data) {
    let { logger } = this.infrastructure
    logger.debug("wsp: bad message")
  }

  onWspPing (data, context) {
```

```
        let { logger } = this.infrastructure
        context.send("Pong", "pong!")
    }

}
```

ДОДАТОК Б

Приклади вихідного коду клієнта

Б.1 Сторінка project/info

```
import { h } from "vue"
import { RouterLink } from "vue-router"
import HeaderLayout from "@/comp.layout/header.js"
import FooterLayout from "@/comp.layout/footer.js"
import Modal from "@/comp.layout/modal.js"
import StepperBox from "@/comp.ctrl/stepperbox.js"

import { idToColor } from "@/lib/utils.js"
import { timestampToDayMonthYear, getTimeZone } from "@/lib/utils.js"
const tz = getTimeZone()

const PLs = ["none", "view", "comment", "edit", "admin", "owner"]

function ProjectInfoView(self) {
   const loc = self.$locale.current
   const canManage = self.involvement?.permission == "admin"
            || self.involvement?.permission == "owner"
   const canDelete = self.involvement?.receiverId == self.project?.ownerId
   const canSeeWorkspace = self.involvement?.permission != "none"
   return (self.project instanceof Object)? [
      // project info
      h("div", { class: ["bv", "pad-05-0"] }, [
         h("div", { style: { "background-color": self.project.idColor, "color": "white" }, class: ["mar-b-05",
"pad-1-05"] }, [,
            h("h2", { }, self.project.title || `${loc.project.self} #${self.project.id}`)
         ]),
         h("p", { class: ["mar-b-05"] }, `${loc.project.createdAt}:
${timestampToDayMonthYear(self.project.createdAt, tz)}`),
         // own involvement info
         (self.involvement instanceof Object)? [
            self.involvement.interactedAt?
            h("p", { class: ["mar-b-05"] }, `${loc.project.interactedAt}:
${timestampToDayMonthYear(self.involvement.interactedAt, tz)}`) :
            h("p", { class: ["mar-b-05"] }, `${loc.project.invitedAt}:
${timestampToDayMonthYear(self.involvement.sentAt, tz)}`),
            h("p", { class: ["mar-b-05"] }, `${loc.project.permission.self}:
${loc.project.permission[self.involvement.permission]}`),
            canSeeWorkspace?
            h("p", { class: ["mar-b-05"] }, h(RouterLink, { to: "/project/workspace/"+self.id, target:
"_blank" }, ()=>loc.project.goToWorkspace+" >>")) : null
         ] : null,
         // delete project
         canDelete?
         h("a", { class:["color-bad"], onClick: ()=> self.onBeginDeleteProject() }, `${loc.action.delete}
${loc.project.self}`) : null
      ]),
```

```
      // users involved in the project
      (self.users instanceof Array)? h("div", { class: ["bv", "pad-05-0"] }, [
         h("div", { class: ["flex-stripe", "mar-b-05"] }, [
            h("h2", { class: ["flex-grow"] }, loc.user.plural),
            canManage?
               h("button", { class: ["pad-025-05"],
               onClick: ()=> self.onBeginAddUser() }, `+ ${loc.project.addUser}`) :
               null,
         ]),
         self.users.map(u=> h("div", { class:["mar-b-05", "user-card", "flex-stripe", "flex-pad-05"] }, [
            h("div", { style: { "background-color": u.idColor, "width": "2.5rem", "flex-shrink": 0, "text-align":
"center", "color": "white" } }, h("h2", { }, u.user.userName[0]?.toUpperCase())),
            h("div", { class: ["flex-grow"] }, [
               h("p", { }, h("b", { }, `${u.user.displayName ?? loc.user.self} @${u.user.userName}`)),
               h("p", { class: ["color-gray"] }, `${loc.user.email}: ${u.user.email}`),
               u.interactedAt?
               h("p", { class: ["color-gray"] }, `${loc.project.interactedAt}:
${timestampToDayMonthYear(u.interactedAt, tz)}`) :
               h("p", { class: ["color-gray"] }, `${loc.project.invitedAt}:
${timestampToDayMonthYear(u.sentAt, tz)}`),
               h("p", { class: ["color-gray", "mar-b-05"] }, `${loc.project.permission.self}:
${loc.project.permission[u.permission]}`)
            ])
         ]))
      ]) : null
   ] : null
}

function AddingUserModal(self) {
   const loc = self.$locale.current
   return h(Modal, { titleText: loc.project.addUser,
      onClickOutside: ()=> self.onCompleteAddUser(false) }, ()=> [
      h("div", { class: ["mar-b-05"] }, [
         h("p", { }, loc.user.userName),
         h("input", { class: ["block",], placeholder: "user",
            value: self.addingUser.userName,
            onChange: (e)=> self.addingUser.userName = e.target.value,
            invalid: !!self.addingUser.bad
         }),
      ]),
      h("div", { class: ["mar-b-05"] }, [
         h("p", { }, loc.project.permission.self),
         h(StepperBox, { class: ["flex-stripe"],
            text: loc.project.permission[PLs[self.addingUser.pLevel??0]],
            min: 0, max: PLs.length-1, step: 1, value: self.addingUser.pLevel??0,
            onChange: (value)=> self.addingUser.pLevel = value
         }),
      ]),
      h("button", { class: ["block"], onClick: ()=> self.onCompleteAddUser(true) }, loc.action.proceed)
   ])
}

function DeletingProjectModal(self) {
   const loc = self.$locale.current
   return h(Modal, { titleText: `${loc.action.delete} ${loc.project.self}`,
      onClickOutside: ()=> self.onCompleteDeleteProject(false) }, ()=> [
      h("div", { class: ["mar-b-1"] }, [
```

```
            h("p", { }, loc.project.confirmDelete),
        ]),
        h("div", { class: ["flex-stripe", "flex-pad-05"] }, [
            h("button", { class: ["flex-grow"], onClick: ()=> self.onCompleteDeleteProject(false) },
loc.action.cancel),
            h("button", { class: ["flex-grow", "color-bad"], onClick: ()=> self.onCompleteDeleteProject(true) },
h("b", { }, loc.action.delete))
        ])
    ])
}


export default {
    props: {
        id: String
    },
    data() {
        return {
            project: null,
            owner: null,
            involvement: null,
            users: null,
            errorMessage: null,
            notAuthorized: null,
            addingUser: null,
            deletingProject: false,
        }
    },
    watch: {
        id: {
            immediate: true,
            handler (newValue, oldValue) {
                if (!newValue) this.errorMessage = this.$locale.current.error.notFound
                else if (newValue!=oldValue) this.getProject()
            }
        }
    },
    methods: {
        async getProject() {
            this.errorMessage = null
            this.notAuthorized = null
            let result = await this.$http.invoke("/project/find", { project: { id: this.id } })
            if (result.success) {
                result.project.idColor = await idToColor(result.project.id)
                this.project = result.project
                this.involvement = result.projectInvolvement
                if (this.project.ownerId) await this.getProjectOwner()
                await this.getProjectUsers()
            }
            else if (result.notAuthorized) {
                this.notAuthorized = true
                this.errorMessage = this.$locale.current.error.notAuthorized
            }
            else if (result.notFound) {
                this.notFound = true
                this.errorMessage = this.$locale.current.error.notFound
            }
```

```
},
async getProjectOwner() {
    let result = await this.$http.invoke("/user/find", { user: { id: this.project.ownerId } })
    if (result.success) {
        this.owner = result.user
    }
},
async getProjectUsers() {
    let result = await this.$http.invoke("/user/in-project", { project: { id: this.project.id } })
    if (result.success) {
        for (let u of result.users) {
            u.idColor = await idToColor(u.receiverId)
        }
        this.users = result.users
    }
},
async addUserToProject() {
    if (this.users.find(u=> u.receiver.userName == this.addingUser.userName)) {
        this.addingUser = null
        return
    }
    let { user } = await this.$http.invoke("/user/find", {
        user: { userName: this.addingUser.userName }
    })
    if (!user) {
        this.addingUser.bad = true
        return
    }
    let result = await this.$http.invoke("/project/involvement/create", {
        project: { id: this.project.id, permission: PLs[this.addingUser.pLevel] },
        receiver: { id: user.id, },
        sender: { id: this.involvement.receiverId }
    })
    if (result.success) {
        this.addingUser = null
        this.getProjectUsers()
    }
    else if (result.exists) {
        this.addingUser = null
    }
    else {
        this.addingUser.bad = true
    }
},
async deleteProject() {
    let result = await this.$http.invoke("/project/delete", {
        project: { id: this.id }
    })
    this.deletingProject = false
    if (result.success) {
        this.$router.push("/")
    }
},
onBeginAddUser() {
    this.addingUser = { }
},
onCompleteAddUser(confirm) {
```

```
            if (confirm) this.addUserToProject()
            else this.addingUser = null
        },
        onBeginDeleteProject() {
            this.deletingProject = true
        },
        onCompleteDeleteProject(confirm) {
            if (confirm) this.deleteProject()
            else this.deletingProject = false
        }
    },
    render() {
        const loc = this.$locale.current
        return h("div", { class: ["ww", "h100", "scroll"] }, [
            h(HeaderLayout, { }, ()=> [
                h("h2", { class: ["clickable"], onClick: ()=> this.$router.push("/") }, loc.app.name)
            ]),
            h("div", { class: ["bv", "hmin70"] }, [
                h("div", { class: ["bv"] }, [
                    h("div", { class: ["wc", "pad-05"] }, [
                        this.project?
                            ProjectInfoView(this) :
                        this.errorMessage?
                            h("p", { class: ["color-bad"] }, this.errorMessage) :
                            [ h("h3", { class: ["mar-b-05"] }, `${loc.project.self} #${this.id}`),
                              h("p", { }, loc.common.loading) ],
                        this.notAuthorized? [
                            h("p", { }, h(RouterLink, { to: "/login" }, ()=> loc.action.login)),
                            h("p", { }, h(RouterLink, { to: "/signup" }, ()=> loc.action.signup)),
                        ] : null,
                        this.addingUser?
                            AddingUserModal(this) : null,
                        this.deletingProject?
                            DeletingProjectModal(this) : null
                    ]),
                ])
            ]),
            h(FooterLayout)
        ])
    }
}
```

Б.2 Сторінка project/workspace

```
import { h, createApp, markRaw as m } from "vue"
import { installable } from "@/lib/installable.js"
import { RouterLink } from "vue-router"
import HeaderLayout from "@/comp.layout/header.js"
import FooterLayout from "@/comp.layout/footer.wsp.js"

import { createSocket } from "@/lib/socket.js"
import { createTempStorage } from "@/lib/storage.js"
```

```
import { nestedAssign } from "common/utils/object"
import { base16id } from "common/utils/id"
import { TaskObjectType as TOType } from "common/wsp/enums"
import { getTimeZone } from "@/lib/utils.js"

import * as ico from "@/comp.icon/index.js"
import Menu from "./comp/menu.js"

import pages from "./pages/index.js"

const defaultRoute = "index"
const notFoundRoute = "notFound"

import WorkspaceApp from "./logic/workspace-app.js"

import * as $vue from "vue"

const WorkspaceAppTemplate = {
   data() {
      return {
         connection: {
            ok: false,
            error: true,
            message: null
         },
         needsRestart: false,
         activeMenuItem: null,
         expandMenu: false,
         route: defaultRoute,
         query: { }
      }
   },
   mounted() {
      this.setupConnectionHandlers()
      this.connectToProject()
   },
   methods: {
      setupConnectionHandlers() {
         let socket = this.$socket
         socket.on("Connect.Ok", ()=> {
            this.connection.ok = true
            this.connection.message = "Connected successfully! :D"
            this.setupApp()
         })
         socket.on("Connect.NotAuthorized", ()=> {
            this.connection.error = true
            this.connection.message = "Not authorized to connect to this project :("
         })
         socket.on("Connect.NotFound", ()=> {
            this.connection.error = true
            this.connection.message = "Project does not exist :("
         })
         socket.on("Connect.Forbidden", ()=> {
            this.connection.error = true
            this.connection.message = "Either banned from project or project does not allow public read."
         })
         socket.on("Connect.NeedsRestart", () => {
```

```
        this.needsRestart = true
    })
},
setupApp() {
    let app = this.$app
    // set up app menu
    app.menu = WorkspaceApp.menu()
    // set up app pages
    app.pages = WorkspaceApp.pages()
    // set up methods
    app.methods = WorkspaceApp.methods()
    // set up data sources
    app.dataSources = WorkspaceApp.dataSources()
    // other stuff
    app.navigate = m((route, query)=> this.navigate(route, query))
    app.tz = getTimeZone()
    app.ready = true
},
async createPlugins() {
    console.log("creating plugins...")
    let theImports = {
        vue: $vue,
        // add more imports here if necessary
    }
    let theRequire = (name)=> {
        if (Object.hasOwn(theImports, name)) return theImports[name]
        else throw new Error(`require(): could not find import '${name}'`)
    }
    this.$storage.plugins ??= [ ]
    this.$storage.pluginscss ??= [ ]
    for (let plugin of this.$storage.project.plugins) {
        if (!plugin.enabled) continue
        let theExports = { }
        try {
            let css = await this.getPluginCss(plugin)
            if (!!css) this.$storage.pluginscss.push(css)
        }
        catch (error) {
            console.error(`Failed to load css for '${plugin.name||plugin.id}'`)
            console.error(error)
        }
        try {
            let code = "let { require, imports, exports } = options;\n" + await this.getPluginCode(plugin)
            let pluginFactory = new Function("options", code)
            pluginFactory({
                require: (name)=> theImports[name],
                imports: theImports,
                exports: theExports
            })
            theImports[plugin.name] = theExports.default
            this.$storage.plugins.push(theExports.default || { })
        }
        catch (error) {
            console.error(`Failed to create plugin '${plugin.name||plugin.id}'`)
            console.error(error)
        }
    }
```

```
},
async installPlugins() {
    console.log("installing plugins...")
    let theApp = {
        ...this.$app,
        storage: this.$storage,
        locale: this.$locale,
        socket: this.$socket,
        http: this.$http
    }
    for (let plugin of this.$storage.plugins) {
        if (plugin?.install instanceof Function) await plugin.install(theApp)
    }
},
async mountPlugins() {
    console.log("mounting plugins...")
    let theApp = {
        ...this.$app,
        storage: this.$storage,
        locale: this.$locale,
        socket: this.$socket,
        http: this.$http
    }
    for (let plugin of this.$storage.plugins) {
        if (plugin?.mount instanceof Function) await plugin.mount(theApp)
    }
},
async getPluginCode (plugin) {
    if (plugin.type == "inline") {
        console.log("loading inline plugin ok")
        return plugin.code
    }
    else {
        console.warn("loading external plugins not impl. yet")
        return ""
    }
},
async getPluginCss (plugin) {
    if (plugin.type == "inline") {
        return plugin.style
    }
    else {
        console.warn("loading external plugins' stylesheets not impl. yet")
        return ""
    }
},
async connectToProject() {
    await this.$socket.ready()
    this.$socket.on("Project.Data", async (data)=> {
        this.$storage.project = data
        window.getProject = ()=> this.$storage.project
        await this.createPlugins()
        await this.installPlugins()
        await this.mountPlugins()
    })
    this.$socket.on("Project.DataPatch", (data)=> {
        nestedAssign(this.$storage.project, data)
```

```
        })
        this.$socket.on("Project.PatchTaskObjects", (data)=> {
          let { before, after } = data
          let project = this.$storage.project
          for (let key in before) {
            if (!after[key]) delete project.data.taskObjects[key]
          }
          for (let key in after) {
            project.data.taskObjects[key] = after[key]
          }
        })
        this.$socket.on("User.Data", (data)=> {
          this.$storage.user = data.user
        })
        this.$socket.send("Connect", {
          session: this.$http.session,
          project: { id: this.$env.projectId }
        })
      },
      navigate (route, query) {
        this.route = route
        this.query = query?? { }
      },
      onMenuClick (item) {
        this.activeMenuItem = item
        this.expandMenu = false
        this.route = item?.route
        this.query = item?.query?? { }
      }
    },
    render() {
      const loc = this.$locale.current
      const connection = this.connection
      const { project, user } = this.$storage
      const app = this.$app
      return h("div", { class: ["h100", "flex-v"] }, [
        this.$storage.pluginscss?.map(css=> h("style", { }, css)),
        h(HeaderLayout, { hless: true, wfull: true, style: {"flex-shrink": 0} }, ()=> [
          h("div", { class: ["clickable"], active: this.expandMenu, role: "wsp-menu-handle", onClick: ()=>
this.expandMenu = !this.expandMenu }, [
            h(ico.MenuHandleIcon, { class: ["icon-20"] })
          ]),
          h("h2", { class: ["clickable"], onClick: ()=> this.navigate(defaultRoute) }, project?.title ??
loc.workspace.self),
          h("span", { class: ["flex-grow"] }, ""),
          (user instanceof Object)?
          h("span", { class: ["shaded", "if-ws", "pad-025-05"] }, "@"+user.userName) : null
        ]),
        h("div", { class: ["bv", "flex-grow", "scroll"] }, [
          connection.ok?
          h("div", { class: ["flex-stripe", "h100"] }, [
            app.ready? [
              h(Menu, { expand: this.expandMenu, items: app.menu, activeItem: (item)=> (this.route ==
item.route), onClick: (item)=> this.onMenuClick(item) }),
              h("div", { class: ["wsp-main", "pad-05"] }, [
                this.needsRestart?
                h("p", { class: ["mar-b-1", "pad-025", "bb"] }, [
```

```
                        "Project workspace needs to be reloaded to sync changes. ",
                        h("a", { class: ["color-bad"], onClick: ()=> this.$reload() }, "Reload")
                    ]) : null,
                    h(app.pages[this.route]?? app.pages[notFoundRoute], { parent: this, query: this.query })
                ])
            ] : h("p", { }, "Preparing app UI...")
        ]) :
        connection.error?
        h("div", { class: ["text-center", "pad-05"] }, [
            h("p", { class: ["color-bad"] }, connection.message),
            h("p", { }, [ h("a", { href: "/" }, "go to main page") ])
        ]) :
        h("p", { }, "Connecting to workspace, please wait...")
    ]),
    h(FooterLayout, { style: {"flex-shrink": 0}, hless: true, wfull: true })
  ])
 }
}

let nestedApps = { }
function createNestedApp(id, parent) {
 if (!id) return

 let socket = createSocket(id)

 let app = createApp(WorkspaceAppTemplate)
 nestedApps[id] = app

 let appStorage = createTempStorage("app")
 let storage = createTempStorage("project")

 app.use(installable("$env", { projectId: id  }))
 app.use(installable("$socket", socket))
 app.use(installable("$http", parent.$http))
 app.use(installable("$locale", parent.$locale))
 app.use(installable("$app", appStorage))
 app.use(installable("$storage", storage))
 app.use(installable("$reload", ()=> parent.onSwitchWorkspace(id, id) ))
 return app
}

function disposeNestedApp(id) {
 if (!id) return
 nestedApps[id]?.unmount()
 delete nestedApps[id]
}

export default {
 props: {
  id: String
 },
 watch: {
  id (newValue, oldValue) {
   if (oldValue && (newValue != oldValue)) {
    this.onSwitchWorkspace(newValue, oldValue)
   }
  }
```

```
    },
    methods: {
        onSwitchWorkspace(id, old) {
            if (!!old) disposeNestedApp(old)
            let app = createNestedApp(id, this)
            app.mount(this.$refs.nestedAppRoot)
        }
    },
    mounted() {
        this.onSwitchWorkspace(this.id)
    },
    render() {
        return h("div", { class: ["h100"], ref: "nestedAppRoot" })
    }
}
```

Б.3 Вкладена сторінка settings.project

```
import { h } from "vue"
import { nestedClone } from "common/utils/object"
import { PermissionLevel as PL } from "common/wsp/enums"

export default {
    props: {
        parent: Object
    },
    data() {
        return {
            before: { },
            after: { },
            changed: false,
            canEdit: false
        }
    },
    mounted() {
        let { project, user } = this.$storage
        this.before = { title: project.title, description: project.data.description }
        this.after = nestedClone(this.before)
        let perm = project.data.users[user.id]?.permission
        if (perm == PL.admin || perm == PL.owner) this.canEdit = true
    },
    methods: {
        async saveChanges() {
            await this.$socket.ready()
            this.$socket.send("UpdateProjectInfo", {
                project: this.after
            })
            this.changed = false
        },
        onReset() {
            this.after = nestedClone(this.before)
            this.changed = false
```

```
        },
        onSaveChanges() {
            this.saveChanges()
        }
    },
    render() {
        let loc = this.$locale.current
        return h("div", { class: ["pad-025", "wwc", "stick-left"] }, [
            h("h3", { class: ["mar-b-05"] }, "Settings"),
            h("div", { class: ["mar-b-05"] }, [
                h("p", { }, "Title"),
                h("input", { class: ["block"],
                    readonly: !this.canEdit,
                    value: this.after.title,
                    onInput: (e)=> { this.after.title = e.target.value; this.changed = true }
                }),
            ]),
            h("div", { class: ["mar-b-05"] }, [
                h("p", { }, "Description"),
                h("textarea", { class: ["block", "height-10"],
                    readonly: !this.canEdit,
                    value: this.after.description,
                    onInput: (e)=> { this.after.description = e.target.value; this.changed = true }
                })
            ]),
            (this.changed && this.canEdit)?
            h("div", { class: ["mar-b-05", "flex-stripe", "flex-pad-05"] }, [
                h("button", { class: ["flex-grow", "color-bad"], onClick: ()=> this.onReset() }, loc.action.reset),
                h("button", { class: ["flex-grow"], onClick: ()=> this.onSaveChanges() }, loc.action.save)
            ]) : null
        ])
    }
}
```

Б.4 Налаштування Vue-Router

```
import { defineRoutes } from "@/lib/router.js"

import IndexPage from "@/pages/index.js"
import IntroPage from "@/pages/intro.js"

import LoginPage from "@/pages/login.js"
import SignupPage from "@/pages/signup.js"
import MyProfilePage from "@/pages/user/me.js"
import UserListPage from "@/pages/user/list.js"
import UserInfoPage from "@/pages/user/info.js"

import ProjectListPage from "@/pages/project/list.js"
import ProjectInfoPage from "@/pages/project/info.js"
import ProjectWorkspacePage from "@/pages/project/workspace.js"

import NotFoundPage from "@/pages/__notfound.js"
```

```
export const routes = defineRoutes([
    // index pages
    { path: "/", component: IndexPage },
    { path: "/index", component: IndexPage },
    { path: "/intro", component: IntroPage },
    // user profile related pages
    { path: "/login", component: LoginPage },
    { path: "/signup", component: SignupPage },
    { path: "/register", component: SignupPage },
    { path: "/me", component: MyProfilePage },
    { path: "/user/discover", component: UserListPage, props: { query: String } },
    { path: "/user/info/:userName", component: UserInfoPage, props: { userName: String } },
    // project pages
    { path: "/project/list", component: ProjectListPage, props: { page: Number } },
    { path: "/project/info/:id", component: ProjectInfoPage, props: { id: String } },
    { path: "/project/workspace/:id", component: ProjectWorkspacePage, props: { id: String } },
    // fallback
    { path: "/:path(.*)", component: NotFoundPage, props: { path: String } },
])
```

ДОДАТОК В

Розширення функціоналу системи

В.1 Приклад вихідного коду плагіна

src/main.js:

```
// dashboard-plugin@0.0.1
// created by create-proj-plugin

import { h } from "vue"
import { markRaw as m } from "vue"

import DashboardPage from "./Dashboard.vue"

const myCircleIcon = ()=> h("svg", { viewBox: "0 0 10 10" }, [
  h("circle", { cx: 5, cy: 5, r: 4, stroke: "#fe8923", "stroke-width": "1.8", fill: "none", })
])

export default {
   mount (app) {
      let route = "dashboard.td"
      let submenu = app.menu.find(category => category.name == "View")
      if (!submenu) {
         submenu = { name: "View", items: [] }
         app.menu.unshift(submenu)
      }
      submenu.items.push({ name: "Dashboard", route: route, icon: m(myCircleIcon) })
      app.pages[route] = m(DashboardPage)
   }
}

import "./style.scss"
```

src/Dashboard.vue:

```
<template>
   <div class="pad-025" plugin="dashboard-plugin">
      <div class="mar-b-1 bb">
         <h3 class="mar-b-1">Dashboard</h3>
         <div class="mar-b-05">
            <p>Group by</p>
            <Selectbox style="width: 12rem;"
               :items="this.groupByOptions"
               :value="this.groupBy"
               @change="value=> this.groupBy = value"/>
         </div>
         <div class="mar-b-05">
            <p>Count metric</p>
            <Selectbox style="width: 12rem;"
               :items="this.countMetricOptions"
               :value="this.countMetric"
```

```
                @change="value=> this.countMetric = value"/>
            </div>
        </div>
        <div class="mar-b-1 bb">
            <h3 class="mar-b-1">Task distribution by {{this.groupByOptions.find(i=>
i.value==this.groupBy)?.text}} / by {{this.countMetricOptions.find(i=>
i.value==this.countMetric)?.text}} at this moment</h3>
        </div>
        <template v-for="g in this.taskDistribution">
            <div class="mar-b-1 bb">
                <p><b>{{g.head.text}}</b></p>
                <div class="pad-05 text-mono">
                    <template v-for="([status, count], i) in g.items.distribution">
                        <div class="dashboard-row">
                            <div style="width: 15rem;" class="one-line">{{status}}</div>
                            <div :style="{ height: '1rem', width: (0.25+20*count/Math.max(g.items.max, 1))+'rem',
backgroundColor: this.colors[i%this.colors.length] }"></div>
                            <div>{{count}}</div>
                        </div>
                    </template>
                    <div class="dashboard-row">
                        <div style="width: 15rem;" class="one-line"><b>max:</b></div>
                        <div><b>{{g.items.max}}</b></div>
                    </div>
                    <div class="dashboard-row">
                        <div style="width: 15rem;" class="one-line"><b>total:</b></div>
                        <div><b>{{g.items.total}}</b></div>
                    </div>
                </div>
            </div>
        </template>
    </div>
</template>

<script>
import Selectbox from "./Selectbox.js"
import { markRaw as m } from "vue"

const countMetricOptions = [
    { value: "count", text: "Count" },
    { value: "points", text: "Points" },
    { value: "hours", text: "Hours" },
]

const groupByOptions = [
    { value: "assigned", text: "Assigned" },
    { value: "taskset", text: "Task set" },
]

const groupingMethods = {
    "assigned"(tasks) {
        let result = { }
        let usersSource = this.app.dataSources.userIds
        for (let task of tasks) {
            result[task.assigned] ??= { head: usersSource.convertItem(usersSource.restoreItem.call(this,
task.assigned)), items: [] }
            result[task.assigned].items.push(task)
```

```
        }
        return result
    },
    "taskset"(tasks) {
        let result = { }
        let taskSetSource = this.app.dataSources.taskSetIds
        for (let task of tasks) {
            result[task.taskSetId] ??= { head: taskSetSource.convertItem(taskSetSource.restoreItem.call(this,
task.taskSetId)), items: [] }
            result[task.taskSetId].items.push(task)
        }
        return result
    }
}

const countingMethods = {
    "count"(task) {
        return 1
    },
    "points"(task) {
        return task.estimatePoints || 0
    },
    "hours"(task) {
        return task.estimateHours || 0
    }
}

function groupByStatus (statuses, items, countMetric) {
    let result = { max: 0, total: 0, distribution: { } }
    for (let item of items) {
        result.distribution[item.status] = (result.distribution[item.status]??0) +
countingMethods[countMetric]?.call(null, item)
    }
    for (let key in result.distribution) {
        result.max = Math.max(result.max, result.distribution[key])
        result.total += result.distribution[key]
    }
    let distributionArray = [ ]
    for (let [name, {index}] of Object.entries(statuses)) {
        distributionArray[index] = [name, result.distribution[name]??0]
    }
    return { max: result.max, total: result.total, distribution: distributionArray }
}

const colors = [
    "#64c989",
    "#6684c9",
    "#d95498",
    "#df8856",
    "#f8c044",
]

export default {
    components: {
        Selectbox
    },
    props: {
```

```
        parent: Object,
        query: Object
    },
    data() {
        return {
            colors: m(colors),
            countMetricOptions: m(countMetricOptions),
            groupByOptions: m(groupByOptions),
            countMetric: "count",
            groupBy: "taskset",
        }
    },
    computed: {
        taskDistribution() {
            let { project } = this.$storage
            let tasks = Object.values(project.data.taskObjects).filter(t=> t.type=="Task")
            let groupedTasks = groupingMethods[this.groupBy]?.call({ project, app: this.$app }, tasks)
            if (!groupedTasks) return null
            groupedTasks = Object.values(groupedTasks)
                .map(({ head, items })=> ({ head, items: groupByStatus(project.data.taskStatuses, items,
this.countMetric) }))
            return groupedTasks
        }
    }
}

</script>


src/Selectbox.js:
import { h } from "vue"

export default {
    props: {
        value: undefined,
        items: Array
    },
    data() {
        return {
            expanded: false
        }
    },
    emits: [
        "change"
    ],
    methods: {
        onSelectItem (item) {
            this.expanded = false
            this.$emit("change", item.value)
        },
        onToggleExpand () {
            this.expanded = !this.expanded
        }
    },
    render() {
        return h("div", { class: ["selectbox-wrapper"] }, [
            h("div", { class: ["flex-stripe", "selectbox-header"], onClick: ()=> this.onToggleExpand() }, [
```

```
            h("span", { class: ["flex-grow", "one-line"] }, this.items?.find(i=> i.value==this.value)?.text),
            h("span", { class: ["clickable", "pad-0-05"] }, "+")
          ]),
          h("div", { class: ["selectbox-body"], display: this.expanded }, [
            this.items?.map(item=> h("p", { class: ["pad-025", "clickable"], onClick: ()=>
this.onSelectItem(item) }, item.text))
          ])
      ])
    }
}
```

src/style.scss:

```
/* dashboard-plugin@0.0.1 */

div[plugin="dashboard-plugin"] {

  .dashboard-row {
    display: flex;
    flex-direction: row;
    align-items: center;
    column-gap: 0.5rem;

    + .dashboard-row {
      margin-top: 0.5rem;
    }
  }

}
```

package.json:

```
{
    "name": "dashboard-plugin",
    "version": "0.0.1",
    "private": true,
    "scripts": {
      "build": "vite build --mode prod",
      "build-debug": "vite build --mode debug --minify false"
    },
    "dependencies": {
      "vue": "^3.2.47"
    },
    "devDependencies": {
      "@vitejs/plugin-vue": "^3.0.3",
      "sass": "^1.82.0",
      "vite": "^3.0.9"
    }
}
```

vite.config.js:

```
// dashboard-plugin@0.0.1
// created by create-proj-plugin

import { resolve } from 'path'
import { defineConfig } from 'vite'
```

```
import vuePlugin from "@vitejs/plugin-vue"

export default defineConfig({
   plugins: [
      vuePlugin()
   ],
   build: {
      target: "es2020",
      lib: {
         entry: resolve(__dirname, 'src/main.js'),
         name: 'dashboard-plugin',
         fileName: 'plugin',
         formats: ["cjs"]
      },
      rollupOptions: {
         // externalize vue so it's not bundled but re-imported
         external: ['vue'],
         output: {
            format: "cjs",
            exports: "named",
            globals: { vue: 'Vue' },
         },
      },
   },
})
```