Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Програмний засіб щодо впровадження Kanban board до навчального процесу»

Виконав: студент 4 курсу
групи ПІ-18б(з)
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Демченко В. С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Хошаба О.М.
(прізвище та ініціали)
Рецензент: к.т.н., ст. вик. КН Озеранський В.С
(прізвище та ініціали)

Допущено до захисту
Зав. кафедр _____
«___» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 - Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Демченко Віталій Сергійович

1. Тема роботи - «Програмний засіб щодо впровадження Kanban board до навчального процесу»

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 р. № 66

2. Строк подання студентом роботи 13 червня 2022 р.

3. Вихідні дані до роботи: Середовище розробки – Visual Studio Code, Мови розробки – TypeScript, JavaScript, Node.js, Операційна система – Fedora Linux 36.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; розробка інтерфейсу та алгоритмів програмного додатка; розробка програмного додатку; тестування додатку, висновки; список використаних джерел, додатки

5. Перелік графічного матеріалу: демонстрація роботи аналогів, блок-схема алгоритмів програмного додатку, діаграма переходів між екранами додатку, блок-схема взаємодії модулів програмного додатку, тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Хошаба О.М., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задач	26.03.2022-10.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи системи	11.04.2022-26.04.2022	Вик.
3	Вибір середовища та мови розробки	27.04.2022-4.05.2022	Вик.
4	Розробка програмного продукту	5.05.2022-24.05.2022	Вик.
5	Тестування роботи системи	25.05.2022-27.05.2022	Вик.
6	Оформлення матеріалів до захисту БДР	27.05.2022-10.06.2022	Вик.

Студент _____ Демченко В.С.

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ Хошаба О.М.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 83 сторінок формату А4, на яких є 47 рисунків, 2 таблиці, список використаних джерел містить 48 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз навчального процесу та можливостей щодо його вдосконалення. Було проаналізовано процес отримання, виконання та здачі завдань студентами, та запропонована програмне рішення для удосконалення цього процесу. Запропоновано використовувати програмний додаток на основі Kanban Board в навчальному процесі для видачі завдань та відстежування стану виконання, що дозволило покращити комунікацію між викладачами та студентами, і підвищить ефективність роботи в цілому. Kanban Board було обрано, як інструмент який гарно себе зарекомендував у великих комерційних компаніях де використовується для ведення проектів з великою кількістю завдань та співробітників. Kanban Board є гарним засобом для розв'язання проблем менеджменту великої кількості завдань, з чим постійно стикаються студенти під час навчального процесу.

У термінології Kanban Board проект це певний предмет, і що б цей предмет був закритий необхідно виконати всі завдання, що були поставлені викладачем. Процес виконання завдань виглядає наступним чином студент отримує завдання воно потрапляє на Kanban Board в колонку “До виконання”, студент виконує і переносить картку з завданням далі по етапах до моменту поки завдання не стане “Виконано”.

Для впровадження Kanban Board в навчальний процес було розроблено програмний додаток з використанням веб технологій. Даний програмний додаток складається з двох частин сервера та клієнту. Клієнт та сервер був реалізовані за допомогою мови програмування TypeScript. В якості середовища розробки було обрано Visual Studio Code.

Отримані напрацювання в даній дипломній роботі можуть бути впроваджені в реальний навчальний процес.

ABSTRACT

The bachelor's thesis consists of 83 A4 pages, which contain 47 figures, 2 tables, a list of sources used contains 48 items.

In the bachelor's thesis a detailed analysis of the educational process and opportunities for its improvement. The process of receiving, completing and submitting assignments by students was analyzed, and a software solution was proposed to improve this process. It is proposed to use a software application based on Kanban Board in the educational process for issuing tasks and tracking the status of implementation, which has improved communication between teachers and students, and increase overall efficiency. Kanban Board was chosen as a tool that has proven itself in large commercial companies where it is used to conduct projects with a large number of tasks and employees. Kanban Board is a good tool for solving the problems of managing a large number of tasks that students are constantly faced with during the learning process.

In Kanban Board terminology, a project is a specific subject, and whatever the subject is closed, all the tasks set by the teacher must be completed. The process of completing the tasks is as follows: the student receives the task, it falls on the Kanban Board in the column "To be completed", the student completes and transfers the card with the task further in stages until the task becomes "Completed".

A software application using web technologies was developed to implement the Kanban Board in the educational process. This application consists of two parts, server and client. The client and server were implemented using the TypeScript programming language. Visual Studio Code was chosen as the development environment.

The achievements in this thesis can be implemented in a real educational process.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасних реаліях, пандемія коронавірус, війна багато хто з студентів та викладачів не мають змоги фізично бути присутніми в стінах навчального закладу. В таких важких умовах навчальний процес все одно триває. Прискореними темпами були впровадженні сучасні засоби для перенесення навчального процесу в онлайн. За короткий термін відбулися кардинальні зміни в процесах, які були незмінними протягом десятиліть.

Процес трансформацій все, ще триває. Оскільки працююча система не означає оптимальна система, необхідно аналізувати процес цифровізації навчання, та знаходити процеси, які, ще можна покращити.

Навчання для студента можна поділити на два етапи «начитка» прослуховування лекцій, та «виконання завдань» закріплення матеріалу з лекцій шляхом виконання практичних завдань, лабораторних, курсових, контрольних тощо.

Перенесення начитки в онлайн відбулося за рахунок створення кабінету студента де кожен студент може отримати інформацію про розклад занять, завантажити необхідні матеріали які публікує викладач по певному предмету. Взаємодія між студентами та викладачами, яка потребувала фізичної присутності в навчальному кабінеті, проведення лекцій, було перенесена в програмні засоби онлайн конференцій. Саме це дало змогу в короткий термін організувати навчання онлайн з мінімальними змінами в сам процес. Процес залишився таким самим, змінився, лише засіб комунікації, студенти та викладачі зустрічаються не в певному кабінеті, а в певній онлайн кімнаті в Google Meet.

Етап практики перенесений в онлайн за знав мінімальної кількістю змін. З нововведень, що відрізняють офлайн виконання завдань студентами, від онлайн формату тепер в студентів новий вид завдань онлайн тести, які повинні проходити студенти, і які перевіряються автоматично. Всі інші завдання, надсилаються викладачу у вигляді файлу, викладач перевіряє і в разі певних недоліків надсилає студенту повідомлення з зауваженнями та проханням доопрацювати завдання.

З надсиланням та перевіркою завдань, все зрозуміло так працювало завжди упродовж багатьох років, те що замість паперового варіанту студент надсилає файл нічого не змінює.

Проте в процесі виконання практичних завдань студент стикається з проблемами менеджменту. Завдань багато і складно відстежувати поточний стан кожного. Студент не бачить всього списку завдань які йому необхідно виконати. Дана проблема приводить до неприємностей, коли студент не виконав завдання тільки через те, що десь збув записати, або пропустив листа від викладача про відправлення завдання на доопрацювання.

Подібні проблеми менеджменту великої кількості завдань існують не тільки в навчальному процесі. На великих виробництвах де відбуваються складні процеси та працюють тисячі співробітників відбувається структурно схожі процеси. Співробітнику видають завдання, він його виконує, проводить через певну кількість етапів, та відправляє на перевірку якості виконаної роботи. Одним з інструментів, які допомагають співробітникам знати, що роботи, а керівникам розуміти стадію виконання кожного з завдань є інструмент Kanban Board.

Kanban Board – це інструмент для візуалізації робочих процесів, призначений для того, щоб допомогти більш зручно управляти робочим процесом та підвищити їх ефективність [2]. Даний інструмент представляє собою дошку, на якій розмічені колонки, які представляють етапи виконання завдання. Та картки які візуалізують окремі задачі і розміщуються по колонкам у відповідності до поточного етапу виконання.

Оскільки проблеми менеджменту великої кількості завдань під час навчального процесу є актуальними. Є доцільним запозичення досвіду великих підприємств, та застосування інструменту Kanban Board в навчальному процесу.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання. Метою роботи є підвищення ефективності навчального процесу. За рахунок впровадження програмного додатку на основі інструменту

Kanban Board.

Основними завданнями розробляємого програмного додатка є вирішення трьох основних задач пов’язаних з процесом отримання, та виконання завдань під час навчального процесу.

Вирішення задачі точності отримання завдань - гарантоване отримання коректного завдання студентом. Навіть, у випадку, коли студента не було підчас видачі завдання.

Вирішення задачі менеджменту завдань - надання зручного засобу для візуалізації та роботи з великою кількістю завдань, та відстеження стану виконання кожного завдання.

Вирішення задачі комунікація між викладачем та студентом по певному завданню - виключення необхідності уточнювати, про яке завдання йде мова під час звернення студента з уточнюючими питаннями по певному завданню.

Об’єкт дослідження – процес видачі та виконання завдань студентами.

Предмет дослідження – методи та програмні засоби, що дозволяють покращити навчальний процес.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

- методи розробки графічного інтерфейсу;
- методи менеджменту великої кількості завдань;
- методи розробки клієнт серверних систем;
- методи для серіалізації текстових даних;
- методи авторизації користувача;

Наукова новизна отриманих результатів. Впровадження інструменту Kanban Board, який використовується на великих виробництвах, в навчальний процес. Впровадження отриманих результатів візуалізує поточний стан виконання завдань та вирішить задачу менеджменту цих завдань.

Практична цінність отриманих результатів. Практичне значення полягає у розробці та тестуванні програмного додатка, що впроваджує Kanban Board в навчальний процес та тим самим оптимізує роботу з завданнями для студентів та

викладачів.

Відповідно до поставленої мети потрібно виконати наступні задачі:

- виконати аналіз предметної області з оптимізації навчального процесу

- виконати порівняльний аналіз аналогів програмних рішень які реалізують Kanban Board.

- визначити основні завдання які необхідно вирішити для розробки програмного додатка для впровадження Kanban Board в навчальний процес.

- виконати проектування програмного додатка для впровадження Kanban Board в навчальний процес.

- розробити алгоритм та програмний код програмного додатка для впровадження Kanban Board в навчальний процес з застосування сучасних засобів розробки та мови програмування TypeScript;

- виконати тестування програмного додатка.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. Автору належать такі результати: архітектура додатку; розроблені алгоритми; клієнтська частини додатку; серверна частина додатку.

Публікації. Результати аналізу, щодо доцільності впровадження Kanban Board в навчальний процес. Були опубліковані в якості тез на науковій конференції - LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2022).

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз стану проблеми

Освіта є важливою для кожної людини. Тому, що роль освіти в усіх сферах життя людини та суспільства в цілому величезна, особливо в сучасному світі.

У найпростішому визначенні освіта – це процес навчання або набуття знань, навичок, цінностей, моралі, переконань і звичок. Весь процес отримання та надання освіти включає кілька етапів. Вона включає навчання, обговорення, дослідження та подібні інші інтерактивні заходи [3].

Освіта визначається як організований, специфічний і чітко спланований процес, спрямований на виявлення, зміну, або подальший розвиток ставлення, обізнаності та здібностей як передавача, так і одержувача знань. Освіта вважається успішною, коли наведені вище пропозиції, зміни та напрацювання йдуть у позитивному напрямку.

Освіта є важливою, оскільки сприяє формуванню та вдосконаленню особистості учнів. Таке формування особистості допомагає учням мати грані можливості на майбутнє, вміти засвоювати та розвиватися, стати особистістю, яка має достатньо чеснот і таланту, щоб задовольнити потреби існування та розвитку суспільства.

З моменту свого заснування освіта стала основою для кожної країни. Спираючись на освіту, країна може краще адаптуватися до світу, створюючи нові ресурси для швидкого та сталого розвитку [4].

Як, було описано вище, навчання є надзвичайно важливим як, для окремої людини її росту та розвитку, так і для суспільства в цілому оскільки розумне, навчене, висококваліфіковане суспільство здатне створювати нові технології та підвищувати ефективність праці.

В нашій країні кожна людина повинна отримати мінімум середню освіту, та має можливість отримати вищу освіту в університеті. Це означає, що кожен точно стикався з навчальним процесом і знає як, він виглядає. Є начитка матеріалу, та практика - виконання завдань для закріплення, це можуть бути домашнє завдання,

самостійні та контрольні роботи, лабораторні, практичні заняття. Після того як, учень опанував матеріал, та продемонстрував його володіння при виконанні певної кількості завдань предмет може бути вважатися опанованим.

В даній дипломній роботі, було проаналізовано проблеми комунікації та менеджменту, які виникають під час отримання та здачі завдань. Було розроблено програмне рішення, яке при впровадженні в навчальний процес вирішує три основі проблеми комунікації та менеджменту під час виконання практичних завдань студентами.

Перша проблема «точність отримання завдань» - при отриманні завдання учень в більшості випадках записує завдання в записник для того, щоб виконати його в подальшому. Іноді виникають випадки коли студент записав, щось некоректно, або був відсутній на лекції і йому передали завдання ті хто був присутній, тут проявляється проблема зламаного телефону — завдання було сказано словесно викладачем, занотовано учнями, та передано тим учням яких не було на парі. Оскільки текст завдання є інформацією, і відсутній механізм гарантування правильної передачі даної інформації на кожній ланці існує можливість не точної передачі завдання.

Проблему “точності отримання завдання” вирішують надсилаючи завдання на електронну пошту. Зручно для викладача і правильне завдання точно отримають всі студенти, навіть ті яких не було на парі.

Друга проблема «менеджмент завдань» - проблема менеджменту завдань, для студента. В студентів є багато предметів та на кожному з цих предметів видається велика кількість завдань, що б не заплутатись в цьому здебільшого всі групують завдання по предметах. На цьому рівні менеджмент не викликає проблем, але коли ми згадаємо, що завдання мають декілька етапів і це теж треба відстежувати, для розуміння обсягу роботи який виконаний, та який необхідно буде виконати.

До прикладу виконання творчого завдання з певною темою можна розбити на такі етапи «До виконання», «Отримано тему», «В роботі», «Відправлено на перевірку», «Відправлено на доопрацювання», «Здано». Не рідкими є випадки

коли студент, не доопрацював роботу коли з якихось причин пропустив листа від викладача про доопрацювання, і помітив собі завдання, як виконане, що є порушенням процесу виконання завдання, оскільки воно не може бути виконано, до повної перевірки викладачем.

Третя проблема «комунікація між викладачем та студентом по певному завданню» - питання в студентів виникають здебільшого не в момент видачі завдання, а під час виконання тому коли студент звертається до викладача йому треба довго пояснювати, що за група, який предмет про яке конкретно завдання йде мова. Ця проблема, не така критична як дві попередніх, але її автоматизація значно спростить життя викладачам. Цю проблему здебільшого вирішують надсилаючи разом, з листом повний текст самого завдання, проблема цього рішення в відсутність прив’язки листування з викладачем до завдання, іноді коментар викладача по тому чи іншому завданню важко знайти в електронній поштовій скриньці.

Для розв'язання цих проблем пропонується запозичити інструмент, який широко використовується в великих компаніях де є значна кількість проект та завдань. В нашому випадку замість проектів — предмети, а процес виробництва продукції, який складається з великої кількості завдань та етапів, такий самий процес, як і процес навчання.

Інструмент, який пропонується запозичити це – Kanban Board, але спочатку необхідно розглянути методологію з якої запозичений цей інструмент, щоб зрозуміти, з чого все починалося.

Отже, Kanban — це система контролю запасів, яка використовується у виробництві «точно вчасно» (JIT). Він був розроблений Тайічі Оно, промисловим інженером Toyota, і отримав свою назву від кольорових карток, які відстежують виробництво та замовляють нові постачання деталей або матеріалів, коли вони закінчуються.

Kanban — це японське слово для знака, тому система канбан просто означає використання візуальних підказок для спонукання до дій, необхідних для

забезпечення течії процесу. Однією з головних цілей Kanban є обмеження накопичення надлишкових запасів у будь-якій точці виробничої лінії [5].

Kanabn Board - є одним з інструментів, які можна використовувати для впровадження Kanban для управління роботою на особистому, або організаційному рівні. Kanban Board візуально зображують роботу на різних етапах процесу за допомогою карток для представлення робочих елементів і стовпців для представлення кожного етапу процесу [7] (рис. 1.1).

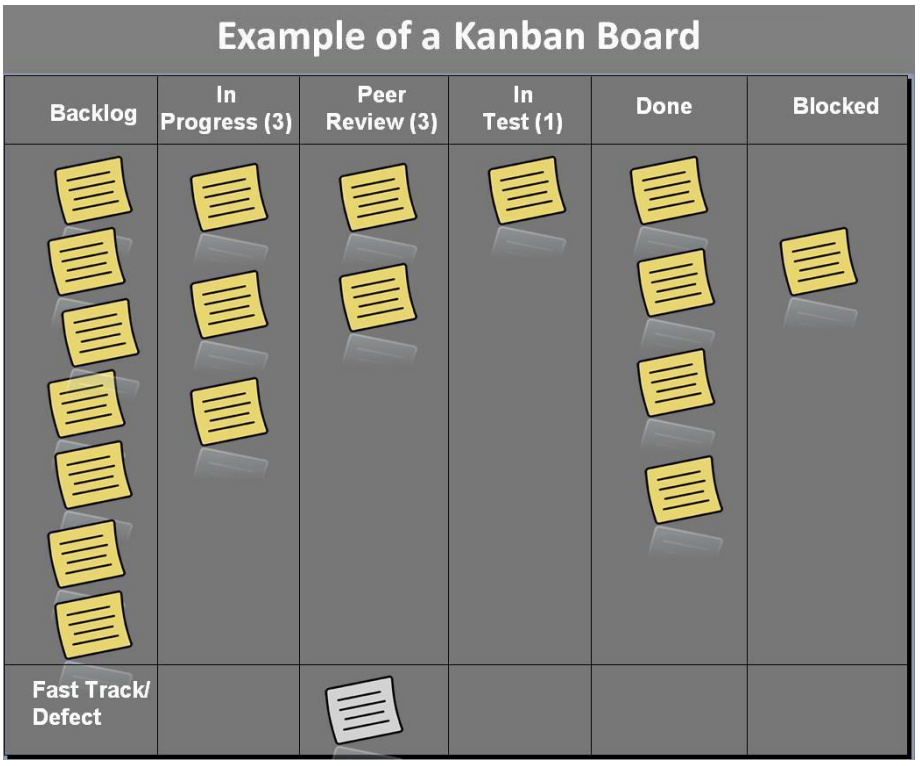


Рисунок 1.1 — Приклад Kanban Board

Оскільки, Kanban Board – це простий спосіб візуалізації будь-яких завдань, які проходять декілька стадій до повного виконання такий підхід можна гарно застосувати для візуалізації академічної роботи та ефективного управління.

Kanban Board добре підходить для візуалізації процесів, які можна розділити на етапи [6]. Як процес можна розглядати виконання завдань студентами, оскільки завдання проходить через певні етапи до того, як воно буде виконано. Студент отримує завдання, виконує його, відправляє на перевірку, завдання може відправитися на доопрацювання, або одразу в фінальну стадію виконано.

Кожна з цих стадій проходить в будь-якому випадку, хоча, на даний момент вони не формалізовані, впровадження інструменту Kanban Board допоможе формалізувати етапи та розв'язувати проблему управління великою кількістю завдань.

1.2 Порівняльний аналіз аналогів

Порівнюючи, аналоги ми розглянемо декілька реалізацій Kanban Board, та проаналізуємо можливість їх використання для наших цілей.

Першим розглянемо один з найпопулярніших засіб для управління проектами за допомогою Kanban Board – Trello.

Trello - багато платформна система управління проектами, розроблена Trello Enterprise, дочірньою компанією Atlassian [4]. Створена у 2011 році компанією Fog Creek Software (тепер Glitch) для створення окремої компанії в Нью-Йорку у 2014 році й продана Atlassian в січні 2017 року [8] (рис. 1.2).

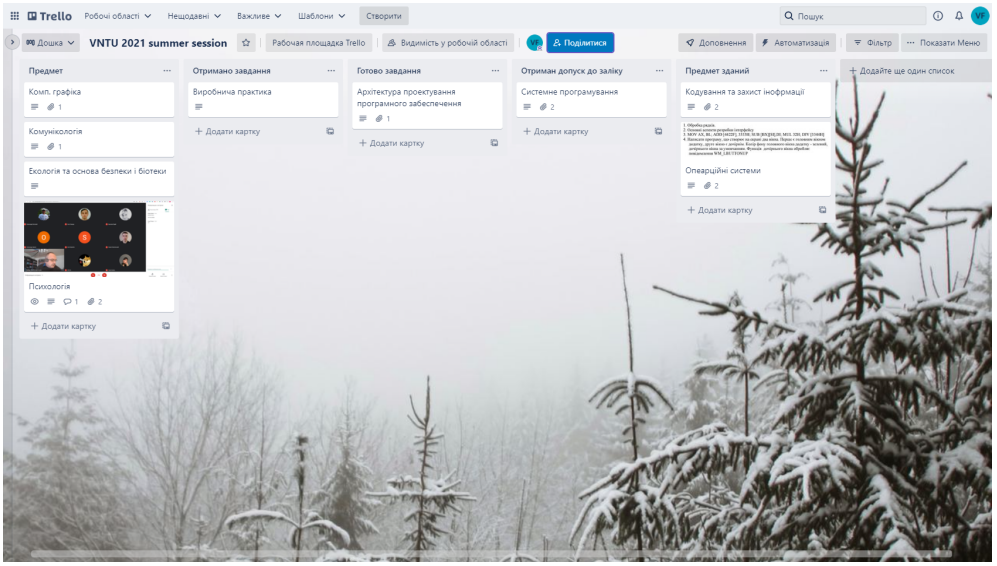


Рисунок 1.2 — Інтерфейс програмного рішення Trello

Вона використовує парадигму керування проектами, відому як Kanban. Проекти зображуються дошками, що містять стовпчики. Стовпчики містять картки, якими зображуються задачі. Картки повинні переходити з попереднього списку до наступного (за допомогою перетягування), таким чином

зображаючи рух якоїсь функції від ідеї, аж до тестування. Картці може бути присвоєно відповідальних за неї користувачів. Користувачі та дошки можуть об'єднуватись в команди [8].

До переваг Trello можна віднести:

- зручний інтерфейс
- веб рішення доступне на всіх платформах
- можливість працювати декількома користувачами, з одною дошкою
- інтеграція з Google Drive, що дозволяє прикріпити файли до завдань

До недоліків Trello можна віднести:

- закритий код програми
- обмеженість безкоштовної версії
- можливість безкоштовно використовувати, лише у персональних цілях

Друге програмне рішення, це одне з найпопулярніших рішень для ведення проектів у великих компаніях з розробки програмного забезпечення Jira, це рішення для великих виробництв. Kanban Board в Jira є лише одним з багатьох інструментів менеджменту (рис. 1.3).

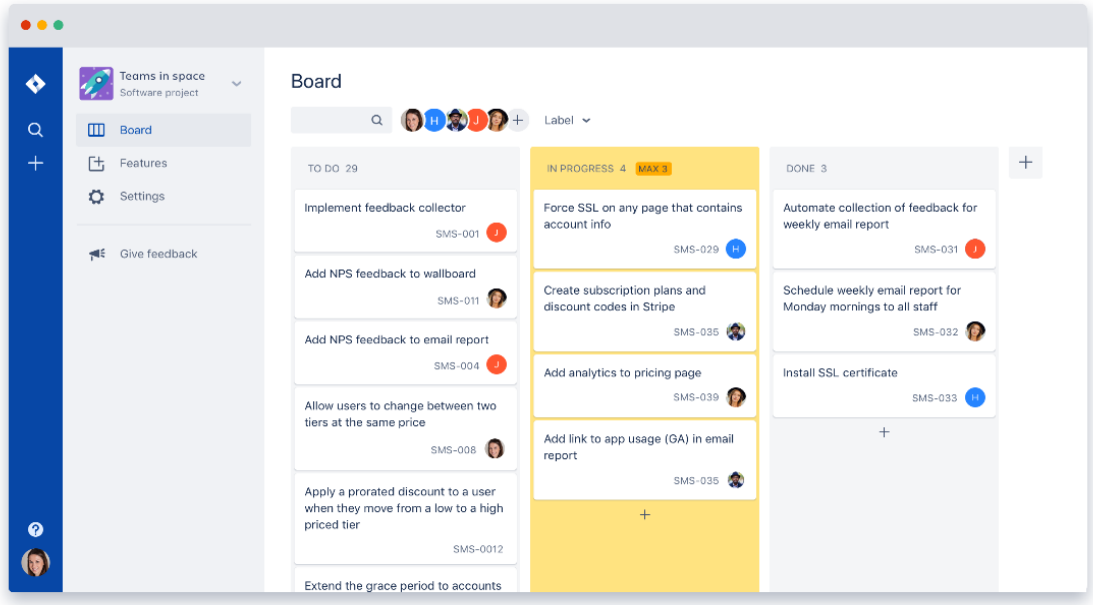


Рисунок 1.3 — Інтерфейс програмного рішення Jira

Atlassian JIRA — система відстеження помилок, призначена для організації спілкування з користувачами, і для управління проектами. Розроблена компанією Atlassian у 2002 році. Доступна у двох версіях: «хмарній» і серверній. Зараз JIRA включає в себе три проекти: JIRA Software (для розробників), JIRA Service Desk (підтримка проекту), JIRA Core (управління проектами), кожен з яких можна купити окремо. Назва системи (JIRA) отримано шляхом модифікації японської назви Годзіла («Gojira»), що в свою чергу є алюзією на назву конкуруючого продукту — Bugzilla. JIRA створювалася як заміна Bugzilla й багато в чому повторює архітектуру останньої. На даний момент (2018 рік) JIRA є однією з найпопулярніших систем управління проектами [9].

Як описував вище, Jira це великий продукт, для великих компаній. Для цілей зазначених в даній роботі можливо використання Jira Core, як рішення для ведення проектів.

До переваг Jira можна віднести:

- зрозумілий інтерфейс
- гнучкість налаштування
- прив’язка до конкретного користувача, та його прав на певну задачу
- інтеграція з сервісом написання документації Confluence
- інтеграція з хостингом репозиторіїв коду Bitbucket

До недоліків Jira можна віднести:

- повільну роботу, велика система іноді треба час на опрацювання запитів
- закритий код
- виключно платне використання продукту
- погана інтеграція з сервісами не від Atlassian

В якості третього продукту для порівняння я обрав також одну з популярних реалізацій Kanban Board продукт Hygger, він націленого саме на менеджмент проектів (рис. 1.4).

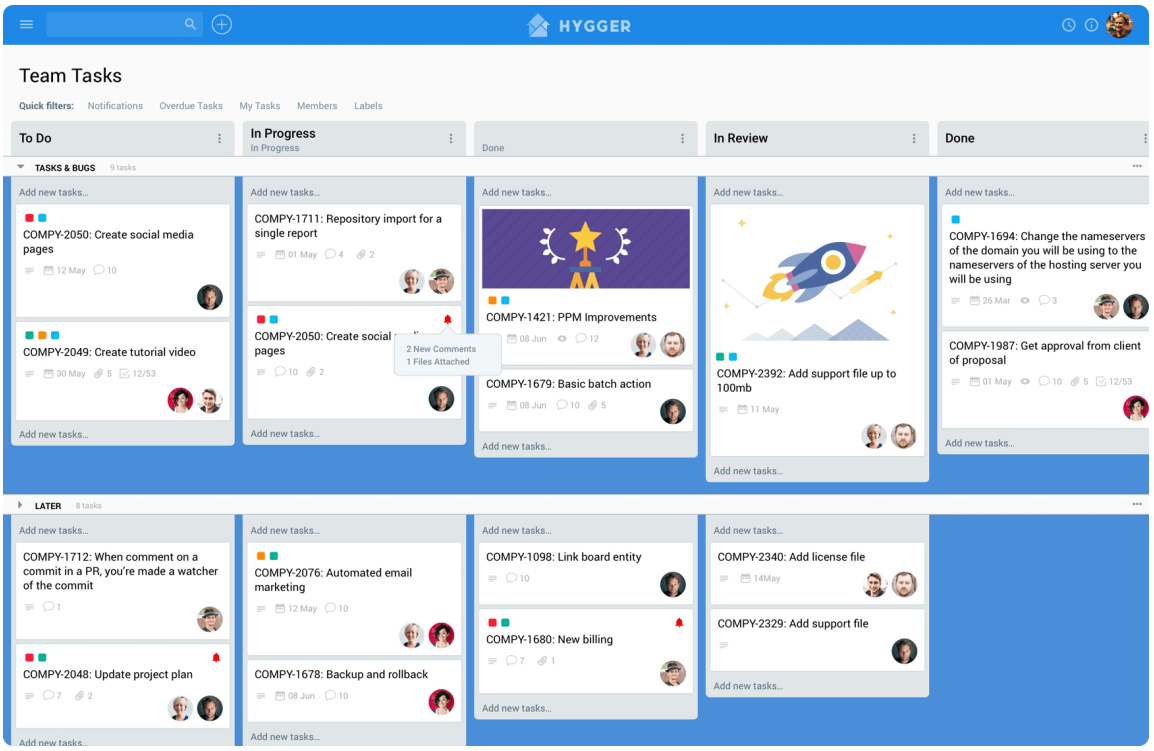


Рисунок 1.4 — Інтерфейс програмного рішення Hygger

Hygger.io — це програма для управління проектами з вбудованою системою визначення пріоритетів, створена Hygger LLC у 2017 році. Вона підтримується на багатьох платформах, включаючи Microsoft Windows, Mac OS, Android, Linux/Unix та iOS. Система підтримує дошки Kanban для гнучкого управління проектами та включає передові інструменти для організації спринтів [11].

До переваг Hygger можна віднести:

- зручний інтерфейс
- веб рішення доступне на всіх платформах
- можливість працювати декількома користувачами, з одною дошкою

До недоліків Hygger можна віднести:

- закритий код програми
- відсутність безкоштовної версії

Zoho Projects — це хмарна програма Kanban, яку використовують Dell, Стенфордський університет і Siemens (рис. 1.5).

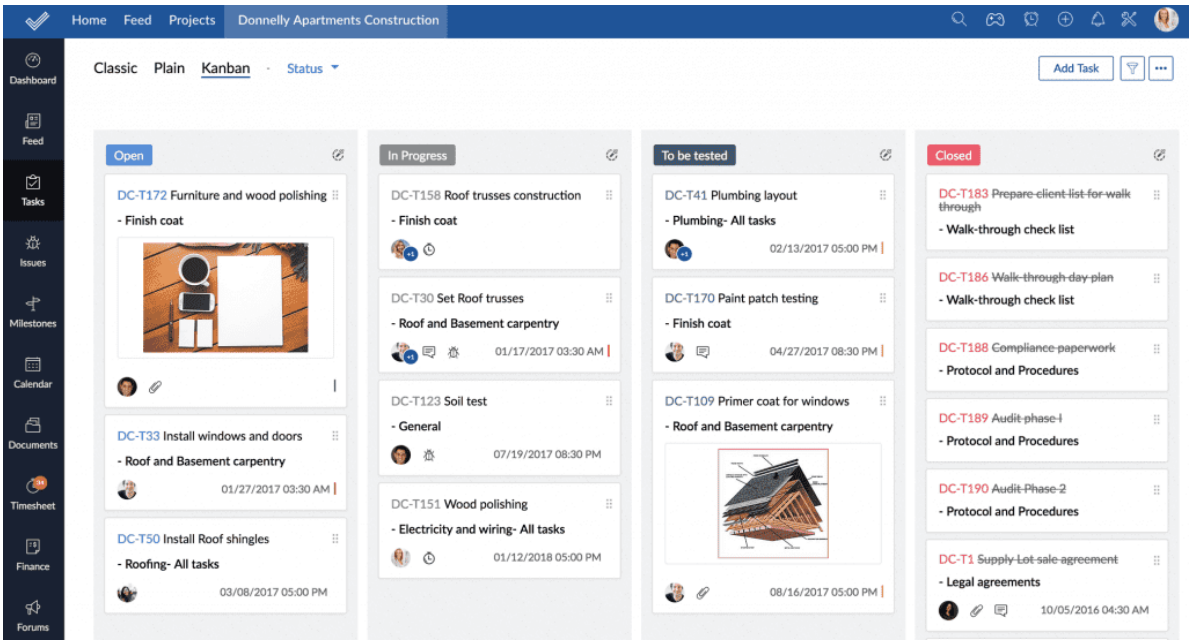


Рисунок 1.5 — Інтерфейс програмного рішення Zoho Projects

До стандартних функцій Zoho Projects включають спільне використання та зберігання документів, призначення завдань і оновлення статусу, а також ранжирування завдань відповідно до поточного рівня серйозності.

Користувачі також можуть визначати відносини між завданнями, реєструвати проблеми на дошках і картках безпосередньо із зовнішніх веб-сайтів, а також оцінювати поточне робоче навантаження окремого співробітника, щоб запобігти вигоранню [12].

Але те, що насправді відрізняє Zoho Projects від конкурентів, — це функції управління соціальними робочими процесами.

Окрім коментування та позначення інших користувачів, у кожного користувача є власна сторінка профілю, на якій відображається потік його поточної та майбутньої діяльності. Користувачі можуть завантажувати файли у свої стрічки, встановлювати особисті статуси та прикріплювати важливі питання у верхній частині своїх каналів.

Існують, ще багато програмних рішень, які реалізують Kanban Board, але з де більшого вони майже однакові. Відрізняються ціною, привабливість інтерфейсу, та інтеграцією з сторонніми сервісами.

Було проведено порівняння аналогів, та результати зведені (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Аналог	Доступність на всіх платформах	Можливість використання в команді	Висока швидкодія програмного додатку	Інтеграція з сторонніми сервісами	Відкритість коду продукту	Можливість безкоштовного використання в навчальних закладах
Trello	так	ні	так	GDrive/ Dropbox	закритий код	можливе безкоштовне персональне використання
Jira	так	так	залежить налаштувань	Confluence/ Bitbucket	закритий код	тільки платна ліцензія
Hygger	так	так	так	відсутня	закритий код	тільки платна ліцензія
Zoho Projects	так	так	залежить налаштувань	відсутня	закритий код	можливе безкоштовне персональне використання

Про аналізувавши аналоги, можемо стверджувати, що рішень на даний момент які, б заодівльнили нашим потребам немає. Було виділено окремим критерієм відкритість коду, та можливість безкоштовного використання. Для цілей даної роботи відкритий код необхідний для подальшої інтеграції з іншими онлайн сервісами навчального закладу, а безкоштовне використання важливе оскільки за цей продукт доведеться платити студентам і не допустимо впроваджувати рішення, яке може бути комусь не доступне з фінансових причин.

Виходячи з цього, вважаю доцільним розробку власного рішення яке може бути легко інтегровано з сервісами навчального закладу, буде мати відкритий код, щоб студенти могли пропонувати вдосконалення продукту та буде абсолютно безкоштовне для використання.

1.3 Постановка задачі

Для розробки та впровадження Kanban Board в навчальний процес необхідно програмне рішення, яке буде доступне на всіх платформах та буде реалізувати наступні задачі:

- створення дошки;
- створення облікового запису;
- створення завдань у вигляді Kanban наліпки на дошці;
- можливість переміщати завдання по етапам;
- можливість редагування етапів на дошці;
- видача завдань певній групі студентів;
- редагування завдань;
- коментування завдань;
- можливість додавати в закріплене до завдання посилань на файли та інші матеріали необхідні для виконання даного завдання;
- можливість виставляти крайній строк здачі завдання.

Упродовж розробки необхідно:

- виконати аналіз предметної області з впровадження Kanban Board в навчальний процес;
- виконати аналіз аналогів які реалізують Kanban Board;
- виконати проектування програмного рішення, що дозволить впровадити Kanban Board в навчальний процес;
- розробити необхідні алгоритми та програмний код Kanban Board на мові програмування TypeScript;
- виконати тестування отриманого програмного засобу.

1.4 Висновки

В першому розділі ми розглянули предметну область навчальний процес та можливості його вдосконалення шляхом впровадження Kanban Board. Було акцентовано на трьох проблемах які прагне вирішити програмний засіб, а саме проблеми «точності отримання завдань», «менеджменту завдань» та «комунікації між викладачем та студентом по певному завданню». Вирішення цих проблем за допомогою одного інструменту значно підвищить зручність роботи з великою кількістю завдань.

Були розглянуті аналоги та проведена порівняльна характеристика, виходячи з якої зроблено висновок про доцільність створення власного рішення з відкритим кодом, вільною ліцензією для використанням та завдяки відкритого коду легкою можливістю інтегрування з будь-якими необхідними сервісами навчального закладу.

Також, в цьому розділі було визначена постановка задачі, яка полягає у розробці програмного додатку Kanban Board, та необхідних функції для впровадження в навчальний процес.

2 ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ТА АЛГОРИТМІВ ДОДАТКА

2.1 Основні вимоги до програмного додатку

Одним з важливих частин навчального процесу є видача та здача студентами завдань. Зазвичай даний процес відбувається в такому вигляді, викладач словесно, або листом на пошту видає завдання по певному предмету, певній групі студентів, які повинні виконати це завдання до зазначеної дати. Після виконання завдання, студенти надсилають на перевірку завдання, викладач перевіряє, та в залежності від якості виконаної роботи надсилає на доопрацювання, або приймає роботу і завдання вважається виконаним [48].

Завдання проходить певні стани «До виконання», «В процесі виконання», «Відправлено на перевірку», «Виконано». Повний процес зміни станів та можливих переходів зображено на рис. 2.1.

Рисунок 2.1 — UML діаграма станів завдання

Вище описано найпростіше завдання, яке однакове для всіх студентів в групі. Проте не обов’язково етапи виконання завдань завжди будуть саме такими, наприклад при виконанні курсової роботи з'являються додаткові етапи. В курсових роботах може бути додатковий етап «Узгодження теми роботи» (рис. 2.2).

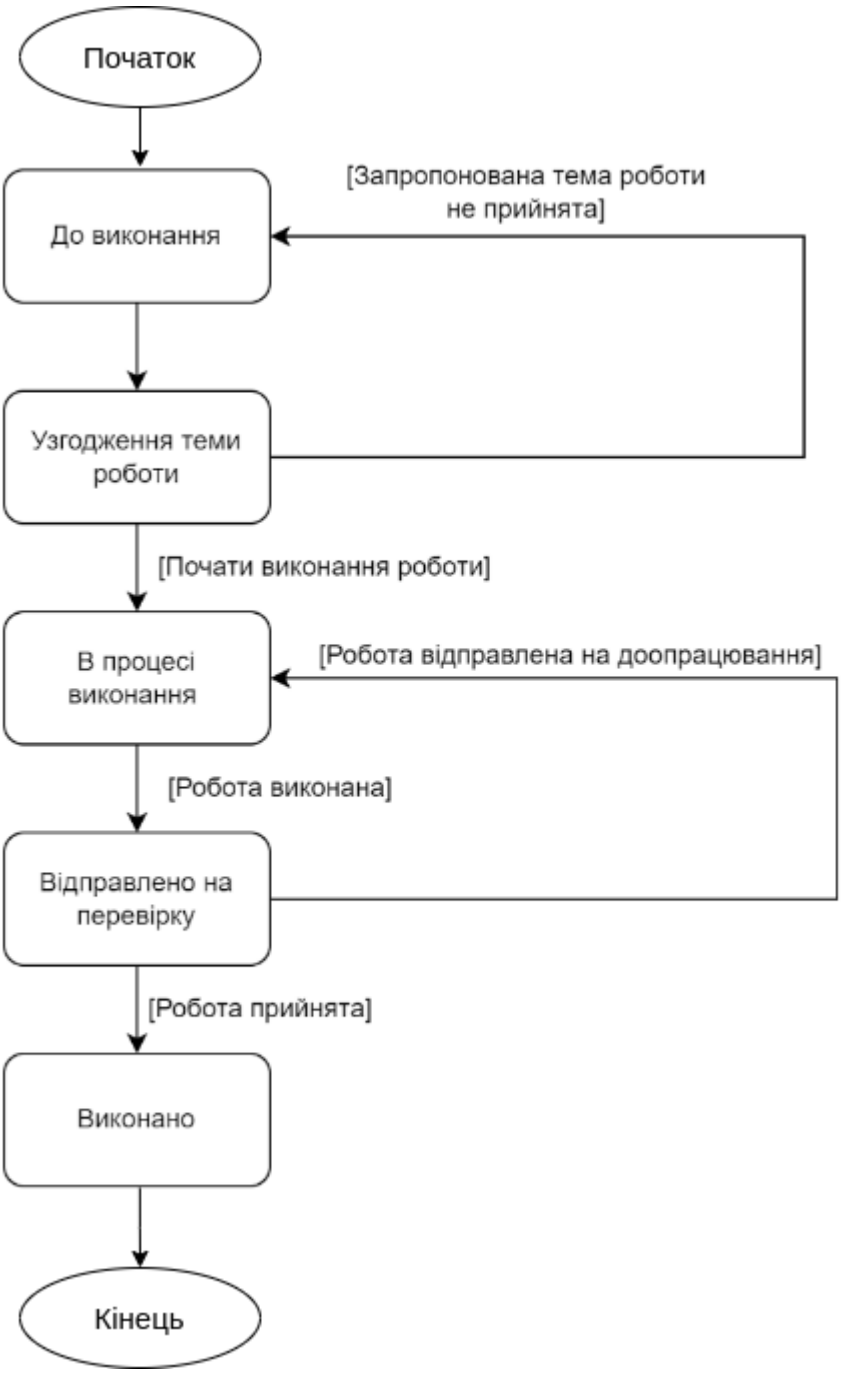


Рисунок 2.2 — UML діаграма станів курсової роботи

На даний момент, кожна зі стадій виконання завдання досить умовна, і

відображається, або в записнику у студента, або листом в електронній пошті, що при рості кількості завдань створює проблеми менеджменту.

Формуючи вимоги до майбутньої системи виходячи з прикладів вище, програмний додаток має надавати візуальне структуроване представлення завдань, та етапів виконання на яких знаходиться кожне з завдань. Етапи в кожному конкретному випадку можуть бути різні в залежності від предмету та типу завдань. В даній роботі для реалізації вимог, для роботи з задачами буде використаний інструмент Kanban Board.

2.2 Розробка інтерфейсу програмного додатка

Оскільки виконання завдання легко можна представити в виді етапів, одним з найкращих інструментів для візуалізації подібних поетапних процесів є Kanban Board. В цьому підрозділі розглянемо більш детально, як повинна виглядати дошка та які основні функції мають бути реалізовані.

Нагадаю, що Kanban Board - це візуальний інструмент, який дає огляд поточного стану роботи [7]. Він може мати, як фізичний вигляд (рис. 2.3) так і цифровий. В будь-якому випадку складається з двох основних елементів це стовпчики та картки.

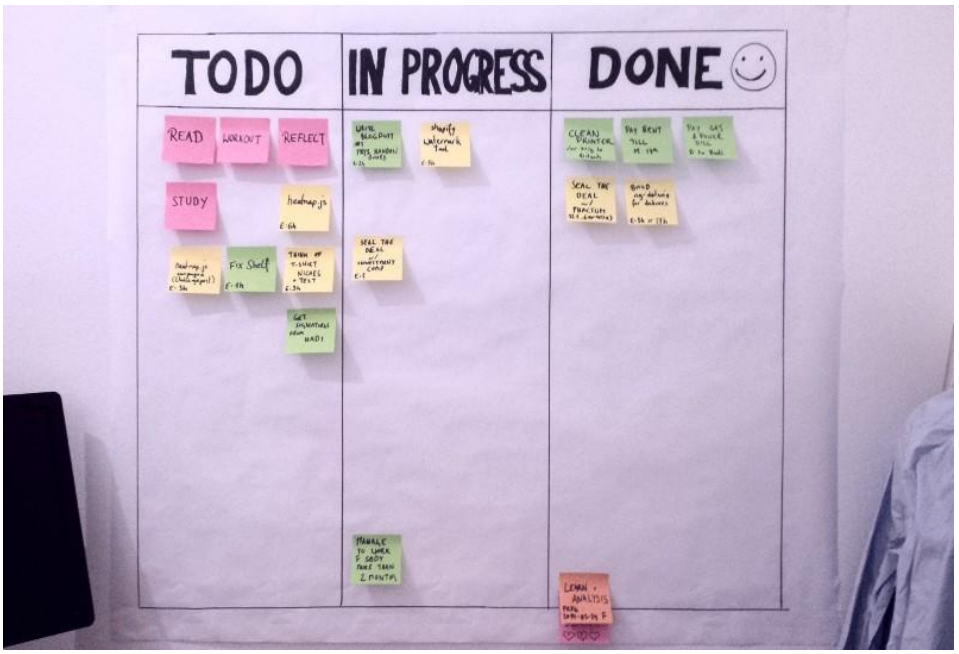


Рисунок 2.3 — Фізичний вигляд Kanban board

Стовпчики в Kanban Board - Кожен стовпець на дошці представляє інший етап вашого робочого процесу. Картки проходять робочий процес до повного завершення [8].

Картки в Kanban Board - це наочне уявлення завдань. Кожна картка містить інформацію про завдання та його статус, наприклад, термін виконання, дату до якої необхідно виконати, опис тощо [8].

В базовому виконанні Kanban Board (рис. 2.4) має вигляд дошки з картками розподіленими по етапах виконання, та можливістю пересувати завдання з одного стовпчика в інший за допомогою жесту «Тягни та відпускай» (англ. Drag and Drop).

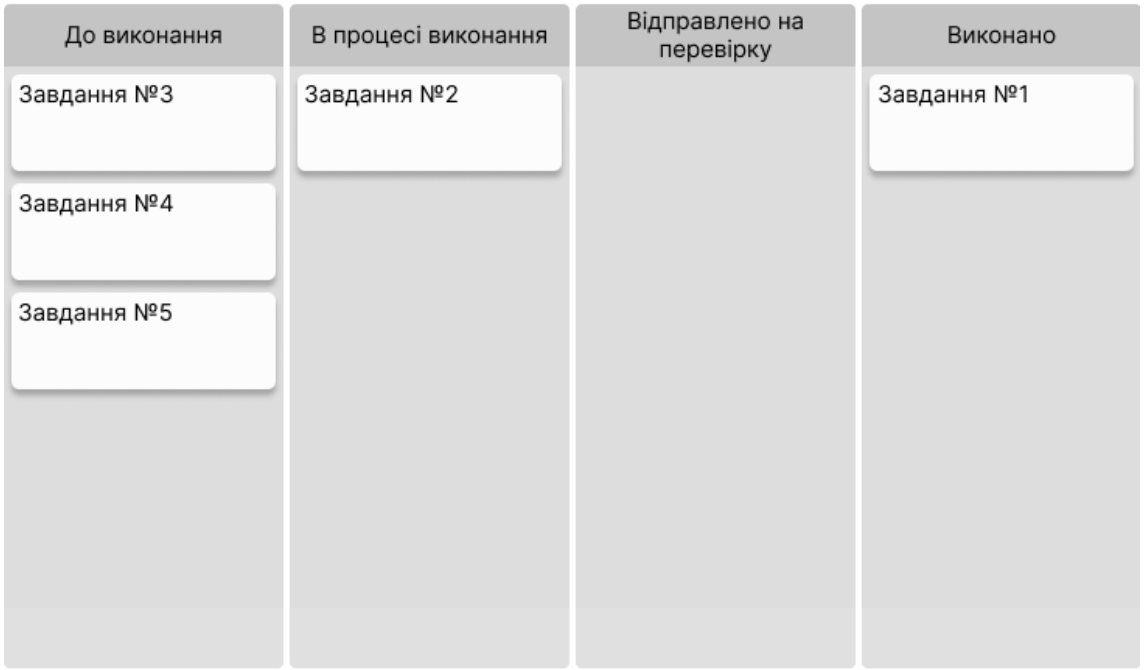


Рисунок 2.4 — Базовий макет інтерфейсу Kanban Board

Мета даної роботи впровадити Kanban Board в навчальний процес, тому необхідно формувати функціональні вимоги до кожного з елементів додатка, з урахуванням того, що даний програмний додаток буде використовуватися в навчальному процесі.

Необхідний функціонал створення Kanban Board. Вікно створення дошки повинно містити такі поля “Назва” - задання назви, та “Учасники” задання списку

учасників які будуть мати доступ до даної дошки. Також, автоматично додавати, до списку учасників користувача, що створив дошку, та не давати змогу користувачу видалити себе зі списку учасників. Даний унеможливить створення порожньої дошки, або дошки у якої відсутній користувач, що створив цю дошку. Виходячи з вимог було створено макет інтерфейсу вікна створення Kanban Board (рис. 2.5).

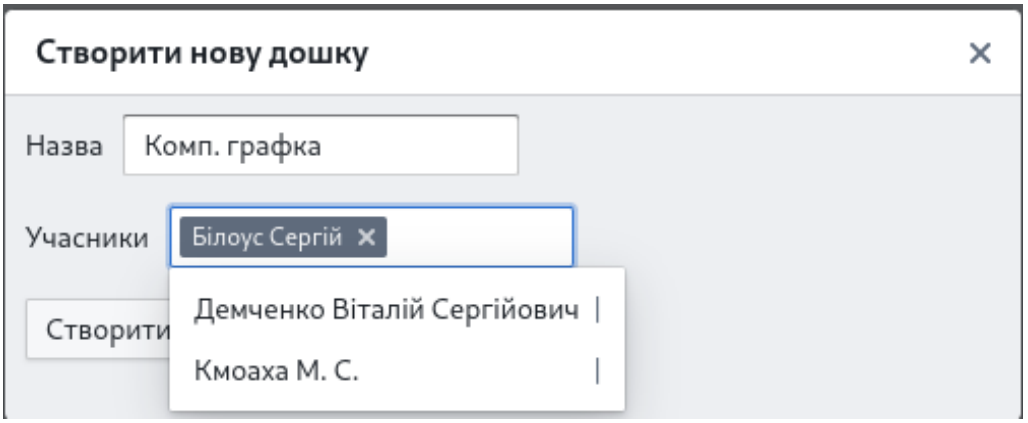


Рисунок 2.5 — Базовий макет інтерфейсу Kanban Board

Після створення користувач, що створив дошку повинен мати змогу відредагувати етапи які буде проходити завдання їх назву та послідовність. Також важливою функцією, є налаштування прав на пересування задач з певного стовпчика, та пересування в певний стовпчик. До прикладу тільки викладач має право пересувати завдання в стовпчик “Виконано”, та після того, як завдання опинилося в “готово до перевірки” студент не може перемістити його на будь-який інший етап.

Створення Kanban Board, та налаштування проходження етапів розглянуто вище, далі необхідно розглянути вимоги до представлення завдань в програмному додатку - кожне завдання має два візуальних представлення перший - це представлення в вигляді картки на Kanban Board, та другий - це розгорнутий вигляд де буде міститися повна інформація про завдання.

Картка задачі повинна мати мінімальну кількість інформації оскільки збільшення розміру картки веде до зменшення кількості карток, які одночасно

можуть показуватися на екрані, що призводить до зменшення ефективність використання Kanban Board. Для студента на картці повинно бути зображено назву завдання, та дату здачі. Етап на якому знаходиться завдання зрозумілий оскільки задача знаходиться в певному стовпчику. Виходячи з вимог для студента отримуємо базовий макет інтерфейсу картки задачі (рис. 2.6).

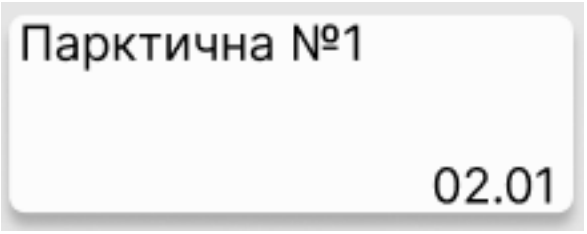


Рисунок 2.6 — Базовий макет інтерфейсу картки задачі

Лаконічне зображення завдання, проте дошкою будуть користуватися не тільки студенти, а й викладачі. Проблеми які, є в поточному рішенні - викладач на дошці буде бачити безліч однакових завдань, та не зрозуміло хто виконує кожне з цих завдань. Для розв'язання цієї проблеми до попереднього макету додамо зображення виконавця завдання, та унікального “id” завдання. Унікальний “id” зручний коли студенти звертаються з питаннями, сказати “id” швидше, аніж шукати завдання по назві, та прізвищу студента. Враховуючи ці вимоги було вдосконалено базовий макет інтерфейсу картки задачі (рис. 2.7).

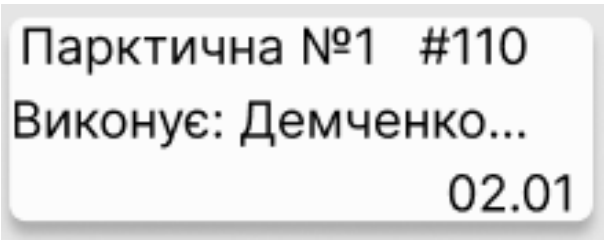


Рисунок 2.7 — Вдосконалений макет інтерфейсу картки задачі

Розгорнутий вигляд завдання повинен мати повну інформацію про завдання, а саме інформацію про те хто виконує завдання, хто створив, або іншими словами видав завдання, етап на якому знаходиться дане завдання, крайня дата виконання

завдання та найголовніше опис завдання. Цього буде достатньо для вирішення першої та другої проблеми, які були описані вище, проте проблема комунікації між викладачем та студентом залишиться без змін, для вирішення цієї проблеми необхідно реалізувати простий чат прикріплений внизу кожної задачі, таким чином, якщо в студента виникнуть питання по конкретному завданню він зможе задати ці питання не на пошту викладачу, а в чат прикріплений до задачі. Дане рішення значно спрощує комунікацію між викладачем та студентом по певному завданню. На основі вимог описаних вище було створено макет інтерфейсу розгорнутого завдання рис. 2.8.

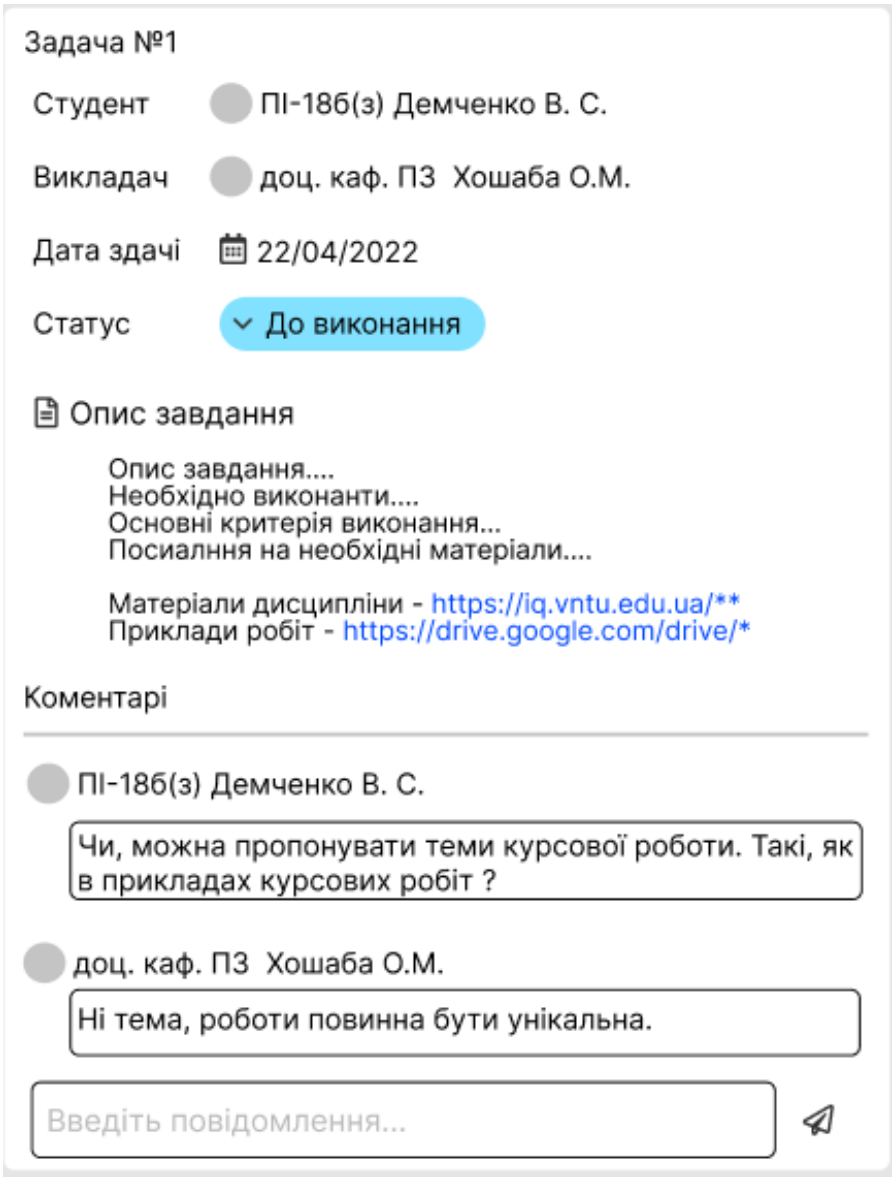


Рисунок 2.8 — Макет інтерфейсу розгорнутої задачі

До важливої функціональної вимоги стосовно канбан дошки, та завдання є доступ до певної дошки, та задачі за посиланням. Це необхідно, для швидкої комунікації, до прикладу замість пошуку задачі викладачу можна надіслати посилання на цю задачу, або викладач замість розповіді про те де в системі шукати дошку по даному предмету може надіслати на посилання, до певної дошки.

В процесі виконання дипломної роботи було розроблено макети графічного інтерфейсу до більшості елементів програмного додатка, що дало змогу усунути недоліки графічного інтерфейсу, ще до старту реалізації в коді.

Макети графічного інтерфейсу користувача було розроблено за допомогою векторного онлайн-сервісу розробки інтерфейсів Figma.

2.3 Ефективний алгоритм зміни послідовності об'єктів

Програмний додаток який реалізує Kanban Board містить велику кількість методів які задіюють зміну послідовності певних об'єктів. До прикладу, під час конфігурації канбан дошки зміна послідовності стовпців, або що більш критично зміна послідовності задач в стовпці оскільки кількість завдань може бути значною, необхідно розробити ефективний алгоритм зміни послідовності.

Вхідні дані масив об'єктів “objs”, кожен об'єкт містить поле “id‘ унікальний ідентифікатор, та поле “ord” місце в послідовності в якій знаходиться даний об'єкт. Та другим аргументом массив цифр “ordIds”, де значення відповідає “id” об'єкта, а індекс в масиві відповідає позиції в послідовності яку повинен мати об'єкт.

Вихідні дані, список об'єктів “objs” з зміненим полем “ord” в кожному об’єкту відповідно до того місця в послідовності яку повинен займати даний об’єкт .

Перший алгоритм який приходить при вирішенні даної задачі, а саме лінійно проходитися по масиву об'єктів, і кожен раз лінійно шукати значення “ord” в масиві є неефективним. Складність наївного алгоритму буде лінійний прохід по масиву “objs” це O(n) і на кожному елементі лінійний пошук “id” в масиві

“ordIds”, складність O(m), сумарна складність O(n) * O(m). Оскільки кількість елементів в “objs” та в масиві “ordIds”, то m = n і складність алгоритму квадратична $O(\ n^2\ )$.

Квадратична складність алгоритму, за умови потенційно великої кількість задач та одночасних запитів призведе, до проблем з продуктивність програмного додатка.

Удосконалити цей алгоритм можна за рахунок застосування бінарного пошуку.

Двійковий пошук порівнює цільове значення із середнім елементом масиву. Якщо вони не рівні, половина, в якій не може лежати ціль, усувається, і пошук продовжується на половині, що залишилася, знову порівнюючи середній елемент із цільовим значенням, і повторюючи це, доки цільове значення не буде знайдено. Якщо пошук закінчується тим, що половина, що залишилася, порожня, ціль не входить до масиву. У гіршому випадку двійковий пошук виконується за логарифмічний час, порівнюючи O(log n), де n – кількість елементів у масиві [22].

Обмеженням двійкового пошуку є те, що даний алгоритм може працювати, лише з відсортованими масивами. Для сортування масиву “objs” в даній роботі було використано стандарту функцію сортування, мови програмування JS, вона використовує ефективний алгоритм сортування, алгоритмічна складність виконання якого в середньому O(n * log n).

Алгоритмічна складність вдосконаленого алгоритму зміни послідовності об’єктів є O(n * log n) попереднє сортування масиву “objs”, лінійний прохід по масиву “ordIds” і на кожному елементі виклик функції пошуку відповідного об’єкта за “id” складність якого в гіршому випадку O(log n).

$$O_{\text{алг}} = O(n\ *\ log(n)\ )\ +\ O(n)\ *\ O(log(n))\ =\ 2\ *\ O(n\ *\ log(n))\ ) =$$

$$= O(n\ *\ log(n)\ )$$

O(n * log(n)) є значно ефективніше за квадратичну складність, яку мав наївний алгоритм. UML діаграма вдосконаленого алгоритму зображена на рис. 2.9.

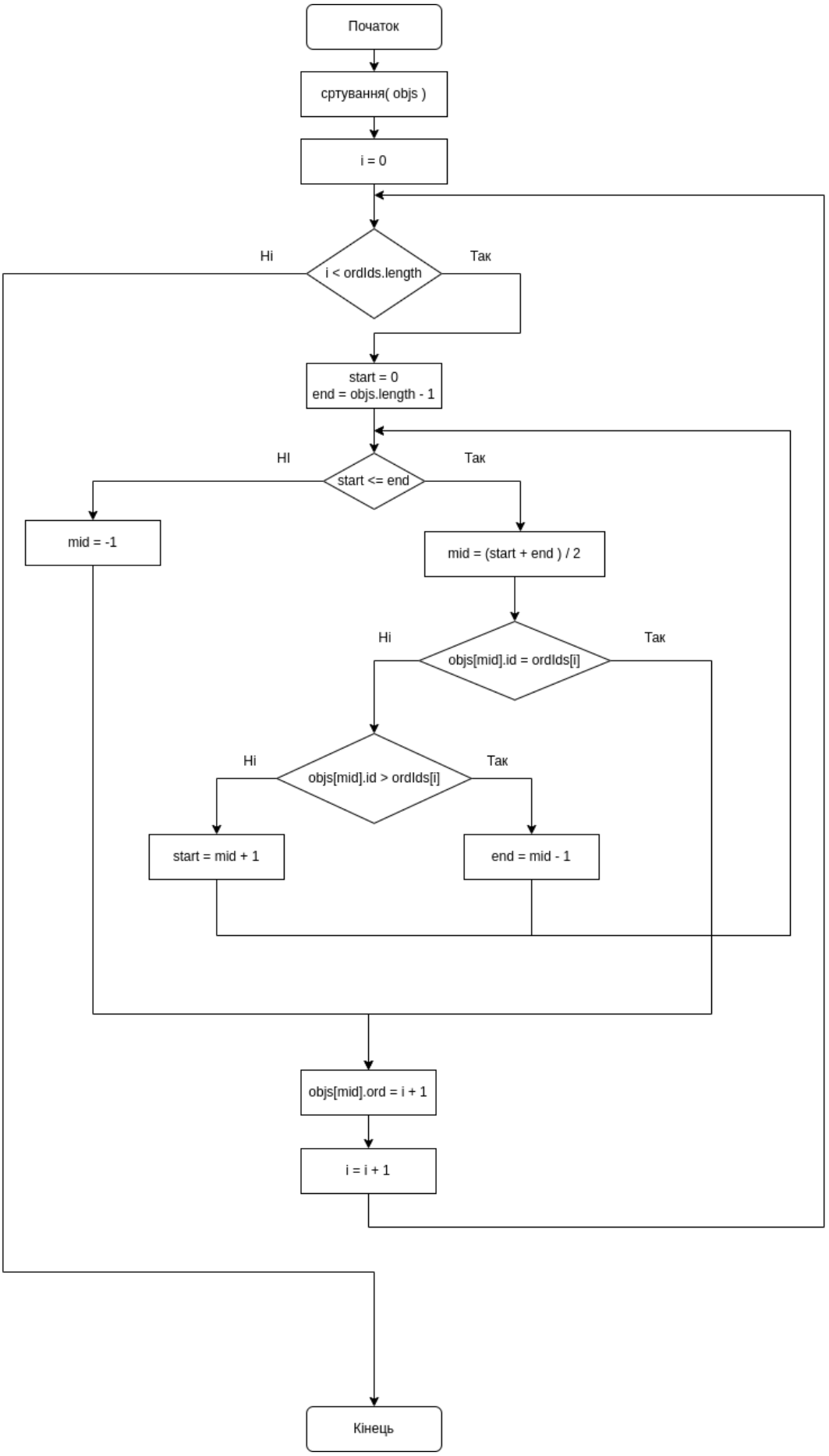


Рисунок 2.9 — Блок схема вдосконаленого алгоритму зміни послідовності об’єктів

2.4 Структура проекту

Програмний додаток є досить великим має багато екранів між якими можливі переходити. Тому необхідно формалізувати, які саме екрани є в програмному додатку, та які зв’язки між ними.

В програмному додатку є такі екрани:

«Вхід в систему» - перший екран де користувача просять, авторизуватися ввівши логін та пароль від облікового запису.

«Головна» - тут буде розміщений список всіх предметів до яких має відношення користувач. Також частиною цього екрану буде шапка в якій є інформація про користувача, та кнопки переходу до створення нової дошки, нового користувача, за умови, що в даного користувача є на це права.

«Створення користувача» - форма в якій пропонується ввести дані нового користувача.

«Створення Kanban Board» - форма для створення Kanban Board.

«Предмет» - тут будуть відображатися всі дошки, за певним предметом, та буде можливість взаємодіяти з ними.

«Завдання» - розгорнута інформація про завдання.

Для відображення зв'язків між екранами було розроблено діаграму переході станів (рис. 2.10).

Діаграма переходів станів є графічною формою надання кінцевого автомата – математичної абстракції, що використовується для моделювання детермінованої поведінки технічних об'єктів чи об'єктів реального світу [13].

В діаграмі переходів, вказано всі можливі екрани на яких може перебувати користувач, та які можливості для переходу він має на кожному з екранів. Створення даної діаграмі на початкових етапах, допомогає уникнути проблем не логічності переходів та надлишкової заплутаності інтерфейсу.

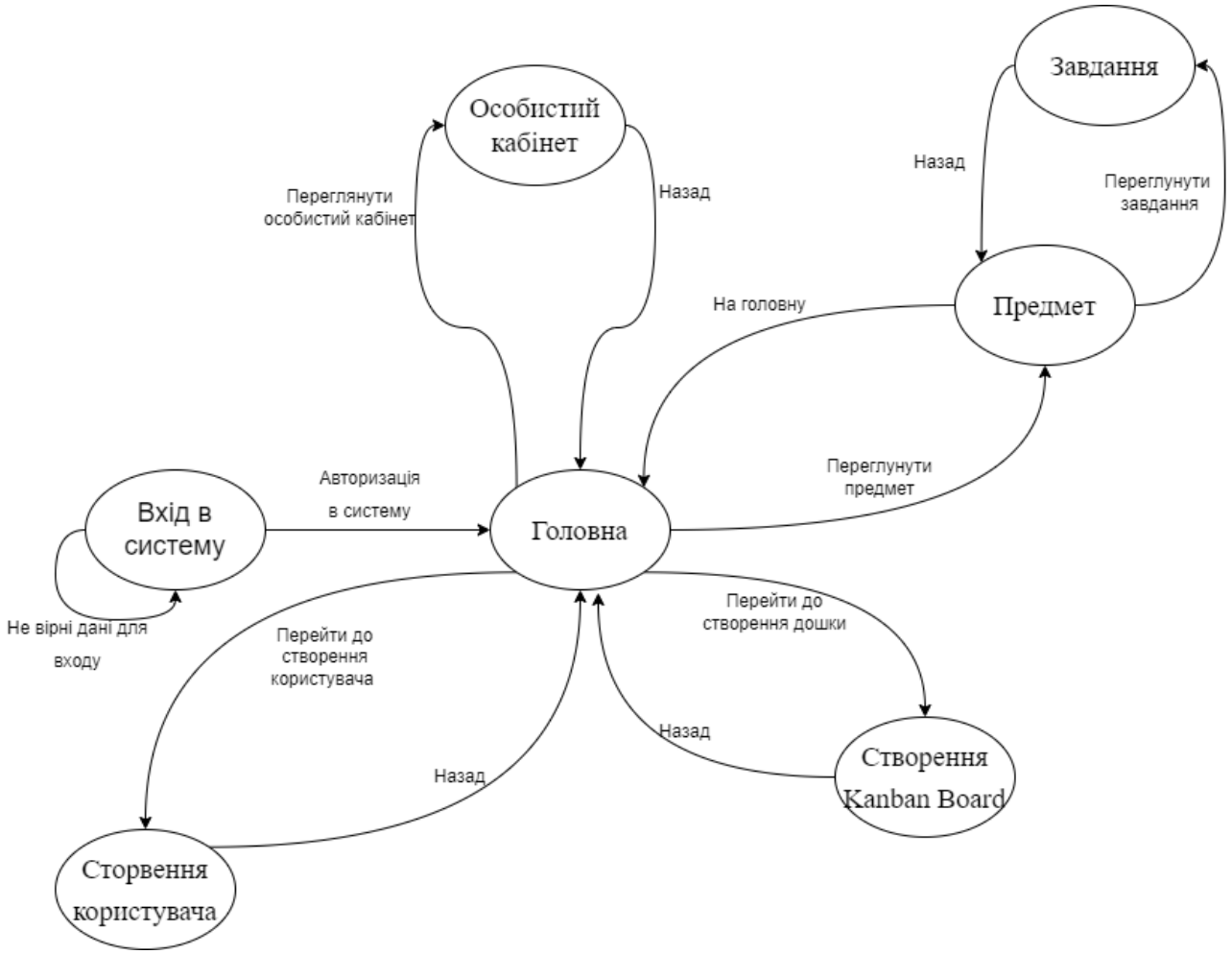


Рисунок 2.10 — Діаграма переходів

Деякі переходи зображені на діаграмі доступні лише при наявності в користувача прав.

«Перейти до створення користувача» може лише користувач з правами адмін.

«Перейти до створення дошки» - можуть лише користувачі з правами адмін, або викладач.

2.4 Висновки

У другому розділі було розглянуто процес отримання та здачі завдань студентами, та встановлені вимоги до інтерфейсу та функціоналу, який необхідний для покращення взаємодії між студентами та викладачами за рахунок впровадження розробляємого додатку. Розроблено графічні макети інтерфейсу які

відповідають функціональним вимогам, до додатку та містять лише необхідний набір елементів, що б зберегти лаконічність інтерфейсу. За допомогою графічних макетів було виявлено певні недоліки, інтерфейсу на етапі проектування до безпосередньої реалізації в коді.

Розроблено ефективний алгоритм зміни порядку об’єктів оскільки додаток, який реалізує Kanban Board містить велику кількість об’єктів порядок, яких необхідно зберігати, та доступ до зміни порядку, яких має користувач. До прикладу порядок задач в стовпці. Удосконалений алгоритм, є значно ефективнішим за наївну версію алгоритму.

Створена діаграма переходів між станами програмного додатку, для візуального представлення, яким чином користувач буде взаємодіяти з системою.

3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Вибір архітектури програмного додатку

Існує декілька архітектурних підходів, які дозволяють реалізувати необхідний функіонал для даного програмного додатку, кожне з рішень має свої переваги та недоліки тому важливою частиною розробки програмного забезпечення є вибір архітектури, з урахуванням подальшого можливого розвитку та масштабування проекту.

Виходячи з вимог, що програмним додатком одночасно повинно мати можливість користуватися велика кількість людей в рамках однієї системи, і вони повинні мати можливість взаємодіяти між собою. Було розглянуто два популярних архітектурних рішення, які підходять для реалізації поставлених вимог клієнт-серверна архітектура, та клієнт-клієнт архітектура.

Архітектура клієнт-клієнт — це архітектура розподіленого додатка, яка розподіляє завдання або робочі навантаження між одноранговими вузлами. Кожен з клієнтів є однаково привілейованими, рівносильними учасниками програми. Кажуть, що вони утворюють однорангову мережу вузлів [26] (рис. 3.1).

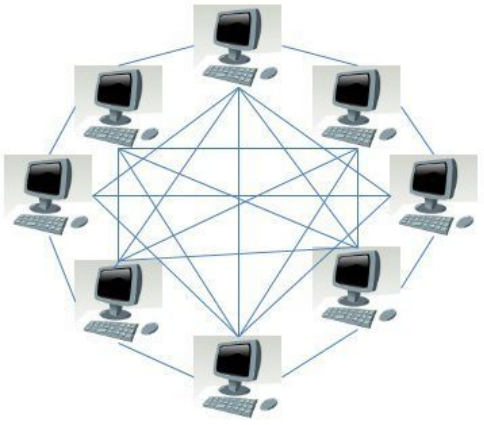


Рисунок 3.1 — Архітектура клієнт-клієнт [26]

Перевагою даного архітектурного рішення, є відсутність додаткових вузлів окрім клієнтів, система розподілена і не залежить від несправності певного вузла. Проте реалізація даної архітектури є досить складним завданням, оскільки дані дії

які робить один клієнт повинні бути синхронізовані з діями усіх інших клієнтів, і не існує єдиного джерела правди для всієї системи.

Другий популярний архітектурний підхід архітектура клієнт-сервер — це архітектура комп'ютерної мережі, в якій багато клієнтів запитують і отримують послугу від централізованого сервера (хост комп'ютера). Клієнтські комп'ютери забезпечують інтерфейс, який дозволяє користувачеві комп'ютера запитувати послуги сервера та відображати результати, які повертає сервер. Сервери чекають надходження запитів від клієнтів, а потім відповідають на них. В ідеалі сервер надає клієнтам стандартизований прозорий інтерфейс, щоб клієнтам не потрібно було знати про особливості системи (тобто апаратного та програмного забезпечення), яка надає послугу (рис. 3.2) [23].

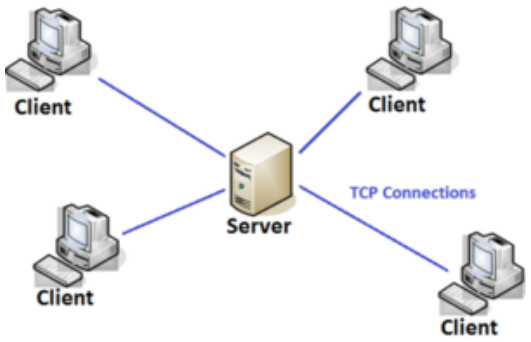


Рисунок 3.2 — Клієнт серверна архітектура [23]

Дане архітектурне рішення розділяє логіку роботи з даними, та збереження цих даних, та місце де відбувається взаємодія з цими даними. Що в свою чергу дає змогу взаємодіяти великій кількості людей в рамках однієї системи при цьому користуючись, абсолютно різними клієнтами. В ролі клієнта може виступати будь яка програма, або засіб які вміють робити запити через інтернет згідно певних протоколів. В ролі сервера виступаю комп'ютер до якого клієнти мають можливість робити запити.

Порівнюючи два архітектурні підходи для даної роботи було обрано використанням клієнт-серверної архітектури через більш просту реалізацію та оскільки вона повністю задовольняє потребам даного проекту.

Архітектура додатку обрана, наступним кроком необхідно розглянути детальніше протоколи взаємодії між вузлами клієнт-серверна архітектура каже, що повинен бути сервер, який надає інтерфейс взаємодії і що до нього повинні звертатися клієнти, стосовно того яким чином відбувається це звернення не уточнюється тому далі необхідно обрати рішення для взаємодії між клієнтом і сервером.

Інтерфейс який надає сервер для взаємодії називається API — це зв'язок між комп'ютерами або між комп'ютерними програмами. Це тип програмного інтерфейсу, який пропонує послуги іншим частинам програмного забезпечення [24]. Саме від API залежить яким чином клієнти будуть взаємодіяти з сервером. Існують кілька популярних на сьогоднішній день архітектурні рішення для побудови API сервера, а саме SOAP, REST, GraphQL, RPC (рис. 3.3).

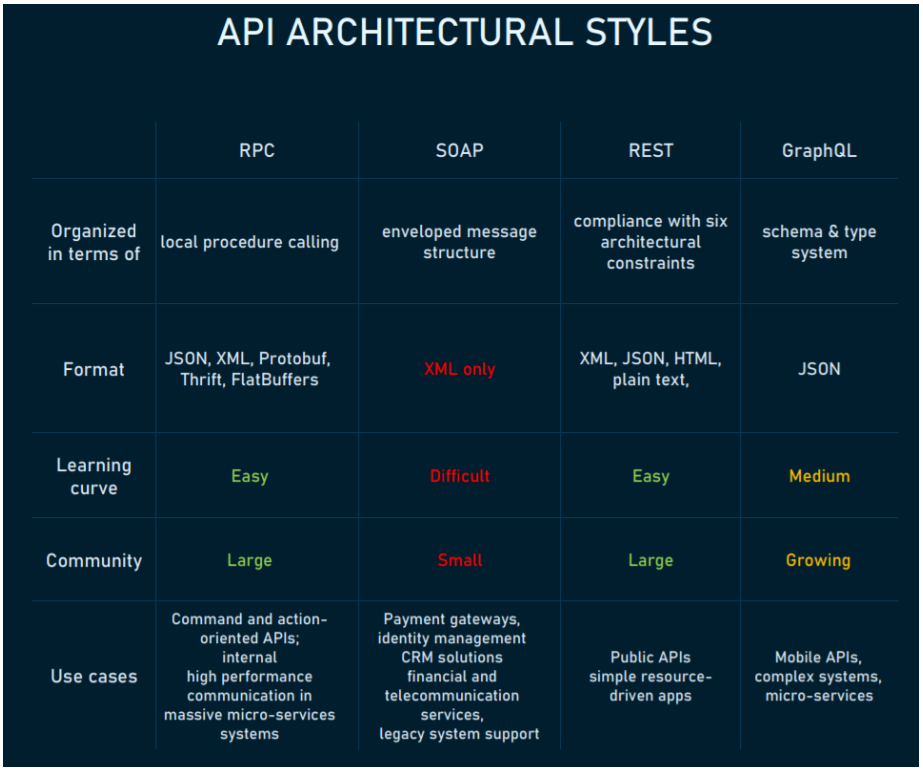


Рисунок 3.3 — Порівнянн аріхтекруних стилів створення API [25]

Кожен з підходів має свої недоліки та переваги. Зробивши аналіз було обрано використовувати REST API, як один з найпростіших та найпопулярніших підходів до створення API.

3.2 Обґрунтування вибору технологій для реалізації клієнту.

В попередньому підрозділі було обрано клієнт-серверну архітектуру, програмного додатку наступним кроком необхідно проаналізувати та обрати найоптимальніші технології для реалізації клієнтської та серверної частини дотаку. Існує велика кількість технологій для розробки програмного забезпечення, кожна з яких має свою сферу застосування переваги та недоліки. Тому, важливою частиною розробки програмного додатку є обрання оптимальних технологій, для реалізації.

Клієнт програмного додатку повинен працювати на великій кількості різних девайсів, тому як кандидати для розробки клієнтської частини додатку було розглянуто кросплатформені мови програмування, такі як C#, Java, JavaScript, Typescript.

Більш детально про кожну з цих мов:

C# — це мова програмування, розроблена та запущена компанією Microsoft у 2001 році. C# — це проста, сучасна й об’єктно-орієнтована мова, яка надає сучасним розробникам гнучкість та можливості для створення програмного забезпечення, яке не тільки працюватиме сьогодні, але й буде застосовуватися протягом багатьох років. майбутнє [26].

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи [27].

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб сторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб сторінки. JavaScript класифікують, як прототипу скрипту мову

програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу [28].

TypeScript — мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки веб застосунків, що розширює можливості JavaScript. Розробником мови TypeScript є Андерс Хейлсберг (англ. Anders Hejlsberg), який створив раніше C#, Turbo Pascal і Delphi. Код експериментального компілятора, котрий транслює код TypeScript в представлення JavaScript, поширюється під ліцензією Apache, розробка ведеться в публічному репозиторії через сервіс CodePlex. Специфікації мови відкриті і опубліковані в рамках угоди Open Web Foundation Specification Agreement (OWFa 1.0). TypeScript є зворотньо сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js [29].

Кожна з наведених мов програмування може бути використана для реалізації клієнту. Підач порівняння мов програмування було виділено наступні критерії мова повинна бути об’єктно орієнтована, мати строгу типізацію, бути простою та мати високу швидкість розробки, мати зручні бібліотеки для створення графічного інтерфейсу та роботи з REST API.

Недоліком мови програмування для даного проекту, є наявність стадії компіляції при розробці на певній мові програмування, оскільки це значно збільшує час розробки. Компільована мова програмування буде більш ефективною з точки зору використання ресурсів, проте розробляємо клієнтська частина додатку не високонавантажених частиною проекту, інтерпретована, або JIT компільована мова програмування в данному випадку переважає над компільованою.

Також бажаним критерієм є можливість виконання мови програмування в браузере оскільки на відміну від нативних програм, користувачу немає

необхідності завантажувати та встановлювати будь, що на свій пристрій, що значно покращує користувацький досвід.

Враховуючи сформовані критерії вище, було проведено аналіз мов програмування, було складено таблицю (табл. 3.1) в якій демонструються недоліки та переваги мов програмування C#, JavaScript, TypeScript.

Таблиця 3.1 – Порівняння особливостей мов програмування

Особливості	Мова програмування			
	C#	Java	JavaScript	TypeScript
Об’єктно-орієнтована	так	так	ні	так
Компільована	так	так	ні	компіляція в JavaScript
Швидкість розробки	відносно швидко	відносно швидко	швидко	швидко
Наявність строгої типізації	так	так	ні	так
Легкість використання	так	так	не завжди очевидний результат виконання	так
Зручність бібліотек	так	більшість зручні	так	так
Можливість виконання в браузері	ні	ні	так	так

Виходячи з порівняння мов програмування мовою програмування для клієнту було обрано TypeScript. Оскільки, це об'єктно-орієнтована мова програмування, яка має строгу типізацію, та можливість виконання в браузері. З недоліків, на відміну від JavaScript дана мова програмування має стадію компіляції.

В якості клієнту було обрано розробити односторінковий сайт використовуючи мову програмування TypeScript. Для роботи з REST API було використано стандартне Fetch API браузера. Для відображення графічного інтересу в браузері використовується HTML (мова розмітки гіпертексту) та CSS

(каскадні таблиці стилів) які є основними технологіями для створення веб-сторінок. HTML надає структуру сторінки, CSS — (візуальний) макет для різних пристроїв. інтерфейсу стандартною мовою розмітки для документів, призначених для відображення у веб-браузері [30].

В якості архітектурного рішення для клієнта було обрано застосування широко розповсюдженого в веб розробці паттерна MVC (рис. 3.4).

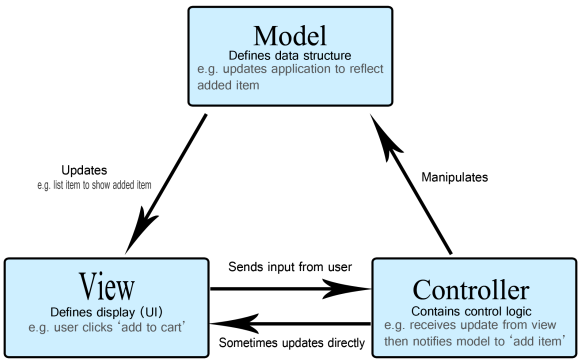


Рисунок 3.4 — Діаграма патерну проектування MVC

MVC (Model-View-Controller) — це шаблон у розробці програмного забезпечення, який зазвичай використовується для реалізації інтерфейсів користувача, даних і логіки керування. Він підкреслює поділ між бізнес-логікою програмного забезпечення та відображенням даних [40].

В ході розробки були використані сучасні бібліотеки розробки веб додатків, які впроваджують перевірені часом підходи до розробки. Бібліотека React дозволяє розробникам створювати великі веб-додатки, які можуть змінювати дані, не перезавантажувач сторінку. Основна мета React — бути швидким, масштабованим і простим. Дана бібліотека працює лише на користувацьких інтерфейсах програми. Це відповідає “View” в шаблоні MVC. Його можна використовувати з комбінацією інших бібліотек або фреймворков JavaScript, таких як Angular JS у MVC [31].

Управління глобального стану клієнтської частини було реалізовано за допомогою бібліотеки Redux — це бібліотека JavaScript з відкритим кодом для

керування та централізації стану програми [32]. В архітектурному підході MVC Redux виконує роль моделі та контроллера. Загалом зміна стану та відображення відбувається наступним чином в Redux (рис. 3.5).

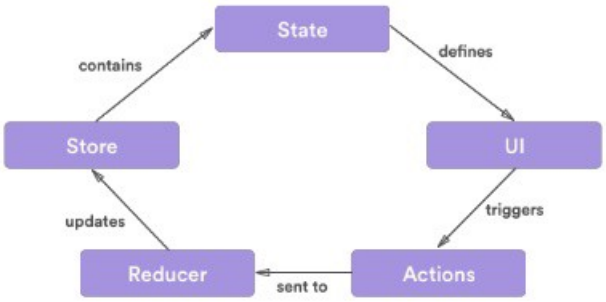


Рисунок 3.5 — Управління станов в бібліотеці redux

Написання стилів для компонентів було реалізовано за допомогою css-module, даний підхід включає можливе співпадіння селекторів стилів і робить не потрібним використання іменування класів html елементів за методологію BEM.

Для вирішення проблеми роутінгу, що б при переході за клієнт переходив на певну стадію, до прикладу певний Kanban Board була використана бібліотека React Router.

Одним з критеріїв до програмного додатку, є можливість прикріпити до задачі опис. Засобів редагування, які надає стандартний html элемент “textarea” є недостатнім, оскільки не покриває потребу додаванні картинок, посилань та не надає зручних засобів форматування тексту. Необхідний повноцінний редактор тексту тому було обрано використати бібліотеку “react-md-editor” (рис. 3.6).

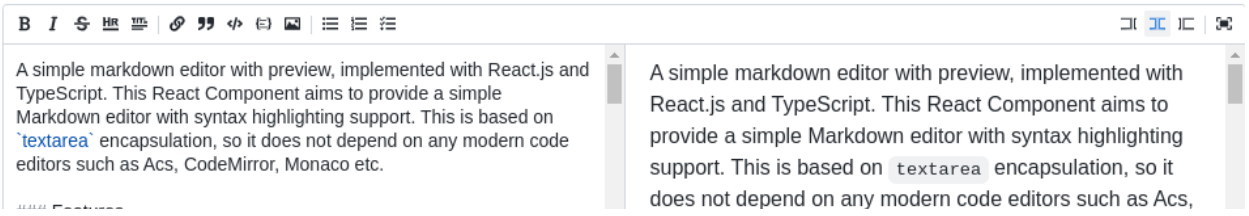


Рисунок 3.6 — Інтерфейс редактора тексту для опису завдання “react-md-editor”

3.3 Обґрунтування вибору технологій для реалізації серверу

У попередньому підрозділі були описані обрані технології та бібліотеки для розробки клієнтської частини проекту. Сервер є незалежною частиною системи, і потенційно в майбутньому до програмного додатку можуть бути розроблені нові клієнти, які будуть працювати з тим самим сервервером завдяки тому, що спілкування між клієнтом і сервером відбувається за допомогою платформно незалежного REST API.

Серед мов програмування, які можливо використати для реалізації серверної частини розглядались ті самі, мови що й для клієнту, але ще як потенційний варіант розглядається мова програмування GoLang оскільки вона набула широкої популярності в останній рокі саме як мова для розробки серверної частини програмних додатків.

Зробивши порівняння мов програмування було обрано на сервері використовувати також TypeScript, основною причиною вибору стало використання однакової мови програмування на клієнті та сервері, що дало змогу винести опис інтерфейсів REST API окремо і імпортувати їх незалежно в клієнт та сервер, що дає безпеку типів, якщо щось зміниться на сервері і сервер почне повертати не коректний тип компилятор TypeScript помітить цю проблему.

Оскільки TypeScript компілюється в JavaScript для виконання серверного коду була використана платформа Node.js.

Node.js — це кросплатформне середовище виконання JavaScript із відкритим вихідним кодом, яке працює на движку V8 і виконує код JavaScript за межами веб-браузера [32].

Для реалізації REST API було використано бібліотеку Express, — це серверний фреймворк веб-додатків для Node.js, випущений як безкоштовне програмне забезпечення з відкритим вихідним кодом під ліцензією MIT. Він призначений для створення веб-додатків та API. Його називають де-факто стандартною серверною платформою для Node.js [33].

На сервері необхідно зберігати дані для цього використовуються бази даних, для роботи з базами даних було обрано бібліотеку Sequelize — це заснований на

промісах Node.js ORM (об’єктно-реляційне відображення) для Postgres, MySQL, MariaDB, SQLite та Microsoft SQL Server. Він має надійну підтримку транзакцій, відносини, швидке й ліниве завантаження, реплікацію читання тощо.

З усіх прийнятих рішень при розробці програмного додатку використання Sequelize є найбільш спірним рішенням. Дана бібліотека була вибрана оскільки ORM модель автоматично генерує типи і таким чином забезпечує безпеку типів між сервером та базою даних. Проте виникли проблеми при роботі з асоційованими полями, які пов'язані з невдалими рішеннями в дизайні бібліотеки для використання з строгою типізацію. А саме при встановленні асоціації в моделі автоматично додаються методи, для роботи з асоційованим полем ці методи система типів TypeScript не бачить, і тому доводиться використовувати ігнорування перевірки типів. Дана проблема виникла через, те що бібілотека написана на мові програмування JavaScript, і лише в подальшому були описані типи для використання цієї бібліотеки в TypeScript.

Засобом адміністрування бази даних було обрано DBeaver (рис. 3.7).

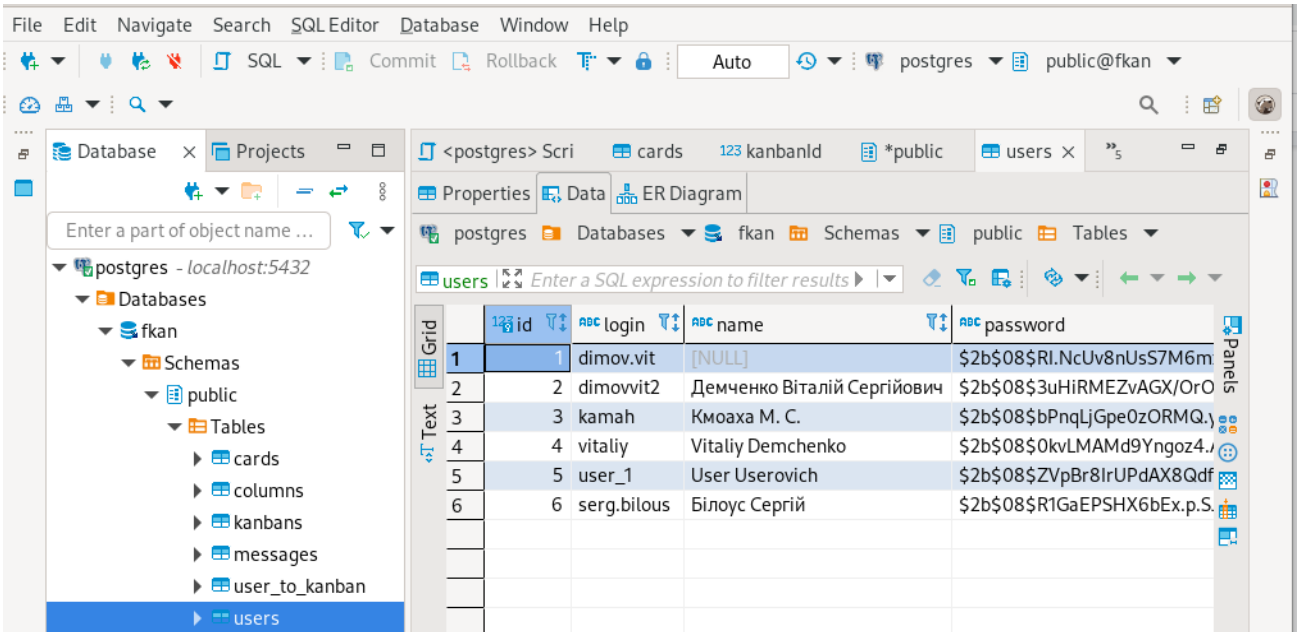


Рисунок 3.7 — Інтерфейс програмного додатку DBeaver

DBeaver — це клієнтська програма SQL та інструмент адміністрування бази даних. Для реляційних баз даних він використовує прикладний програмний

інтерфейс JDBC для взаємодії з базами даних через драйвер JDBC. Для інших баз даних він використовує власні драйвери баз даних [39].

3.4 Допоміжні засоби розробки програмного додатку.

В якості середовища розробки було обрано Visual Studio Code (рис. 3.8).

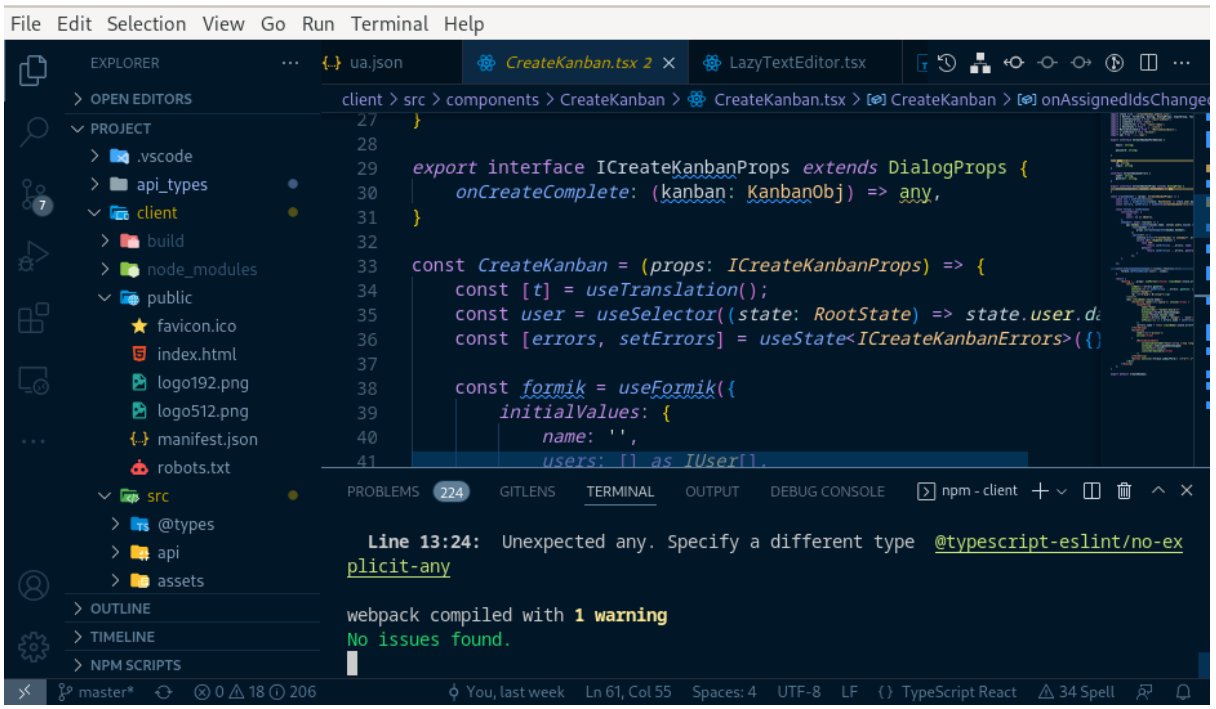


Рисунок 3.8 — Інтерфейс програмного додатку Visual Studio Code

Visual Studio Code — це легкий, але потужний редактор вихідного коду, який працює на вашому робочому столі та доступний для Windows, macOS та Linux. Він поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов (наприклад, C++, C#, Java, Python, PHP, Go) і середовищ виконання (наприклад, .NET і Unity) [14].

Середовище розробки було обрано через швидкодію та гнучкість даного рішення. Без встановлених додатків, це майже звичайний редактор тексту, але завдяки плагинам може розширюватися до повноцінної IDE при цьому буде містити лише ті засоби, якими користується розробник, що позитивно впливає на швидкодію.

Розробка додатку велася в Git репозиторі, що дало можливість контролювати версію додатку, експериментувати з певними технологіями та повертатися до попередньої версії додатку, якщо експерименти виявилися не успішними.

Git — це програмне забезпечення для відстеження змін у будь-якому наборі файлів, яке зазвичай використовується для координації роботи програмістів, які спільно розробляють вихідний код під час розробки програмного забезпечення. Його цілі включають швидкість, цілісність даних і підтримку розподілених, нелінійних робочих процесів [34].

Потрібно зауважити, що клієнт та сервер знаходяться в одному репозиторії оскільки специфічна версія клієнта, працює лише з специфічною версією сервера і навпаки, тому для запобігання ситуації коли поточні коміти клієнту та серверу не сумістимі, було прийнято рішення тримати все в одному репозиторії і кожен коміт цьому репозиторії точно має сумістиму версію клієнту та серверу.

3.5 Вирішення проблеми надійного зберігання даних на сервері

В більшості проектів де є вузол сервер вирішується задача постійного зберігання даних. Найпростішим варіантом є збереження даних файлами в файловій системі. Даний підхід наївному виконанні без суттєвих доопрацювань не забезпечує безпеку даних, ефективність її обробки, та можливість масштабування при рості кількості даних. Реалізація засобів, які відповідають критеріями є досить складною задачею.

Тому в більшості випадках використовують готові рішення для надійного забезпечення та роботи з даними даних - бази даних.

У обчислювальній техніці база даних — це організована сукупність даних, які зберігаються та доступні в електронному вигляді. Невеликі бази даних можна зберігати у файловій системі, тоді як великі бази даних розміщуються в комп’ютерних кластерах або хмарних сховищах. Проектування баз даних охоплює формальні методи та практичні міркування, включаючи моделювання даних, ефективне представлення та зберігання даних, мови запитів, безпеку та

конфіденційність конфіденційних даних, а також проблеми розподілених обчислень, включаючи підтримку одночасного доступу та відмовостійкості [35].

Зазвичай база даних виступає в ролі окремого сервісу з яким спілкується сервер, і до якого напряму не має доступу клієнт що зображено на (рис. 3.9).

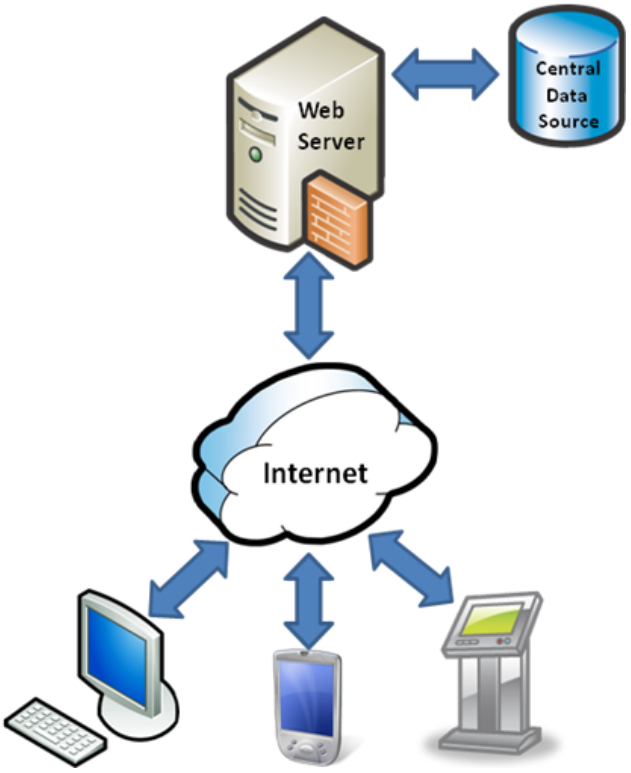


Рисунок 3.9 — Діаграма комунікації між клієнтом, сервером та базою даних

Найпопулярнішими є два види реляційні та документо орієнтовані бази даних. В даній дипломній роботі було обрано використовувати реляційну базу даних оскільки даний тип баз даних краще працює з обробкою складних запитів, якими можуть бути запити на отримання карток завдань на Kanban Board з складними фільтрами. В якості системи керуванн базою даних було обрано PostgresSQL.

Зазвичай створення таблиць в реляційній базі даних відбувається за рахунок виконання SQL запитів. Даний підхід, є перевіреним часом, проте недоліком є відсутність гарантій безпеки типів між сервером та базою даних. При оновлені структури бази даних необхідно кожен раз вручну проходитися по всьому коду серверу і перевіряти чи є запит до полів які вже відсутні в базі даних.

Вирішуючи цю проблемо було застосовано ORM підхід до роботи з базою даних. Об’єктно-реляційне відображення (ORM) — це техніка програмування для перетворення даних між системами типів за допомогою об’єктно-орієнтованих мов програмування. Це створює, по суті, «базу даних віртуальних об’єктів», яку можна використовувати з мови програмування [36]. Було використано бібліотеку Sequelize.

В процесі розробки програмного додатку виникла необхідність в створенні п’яти ORM моделей: user-model, kanban-model, column-model, card-model та message-model.

Створення моделі kanban відбувається в кілька кроків. Необхідно створити загальний інтерфейс моделі в файлі /apit_types/api.d.ts цей файл містить лише типи і створення для поширення інтерфейсів між клієнтом і сервером. Приклад коду:

```
interface KanbanObj {
    id: number,
    name: string,
    ownerId: number,
    columns?: number[] | IColumn[],
    users?: number[] | IUser[],
}
```

Код вище описує інтерфейс KanbanObj який містить id унікальний ідентифікатор канбан дошки, name - назву, ownerId - id користувача який створив дошку, columns - містить id всіх стовпців які належать до даного канбану, або дані про колонки залежить від запиту, users - список id всіх користувачів які мають відношення до даної дошки, або дані користувачів, які додані до даної дошки.

Після опису інтерфейсу необхідно створити модель за допомогою бібліотеки Sequelize.

Приклад коду:

```
type IKanbanCreation = Optional<KanbanObj, 'id'>;
class Kanban extends Model<KanbanObj, IKanbanCreation> {
        declare id: number;
        declare name: string;}
Kanban.init({
  id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
  name: { type: DataTypes.STRING, unique: true,}
}, { sequelize, modelName: 'kanban' });
```

Основна частина це Kanban.init де відбувається ініціалізація ORM моделі. Першим аргументом передається словник з полями та їх властивостями. Поле id має тип Integer, автоматично інкрементується при створенні нового екземпляру, та поле name яке має тип String та є унікальним.

В частині прикладу коду створюється тип IKanbanCreation, це тип який опису аргументи, які необхідні для створення нового екземпляру. Оскільки id поле заповнюється автоматично, для створення воно, є не обов'язковим аргументом.

При ініціалізації моделі не створюються асоційовані поля оскільки спочатку необхідно створити всі моделі, а потім встановити між ними залежності.

Приклад коду:

```
Kanban.belongsToMany(User, { through: 'user_to_kanban' });
Kanban.belongsTo(User, {foreignKey: { name: "ownerId" }, as: 'owner'});
User.belongsToMany(Kanban, { through: 'user_to_kanban' });
Kanban.hasMany(Column);
Column.hasMany(Card);
Card.belongsTo(User, {foreignKey: {  name: "creatorId" }, as: 'creator'});
Card.belongsTo(User, { foreignKey: { name: "assignedId" }, as: 'assigned'});
Message.belongsTo(Card, {foreignKey: { name: "creatorId" }, as: 'creator'});
```

На початку відбувається імпорт всіх моделей. Далі встановлення асоціацій між моделями. У одного користувача є багато канбан моделей та в одному канбані задіяно багато користувачів це відношення ManyToMany описується двома стрічками *“Kanban.belongsToMany(User…”* та *“User.belongsToMany(Kanban…”* В канбан дошки може бути лише один користувач який її створив відношення описується в термінології ORM канбан належить до одного користувача *Kanban.belongsTo(User*. Наступні стрічки коду описують взаємовідносини між іншими моделями.

Оскільки ORM це абстракція над справжньою базою даних після побудови всіх моделей в PostgreSQL отримаємо наступну ER діаграму зв'язку (рис. 3.10).

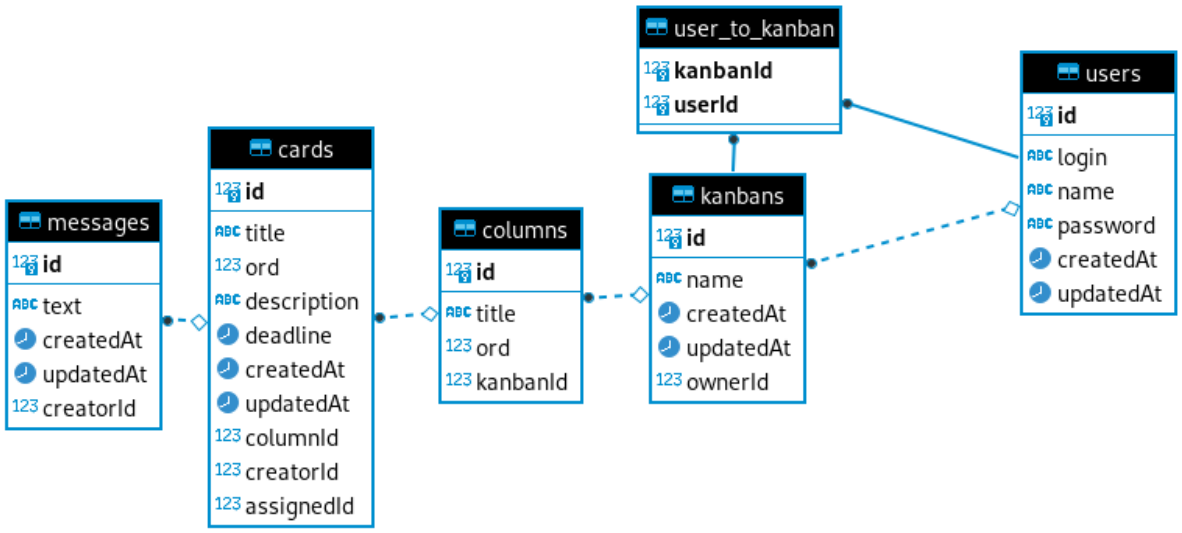


Рисунок 3.10 — ER діаграма

Діаграма було отримана за допомогою інтсрумента адміністрування бази даних DBeaver.

3.6 Розробка модуля авторизації з застосування JWT

При використанні клієнт-серверної архітектури доводиться вирішувати проблеми ідентифікації користувачів. Сервер обробляє велику кількість запитів одночасно, і йому необхідно знати, запит якого користувача він обробляє.

Наївним рішенням даної проблеми є додавання id користувача до запиту таким чином сервер може зчитати id клієнта та обробити запит з урахуванням цієї інформації. Недоліком є можливість клієнта підмінити цей id і видати себе за іншого користувача. В даному програмному додатку, користувач може отримати можливості до яких він не повинен мати доступ. Наприклад студент отримає можливість створити канбан дошку, або перевести завдання у стадію виконано. Найгірше, те що в системі навіть не збережеться хто саме скористався дірою безпеки, оскільки всі запити будуть надіслані з id користувача який мав доступи.

Отже додавання id користувача до запиту не є безпечним і будь який користувач може себе видати за будь кого. Наступним кроком вдосконалення ідентифікації користувача є впровадження дякого механізму авторизації. Перш ніж почати працювати з програмним додатком користувачу доведеться пройти авторизацію за логіном та паролем. Маючи валідний логін та пароль на клієнті можна надсилати цю пару з кожним запитом та перевіряти на стороні сервера чи дійсно цей користувач виконує запит.Виникає проблема зберігання логіну та пароля просити користувача кожен раз вводити логін та пароль не є гарним користувацьким досвідом тому логін та пароль необхідно десь зберігати на та при повторному відкритті сторінки отримати дані з сховища. В ролі такого постійного сховища можуть виступати файли cookie. Браузер може зберігати файл cookie та надсилати його назад на той самий сервер із наступними запитами. Зазвичай файл cookie HTTP використовується, щоб визначити, чи надходять два запити з одного браузера — наприклад, щоб користувач залишався в системі. Він запам'ятовує інформацію про стан для протоколу HTTP без стану [37].

В цій роботі опускається можливості перехоплення пакетів та отримання логіну та паролю третьою стороною, вважаючи що клієнт з сервером комунікує через захищений канал зв'язку по протоколу HTTPS.

Рішення наведене вище однозначно ідентифікує користувача проте є і свої недоліки. Створюється додаткове навантаження на базу даних на кожен запит від клієнта, серверу необхідно робити запит до бази даних та отримувати інформацію

про користувача, для перевірки валідності логіну та паролю. Вирішуючи проблему надмірного навантаження на базу даних був створений стандарт JWT.

JSON Web Token (JWT) — це відкритий стандарт (RFC 7519) для створення токенів доступу на основі JSON. Як правило, використовується для передачі даних для автентифікації в клієнт-серверних програмах. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи [38].

Користувач надсилає логін та пароль при авторизації. Сервер перевіряє валідність логіну та паролю. Після перевірки сервер генерує унікальний токен авторизації який надсилає клієнту у якості відповіді на запит авторизації. Клієнт зберігає даний токен в файли cookie, та надсилає даний токен з кожним запитом на сервер (рис. 3.11).

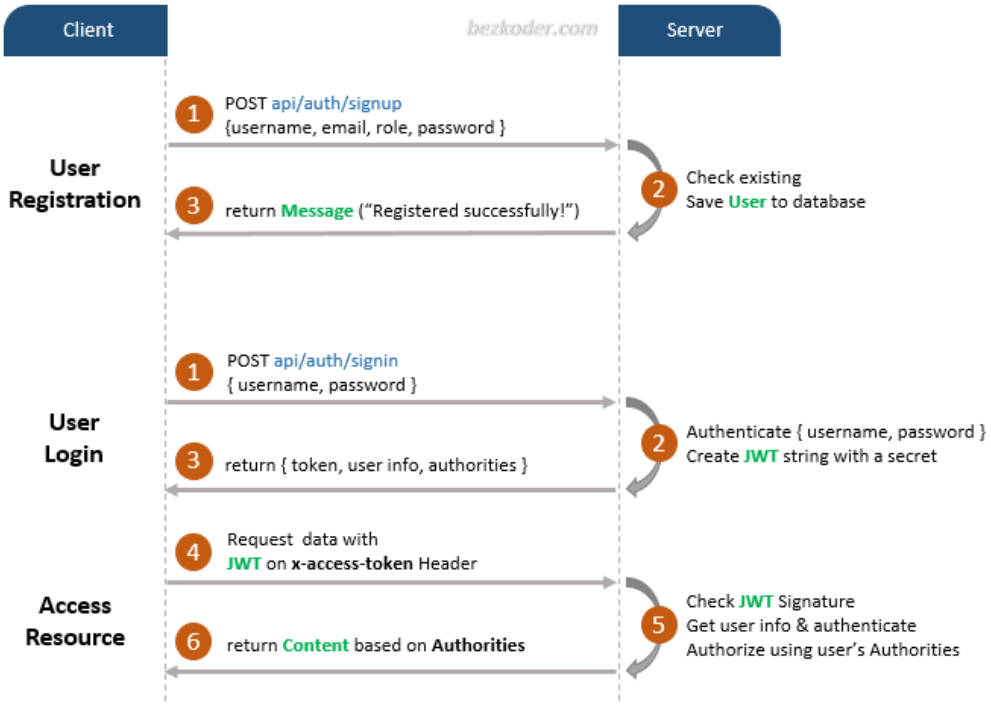


Рисунок 3.11 — Діаграма запитів з використанням JWT

Токен це зашифровані дані для розшифровки який необхідний ключ який знаходиться на сервері. Тому замість того, щоб робити запит в базу даних сервер отримує id користувача розшифровуючи токен.

Код авторизації користувача:

```
export const login = async (req: Request, res: Response<IErrorResponse | { user: IUser, accessToken: string }>) => {
  try { const user = await User.findOne({where: {login: req.body.login},raw: true,});
    if (!user) { return res.status(404).send({ error: "User Not found." });}
    const isValidUser = await bcrypt.compare(req.body.password,user.password );
    if (!isValidUser) {return res.status(401).send({error: "Invalid Password!"});}
 const token = jwt.sign({ id: user.id }, SECRET, {expiresIn: TOKEN_TIME_EXPIRE});
    res.status(200).send({user: { ...userDataFilter(user) },accessToken: token});
  } catch (err: any) { res.status(500).send({ error: err.message })}}
```

Приведений код описує функцію, яка обробляє авторизацію користувача. Дана функція викликається в “router” (термін з бібліотеки) бібліотеки express тому приймає два стандартних для даної бібліотеки аргумента req - об'єкт запиту, та res - об’єкт відповіді. Обробка помилок виконується за рахунок обгортання всього тіла функції в конструкцію “try{}catch(e){}” і в разі виникнення непередбачуваних помилки сервер поверне клієнту HTTP код 500 та текст повідомлення про помилку. Першою стрічкою виконується запит в базу даних, що б отримати інформацію про користувача. Наступною стрічкою перевіряється чи був знайдений даний користувач за логіном, якщо ні то клієнт отримає HTTP код помилки 404, та на цьому виконання функції припиниться. При наявності даних про користувача наступним кроком буде перевірка валідності даного користувача. Було прийнято рішення шифрувати всі паролі, які записуються в базу даних тому порівняння відбувається через бібліотеку, якою було проведене шифрування “bcrypt”. Якщо, користувач ввів не валідний пароль буде відправлена відповідь з HTTP кодом 401, що за стандартом є кодом відповіді на запит не авторизованого користувача. При проходженні валідації буде згенеровано token “jwt.sign(...)” та надіслано клієнту, даний токен клієнт повинен надсилати з кожним запитом, для підтвердження, що запит робить певний авторизований користувач.

Код перевірки токену авторизації “kanban-id-middleware.ts”:

```
export const verifyToken = (req: Request, res: Response, next: NextFunction) => {
  const token = req.header('x-access-token');
  if (!token) {return res.status(403).send({ error: "No token provided!"});}
  jwt.verify(token, config.SECRET, (err: any, decoded: any) => {
     if (err) {return res.status(401).send({error: "Invalid token!"});}
     req.userId = decoded.id;  next(); })};
```

Код наведений вище описує функцію “*verifyToken” яка використовується в якості* “middleware”. Дана функція викликається кожен раз перед обробкою запитів, які потребують перевірки JWT. До таких запитів відносяться всі окрім запиту авторизації “/auth/login”. Дана функція викликається бібліотекою “epxress” тому приймає аргументи req - об'єкт запиту, та res - об’єкт відповіді та останній аргумент це колбек функція next яка повинна бути викликана після завершення виконання функції “*verifyToken*”. Першою стрічкою з об’єкту “req” зчитується токен який надійшов з клієнту в хедері запиту. Якщо, токен не знайдений одразу відправляє клієнту відповідь з HTTP кодом 403. При наявності якогось токена відбувається його перевірка функцією “*jwt.verify”*. Дана функція приймає в якості аргументів токен, секрет з конфігу сервера, та функцію обробник, оскільки операція перевірки є асинхронною. В функції обробнику перевіряється чи є помилка, якщо є то означає що токен не валідний надсилаємо відповідь з HTTP кодом 401. При коректному проходженні верифікації токена записуємо id користувача в об’єкт “*req*” та викликаємо функцію “next()”.

Таким чином реалізується верифікація користувачів за допомогою JWT.

3.7 Оптимізація клієнтського додатку

При розробці веб додатків часто виникає проблема довгого завантаження. На Відміну від нативних програм, які завантажуються заздалегідь, веб додатки завантажуються в той момент коли користувач перейшов за посиланням. Тому для

гарного користувацького досвіду необхідно мінімізувати час очікування. Зробити це можна за допомогою оптимізації розміру файлів додатку.

Для дослідження процесу завантаження додатку було використано Chrome Dev Tools утиліту Network [41]. Кожен з наведених замірів в даному розділі був виконаний з однаковими налаштуваннями:

- Disable cahce - true - відключена функцією кешування файлів, що імітує перший вхід на сторінку;
- Bandwidth throttling - Fast 3G - обмеження пропускною задтності мережі, що імітує завантаження через 3G інтернет.

Виконавши профайлінг завантаження сторінки отримано наступні результати (рис. 3.12).

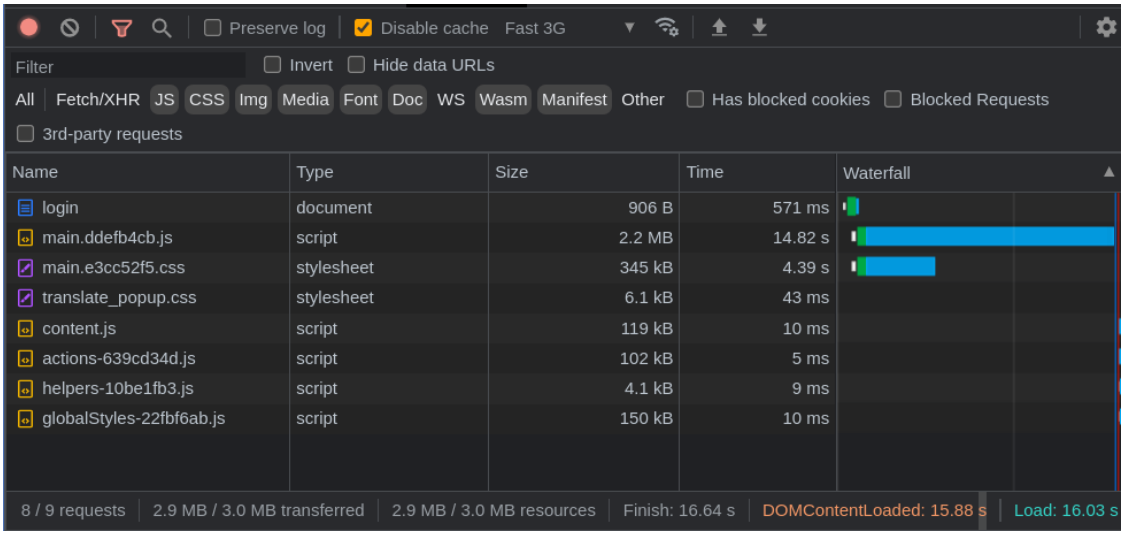


Рисунок 3.12 — Результати профайлінгу не оптимізованого додатку

Час завантаження без оптимізацій становить 16,5 секунд. Заміри були проведені декілька разів розбіжність в результатах 0,1 секунд, що є несуттєвим. Проаналізувавши (рис. 3.12) видно, що майже весь час завантаження витрачається на завантаження файлу main.ddefb4cb.js, який містить весь код программи. Хеш в імені додано, щоб браузер не використовував попередню версію бандлу.

Розмір файлу з кодом можливо зменшити шляхом застосування мініфікації та агліфікації коду.

Мінімізація — це процес видалення всіх непотрібних символів з вихідного коду інтерпретованих мов програмування або мов розмітки без зміни його функціональності. Ці непотрібні символи зазвичай включають пробіли, символи нового рядка, коментарі, а іноді й роздільники блоків, які використовуються для додання читабельності коду, але не потрібні для його виконання. Мініфікація зменшує розмір вихідного коду, роблячи його передачу через мережу (наприклад, Інтернет) більш ефективною. У культурі програмістів метою розважальних змагань з гольфу є прагнення до надзвичайно мінімізованого вихідного коду [43].

Приклад застосування мініфікації на фрагменті коду з бандлу до мніфікації:

```
const App = () => {
    _s();
    const { accessToken } =
(0,react_redux__WEBPACK_IMPORTED_MODULE_1__.useSelector)(state => state.user);
if(!accessToken){return
/*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_4__.jsxDEV)(react_router_dom__WEBPACK_IMPORTED_MODULE_5__.Navigate, {
```

Приклад коду після мініфікації:

```
constApp=()=>{_s();const{accessToken}=(0,react_redux__WEBPACK_IMPORTED_MODULE_1__.useSelector)(state=>state.user);if(!accessToken){return(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_4__.jsxDEV)(react_router_dom__WEBPACK_IMPORTED_MODULE_5__.Navigate,{
```

Також, одним з підходів до оптимізації є розділення файлу з кодом на частини та використання відкладеного завантаження.

Відкладене завантаження – це практика затримки завантаження або ініціалізації ресурсів чи об’єктів, поки вони фактично не знадобляться для

підвищення продуктивності та економії системних ресурсів. Наприклад, якщо на веб-сторінці є зображення, яке користувач має прокрутити вниз, щоб побачити його, ви можете відобразити заповнювач і завантажити повне зображення лише тоді, коли користувач прибуде до його розташування [44].

Для профайлінгу main.js був використаний плагін до системи збірки бандлів webpack - webpack-bundle-analyzer [42] Результат профайлінгу не оптимізованого бандлу (рис. 3.13).

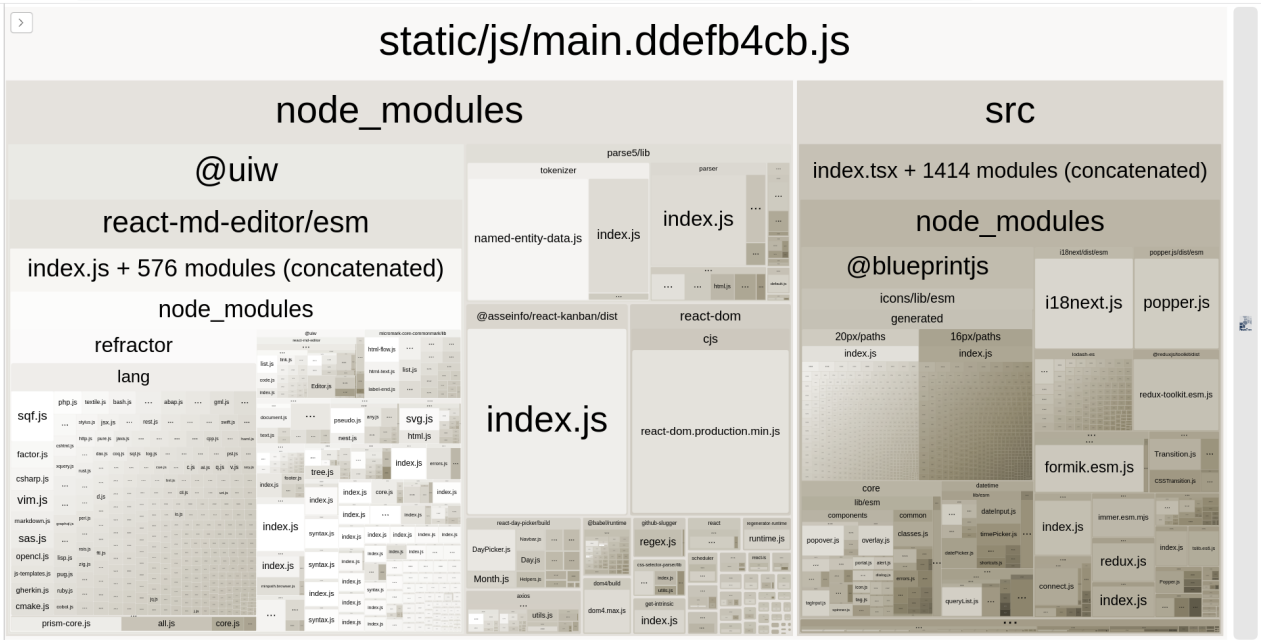


Рисунок 3.13 — Результат профайлінгу не оптимізованого бандлу

Аналізуючи результати профайлінгу зроблено висновок, що третину бандлу займає @uiw/react-md-editor. Дана бібліотека використовується для реалізації зручного редактору редагування опису до завдання в задачі. Отже можна, підвантажувати, дану бібліотеку лише коли користувач перейде в повне представлення завдання.

Бібліотека @uiw/react-md-editor реалізує React компонент MDEditor, який викликається в файлі Task.tsx :

<FormGroup label={t("Опис завдання")}>

<MDEditor value={card.description} onChange={(val: any) => ''}/></FormGroup>

Реалізовуючи відкладене завантаження, було реалізовано компонент обгортку LazyTextEditor, який замінює MDEditor в компоненті Task. Код реалізації компоненту LazyTextEditor:

```
const MDEditor = lazy(() => import('@uiw/react-md-editor'));
interface ITextEditorProps {
   onChange: (value: string | undefined) => void; value?: string,
   hideToolbar?: boolean, preview?: 'live' | 'edit' | 'preview' }
export const LazyTextEditor = (props: ITextEditorProps) => {
   return (<Suspense fallback={<LoadField fill={true} />}>
       <MDEditor {...props}/> </Suspense>)
};
```

Перша стрічка застосування функції lazy яка приймає функцію, що викликає динамічний import(). Та повертає Promise, який розв’язується до модуля з експортом за замовчуванням, що містить компонент React. Наступною стрічкою опис інтерфейсу ITextEditorProps, даний інтерфейс описує аргументи, які приймає компонент LazyTextEditor. Функція *onChange* приймається в якості аргументу, та викликається при зміні тексту в едіторі. Параметр *hideToolbar* приймається, в якості аргументу та задає режим відображення верхньої стрічки компоненту едітора з інструментами редагування тексту. Окремий опис інтерфейсу зандобився, оскільки імпортуючи інтерфейс компонента, до бандлу потрапить вся бібліотека, тому необхідно було описано поля аргументів, які використовуються в проекті. Далі опис компонента LazyTextEditor, всередині якого компонент Suspense - це спеціальний компонент, для відкладеного завантаження, він буде чекати на резолв від промісів компонентів, які знаходяться всередині. Поки, компонент не завантажений, буде відображатися даний компонент буде відображати <LoadField fill={true} />.

Після проведення оптимізації проведено повторний профайлінг бандлу клієнтської частини програмного додатку (рис. 3.14).

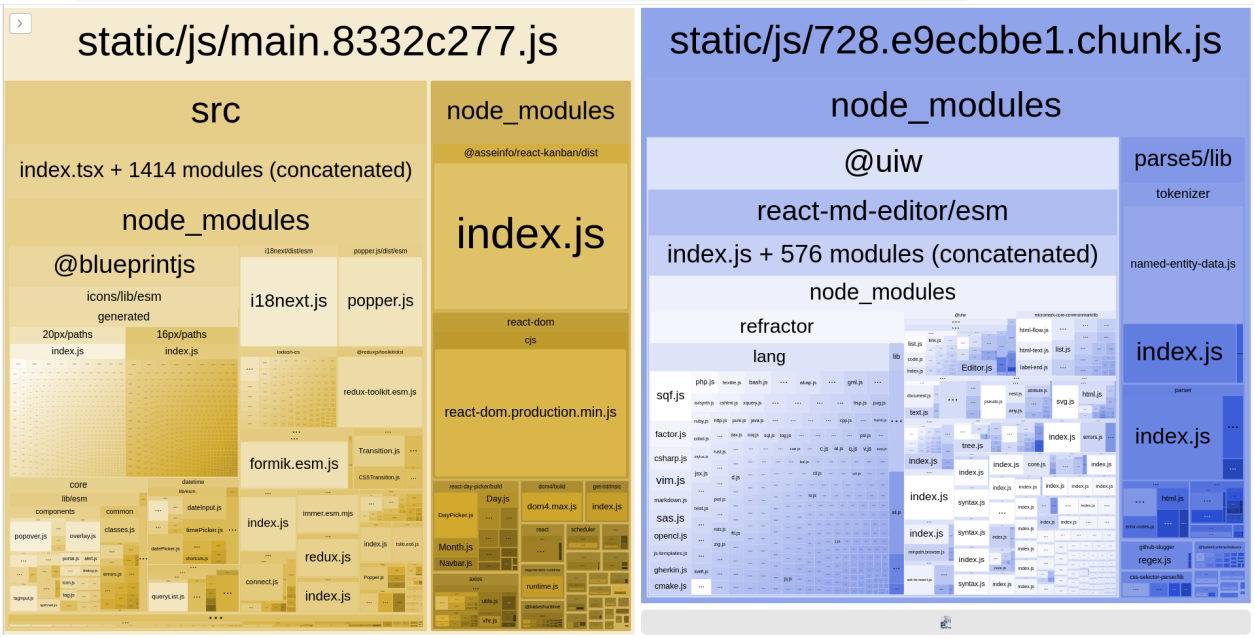


Рисунок 3.14 — Результат профайлінгу оптимізованого бандлу

Профайлінг бандлу демонструє розділення ресурсів майже порівну між main.js який містить основний код, та chunk.js який містить лише редактор тексту. Дана оптимізація повинна суттєво зменшити час завантаження клієнту.

Перевіряючи, як фактично вплинула дана оптимізація на швидкість завантаження клієнтської частини додатку. Було зібрано клієнт та проведено повторний профайлінг за допомогою Chrome DevTools Network (рис. 3.15).

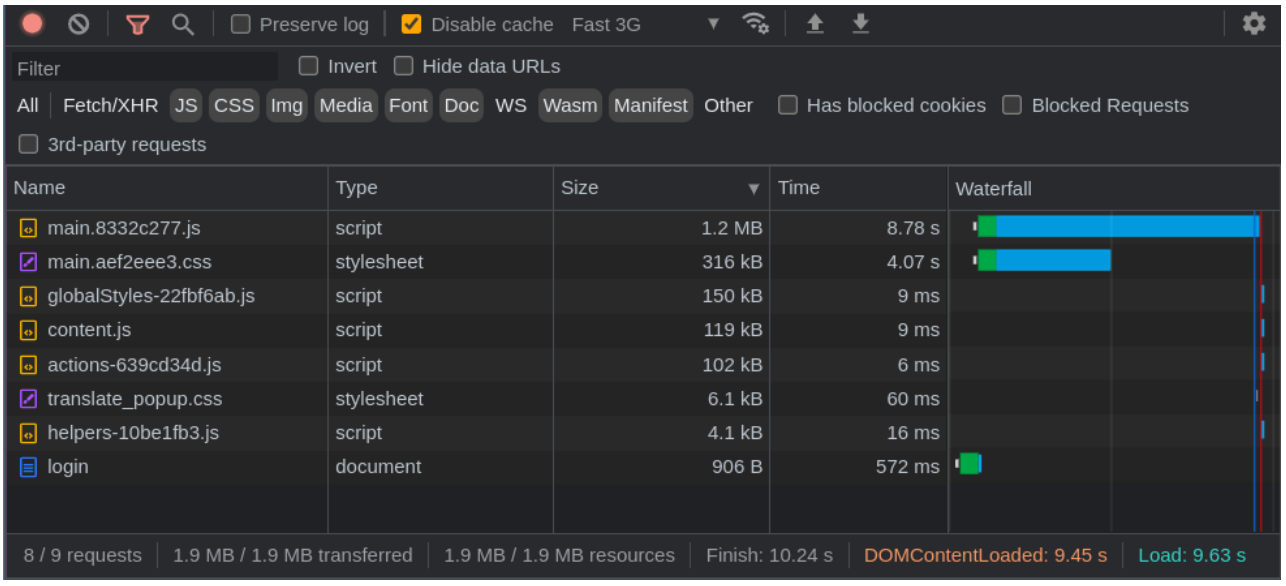


Рисунок 3.15 — Результати профайлінгу оптимізованого додатку

В результаті профайлінгу було отримано, що час завантаження сторінки суттєво зменшився з 16,5 секунд до 10,2 секунд. В процентному співвідношенні клієнтська частина додатку стала завантажуватися на 38% швидше ніж до оптимізації.

3.8 Висновки

У третьому розділі було обґрунтовано вибір клієнт-серверної архітектури програмного додатку, та застосування REST API для комунікації сервера та клієнта. Проведено аналіз засобів для реалізації клієнтської та серверної частини.

Для реалізації клієнтської частини було обрано використання веб платформи, мови програмування TypeScript. Та використання сучасних бібліотек веб розробки React та Redux.

Для реалізації серверної частини було обрано викоритсовувати також мови програмування TypeScript. Дане рішення дало змогу реалізувати безпеку типів між клієнтом та сервером за рахунок винесення інтерфейсів Rest API в окремий файл. Було обрано застосовувати бібліотеку express, для реалізації Rest API. Для надійного збереження та роботи з даними на сервері було обрано базу даних PostgreSQL та ORM бібліотеку sequelize.

Проблему верифікації коритсувачів на сервері було вірешно за допомогою оптимального рішення JWT.

Було проведено оптимізацію клієнтської частини додатку, та зменшено час завантаження на 38%.

4 ТЕСТУВАННЯ ДОДАТКУ

4.1 Автоматичне тестування

Тестування програмного забезпечення – це метод визначення того, чи відповідає фактичний програмний продукт очікуваним вимогам, і гарантує, що програмний продукт не має дефектів. Це передбачає запуск програмних/системних компонентів за допомогою ручних, або автоматизованих інструментів для оцінки однієї або кількох властивостей, що цікавлять [18].

Існують багато методів тестування програмного забезпечення. Одним з розповсюджених методів є написання автоматичних тестів, які перевіряють правильність роботи програми.

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми [45].

В даній роботі, для тестування програмного додатку було обрано застосовувати фреймворк для тестування Jest.

Jest — це платформа для тестування JavaScript, розроблена для забезпечення коректності будь-якої кодової бази JavaScript. Він дозволяє писати тести з доступним, знайомим і багатим на функції API, який швидко дає результати [46].

В процесі розробки програмного додатку було реалізовано ефективний алгоритм зміни послідовності елементів. Були написані тести для перевірки коректності роботи реалізації даного алгоритму в файлі reorder-elements.spec.ts приклад коду:

```
import { reorderObjs } from '@root/utils/reorder-elements';
describe('Test reorder algorithm', () => {
        it('test case 1 inverse start order', () => {
                const ordList = [1, 2, 3, 4, 5];
                const objs = [{ id: 1, ord: 5 },{ id: 2, ord: 4 },{ id: 3, ord: 3 },{ id: 4,
ord: 2 },{ id: 5, ord: 1 }]
```

Першою стрічкою імпортується файл, з реалізацією алгоритму. Наступна стрічка команда *describe('Test reorder algorithm',* декларує набір тестів. Далі другим аргмунетом функція *describe* приймає функцію в якій описані окремі тест кейси. Стрічка *it('test case 1 inverse start order',* декларує конкретний текст кейс. В даному прикладі - це тест який перевіряє роботу алгоритму при повністю інвертованій послідовності у початковому масиві. Далі відбувається ініціалізація змінних з тестовими вхідними даними. *const ordList = [1, 2, 3, 4, 5]* - масив послідовності *id* об'єктів, який повинен бути після виконання, алгоритму. *const objs = [{ id: 1, or…* - початковий масив об'єктів послідовність. Далі відбувається виконання, функції *reorderObjs*, та перевірка результату виконання. Приклад коду:

```
reorderObjs(ordList, objs);
const expectedObj = [ { id: 1, ord: 1 }, { id: 2, ord: 2 }, { id: 3, ord: 3 },{ id: 4, ord: 4 },{ id: 5, ord: 5 }];
expectedObj.forEach(obj => expect(objs).toContainEqual(obj));});
```

Перша стрічка виклик функції *reorderObjs(ordList, objs)*. Та на наступній стрічці об'ява змінної з очікуваним масивом, який повинен бути після коректного виконання функції. Останньою стрічкою тест кейсу *expectedObj.forEach(o…* відбувається перевірка чи відповідає очікуваний об'єкт тому, що отримали після виконання функції. Якщо, так то тест вважається пройденим, в іншому випадку тест є не пройденим, про що буде свідчити лог при виконанні всіх тестів.

Тестуючи, даний модуль було написано 6 тест кейсів.

В даній роботі крім модульного тестування були написані автоматичні тести для інтеграційного тестування. Інтеграційне тування - визначається, як тип тестування, де програмні модулі логічно інтегровані та тестуються як група. Типовий програмний проект складається з кількох програмних модулів, закодованих різними програмістами. Метою цього рівня тестування є виявлення дефектів у взаємодії між цими програмними модулями, коли вони інтегровані [47].

Оскільки в програмному додатку присутній сервер, який надає певне REST API для взаємодії, необхідно написати тести, які перевіряють коректність роботи кожного запиту. Також це важливо, оскільки в майбутньому можуть бути створені нові клієнти, для взаємодії з системою. У випадку коли REST API працює не за специфікацію, після виявлення і виправлення помилки в роботі REST API, необхідно буде виправляти всі наявні клієнти.

Реалізуючи інтеграційні тести необхідно запускати сервер в ізольованому оточенні, таким чином, що б попередній запуск тестів не впливав на наступний. При реалізації даної вимоги виникла проблема збереження запитів з попереднього запуску тестів. Одним з рішень даної проблеми, є очищення бази даних перед запуском тестів. Видаляти дані з продакшн бази заради тестів є небезпечним, тому для тестування буде використовуватися інша база даних. Було прийнято рішення скористатися перевагою ORM бібліотеки seqlizer, що дана бібліотека сумісна з великою кількістю баз даних. Тому обрано в якості тестової бази даних використовувати SQLite. Приклад коду ініціалізації seqlizer файл db.ts:

```
import config from '@root/config';
const sequelize =config.SEQUELIZE.dialect
      ? new Sequelize(config.SEQUELIZE.dialect, config.SEQUELIZE.config)
      : new Sequelize(config.SEQUELIZE.config);
export default sequelize;
```

В даному прикладі імпортується конфіг для ініціалізації, та відбувається створення нового екземпляру класу *Sequelize.* Опис кофінгу для ініціалізації файл config.ts:

```
switch (env) {
    case 'test':  return { dialect: 'sqlite::memory',
     config: {...ormConfig,  storage: process.env.TEST_DB_STORAGE}
    default:
```

В прикладі змінна env містить значення стандартної змінної оточення node.js *process.env.NODE_ENV.* В залежності від значення цієї змінної змінюється конфіг за яким ініціалізується бібліотека *sequelize.* В прикладі *dialect* вказаний, як sqlite::memory - це одна з функцій sequelize, яка дозволяє працювати з SQLite базою даних в оперативній пам'яті, без створення файлу бази даних, що пришвидшує проходження тестів.

Приклад реалізації одного з тест кейсів REST API файл index.spec.ts:

```
it('Create user', async () => {
    const userData = { ...MockUserData };
    const createResult = await request(app)
      .post('/api/auth/signup').send(userData)
     .expect(200);
    const id = createResult.body.user.id;
    const userResult = await User.findOne({ where: { id } });
    expect(userResult).toMatchObject({ ...MockUserData });
  });
```

Даний приклад перевіряє коректність роботи запиту на створення нового користувача “/api/auth/signup”. Змінна “*MockUserData*” містить дані тестового користувача. Функція “*request(app)*” приймає в якості аргументу екземпляр класу express який реалізує REST API. Дана функція повертає об’єкт зі зручними методами для виклику необхідних запитів. В наступних двох стрічках відбувається відправка запиту з даними. Стрічка “*.expect(200)”* вказує, що даний запит очікувано повинен виконатися з кодом 200. В наступних двох стрічках виконується запит в базу даних, для отримання даних про користувача якого, перевіряємий запит повинен був створити. Та останньою стрічкою перевіряється коректність, даних користувача.

У ході виконання дипломної роботі було реалізовано 4 набори тестів, та 21 тест кейс. Всі тести проходять успішно. Результат виконання тестів (рис. 4.1).

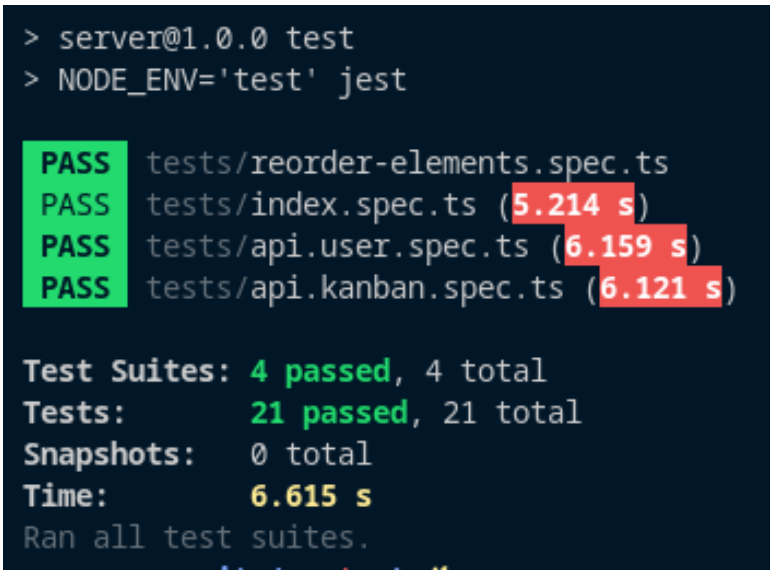


Рисунок 4.1 — Скріншот результату виконання тестів

4.2 Тестування модуля авторизації

Функціональне тестування – один із видів тестування, спрямованого на перевірку відповідності функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, володіє усім необхідним замовнику функціоналом [18].

Програмним додатком буде користуватися велика кількість людей, перше з чим буде взаємодіяти користувач, це модуль авторизації. Тому перший модуль функціональність, якого буде перевірено це буде модуль авторизації.

Перевірку, функціоналу клієнтської частини було проведено вручну. При кожному тесті перевірялась робота клієнту, на кожному з популярних на сьогодні браузерів Chrome, Firefox та Safari.

Не авторизований користувач переходячи, за посиланням повинен в будь якому випадку потрапляти на екран авторизації користувача, в незалежності від, того який шлях в URL вказаний.

Було перевірено перехід не авторизованим користувачем за посиланнями які з такими шляхи в URL “/”, “/main”, “/kanban/1”, “/registration”, “/logout”. У всіх випадках клієнт відпрацьовував коректно шлях URL змінювався на “/login” та відображався екран авторизації (рис. 4.2).

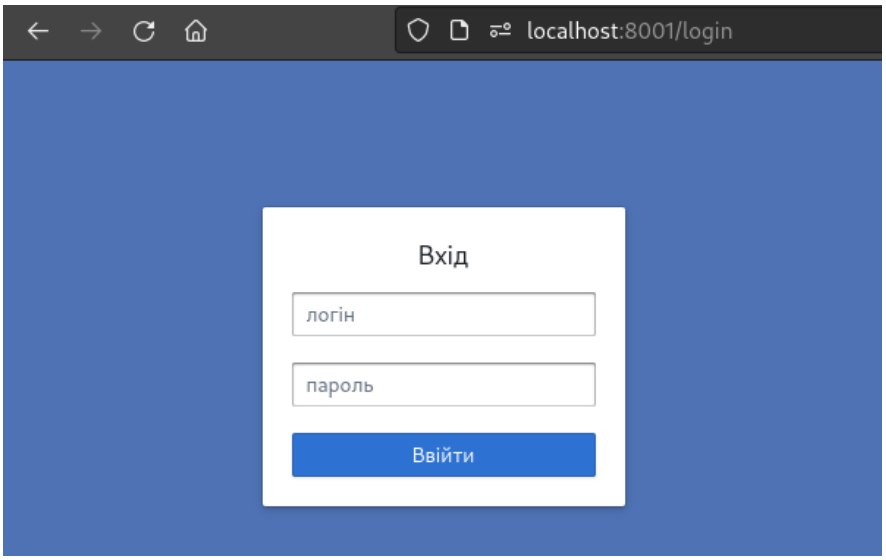


Рисунок 4.2 — Екран авторизації

Перевірка функціоналу обробки некоректних даних авторизації. Було виконано декілька спроб авторизуватися з різними варіантами невалідних даних. В усіх випадках клієнт відпрацьовував згідно вимогам, а саме відображав повідомлення про помилку (рис. 4.3).

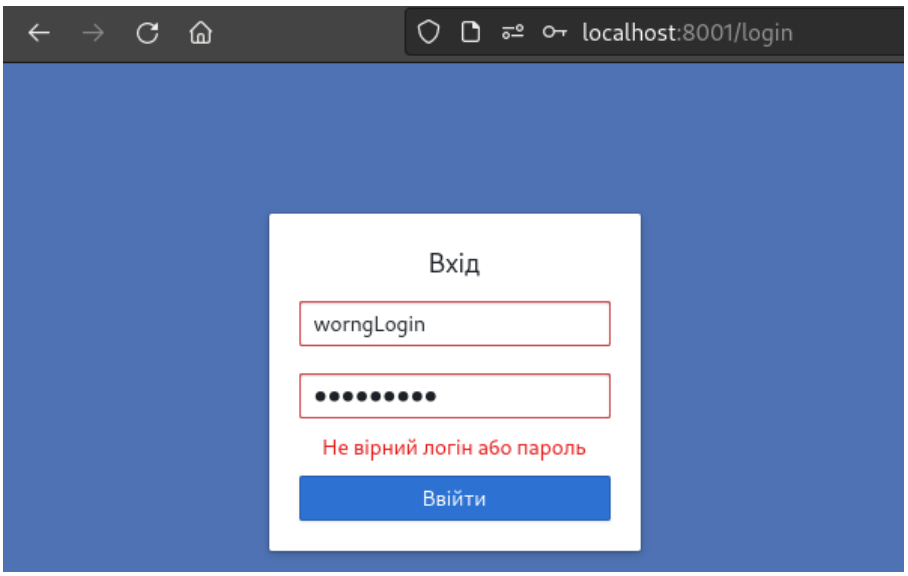


Рисунок 4.3 — Екран авторизації помилка

Перевірка функціоналу авторизації з валідними даними для входу в систему. Було виконано, авторизації з даними декількох різних користувачів. В усіх випадках клієнт відпрацьовував коректно, після авторизації клієнт відображав

головний екран, з списком дошок які доступні даному авторизованому користувачу (рис. 4.4).

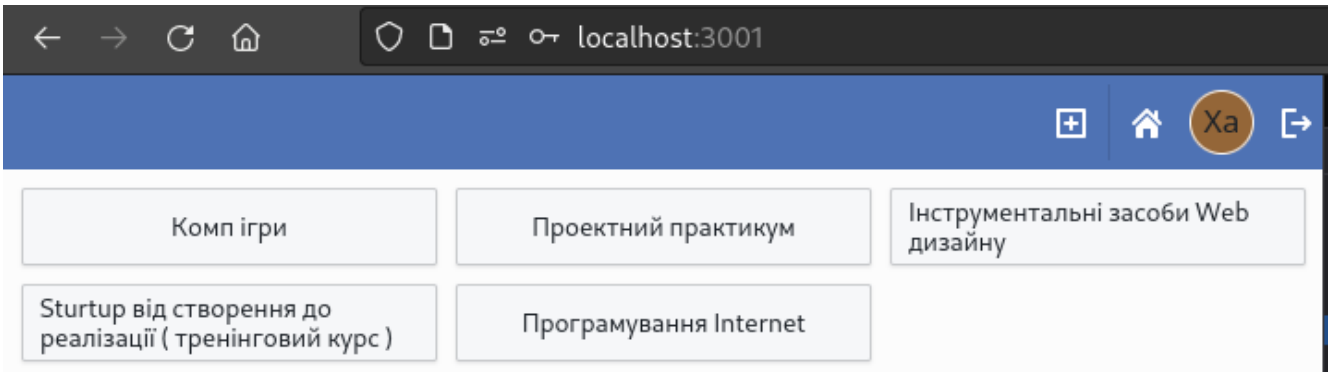


Рисунок 4.4 — Головний екран

4.3 Тестування модуля Kanban Board

Наступним модулем для тестування було обрано Kanban Board, адже це модуль навколо, якого побудована вся система і його коректна функціональність є критичною, для програмного додатку. Створивши нову дошку ми потрапляємо на пустий екран дошки (рис. 4.5).

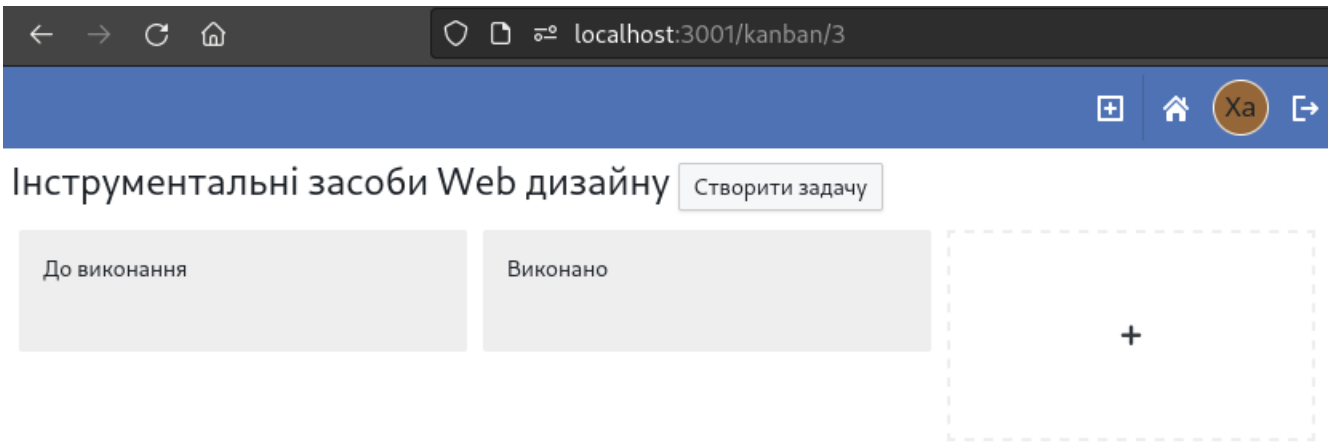


Рисунок 4.5 — Нова канбан дошка

Тепер необхідно налаштувати етапи на новій дошці, для цього треба натиснути на кнопку «+».

Після чого з’явиться форма додавання нового етапу (рис. 4.6).

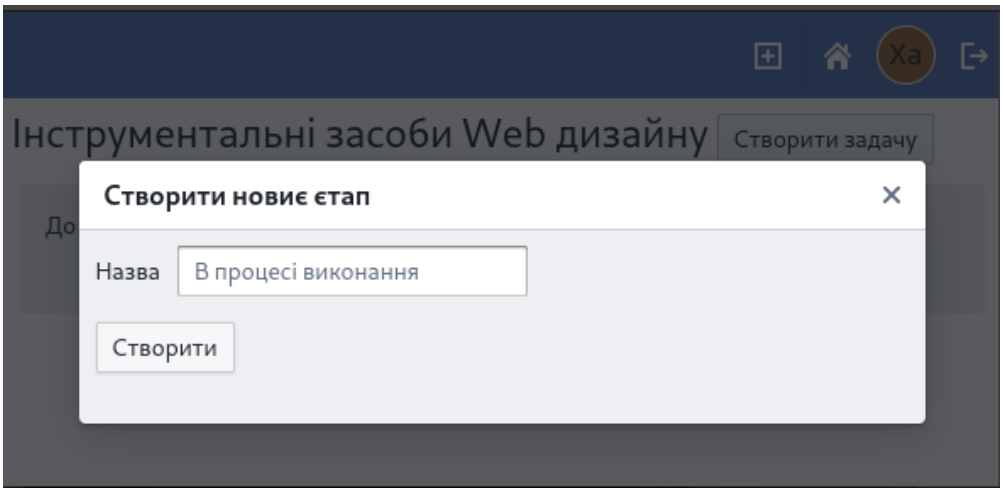


Рисунок 4.6 — Форма додавання нового етапу

Після введення назви нового етапу та натискання на кнопку «Створити» буде додано до дошки новий етап (рис. 4.7).

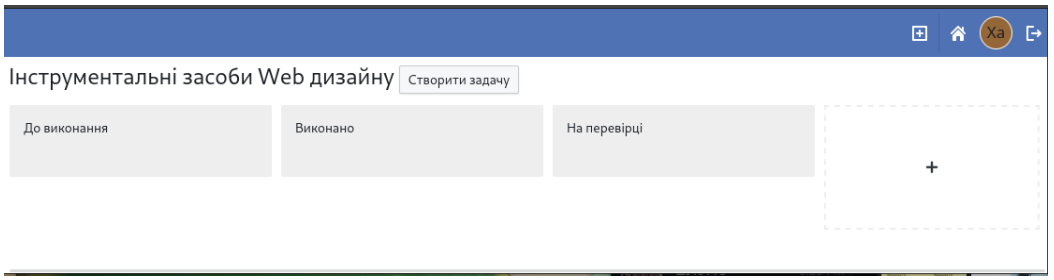


Рисунок 4.7 — Результат додавання нового етапу

Новий етап додався в кінці всіх списку. Було проведено перевірку можливостей зміни послідовності етапів на дошці. Зміна послідовності відбувається за допомогою дії тягни та кидай (англ. Drag and drop). Необхідно натиснути на необхідний етап та перетягнути його в те місце в послідовності етапів де він повинен бути (рис 4.8).

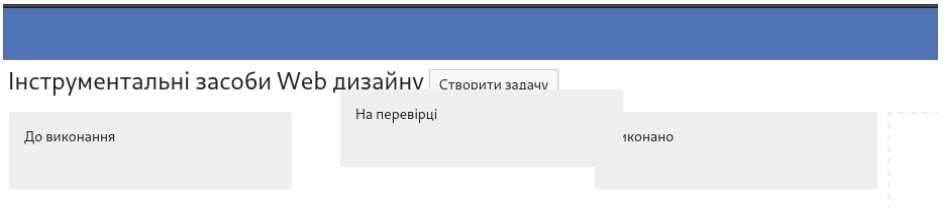


Рисунок 4.8 — Зміна послідовності етапів

Другою частиною тестування Kanban Board є перевірка функціональності пов’язаної з картками.

Протестуємо можливість створення карток на дошці, для цього необхідно натиснути на кнопку «Створити задачу». Також було перевірено, дана кнопка відображається лише користувачам з ролями «admin» та «teacher». Після натискання з’явиться форма для створення картки (рис. 4.9).

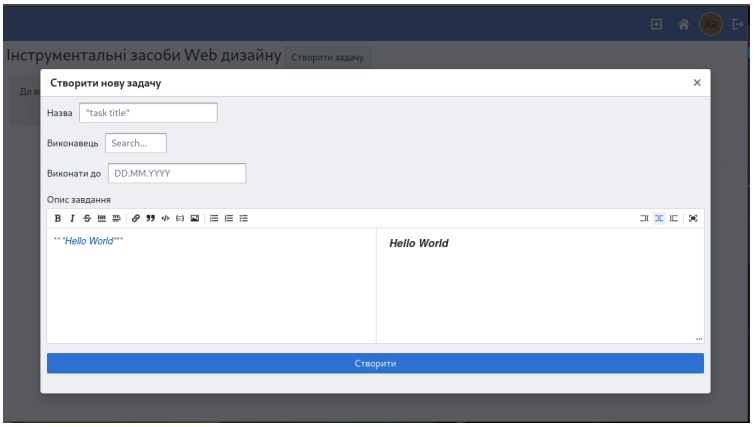


Рисунок 4.9 — Форма створення картки

Після відкриття форми необхідно заповнити поля. В полі виконавець можливо вибрати необхідних користувачів (рис. 4.10).

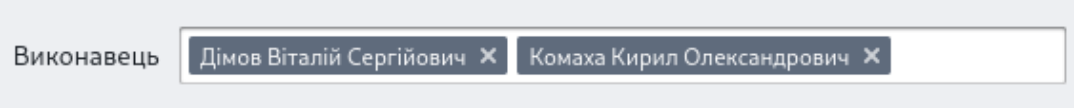


Рисунок 4.10 — Заповнене поле «Виконавець» у формі створення задачі

При виборі декількох користувачів, у полі «Виконавець» буде створена копія задачі для кожного з них. Поле «Опис завдання» на дві частини, перша для розмітки тексту в форматі Markdown, друга для попереднього перегляду результату. Для більш зручного редагування додані функціональні кнопки, які дозволяють швидко стилізувати текст, та додавати списки, посилання. Також існує можливість розгорнути редактор на весь екран, для більш зручного редагування (рис. 4.11).

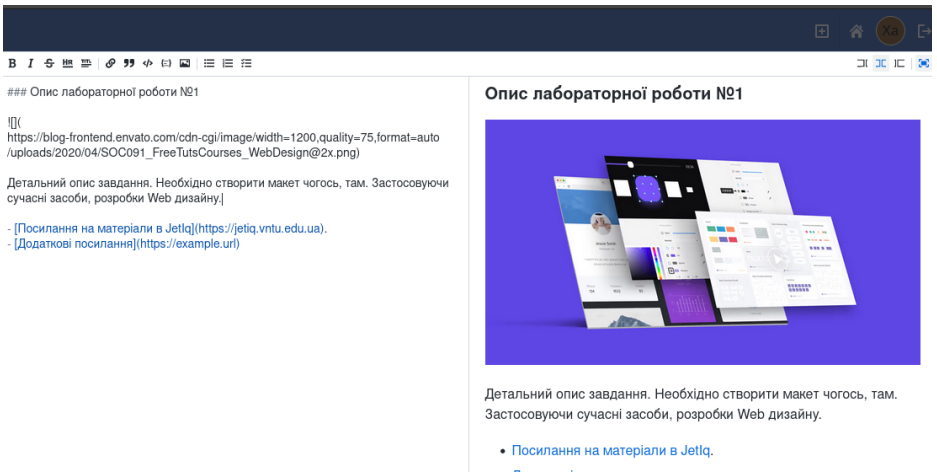


Рисунок 4.11 — Заповнене поле «Опис завдання» у формі створення задачі

Було перевірено функціонал ректору тексту. Можливість додавати зображення, та посилання. Після натискання на кнопку «Створити» форма закривається, та повертає користувача на оновлений Kanban Board, на якому відображаються дві щойно створені картки задач (рис. 4.12).

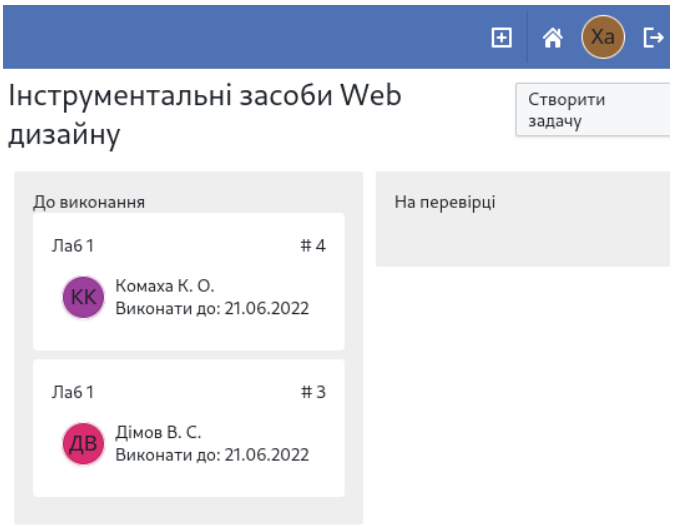


Рисунок 4.12 — Скріншот результату створення завдання

Далі необхідно протетсувати можливість пересувати картки з одного етапу на інший для цього створимо деяку кількість карток на дошці таким самим шляхом, як було описано вище.

В результаті додавання карток на дошку. Всі нові картки потрапляють на етап «До виконання». Дошка має початковий вигляд (рис. 4.13).

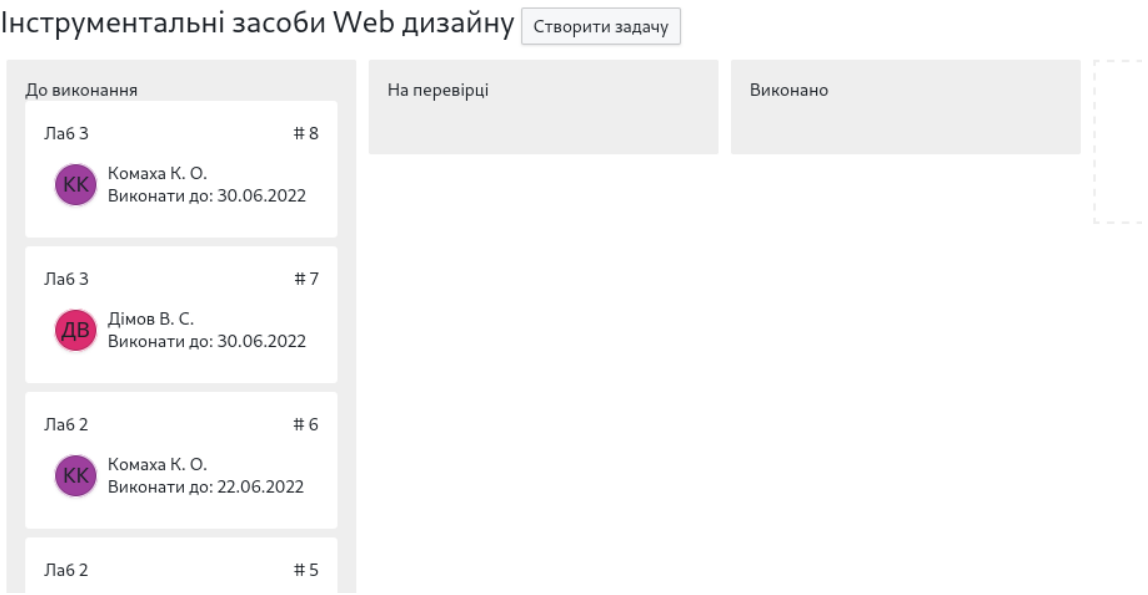


Рисунок 4.13 — Скріншот дошка з доданими тестовими картками

Для переміщення картки з етапу на етап використовується аналогічна дія, як і для зміни послідовності етапів тягни та кидай. Треба натиснути на картку та не відпускаючи перетягнути її до необхідного етапу (рис. 4.14).

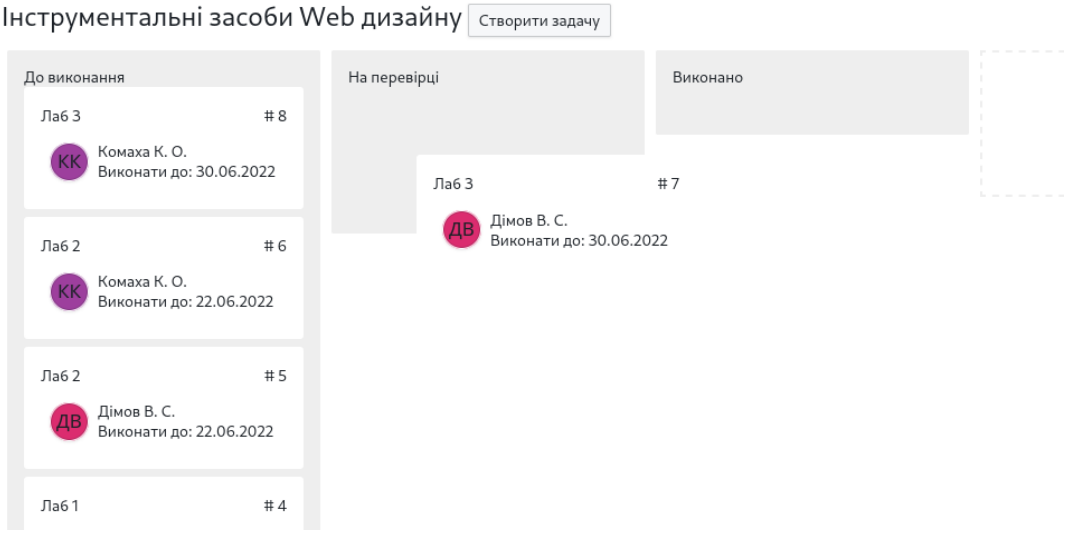


Рисунок 4.14 — Скріншот перетягування картки на інший етап

Після того як, картка опинилися над необхідним етапом, це також буде відображено завдяки пустому місцю, яке відображає майбутнє розташування кардки. Необхідно відпустити праву клавішу миші, результат картка перенесена на необхідний етап (рис. 4.15).

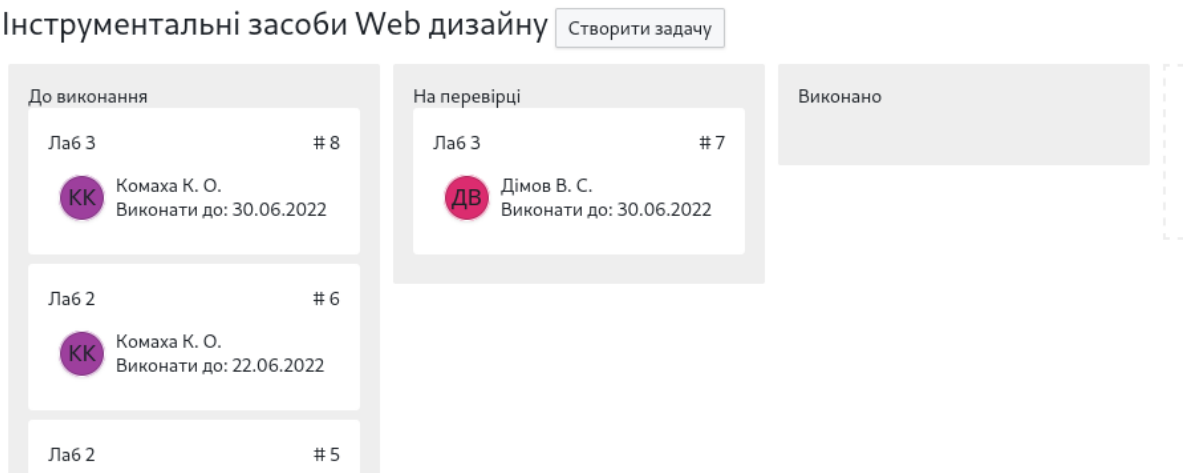


Рисунок 4.15 — Скріншот результату перетягування картки на інший етап

Було перевірено функціонал Kanban Board, а саме можливість редагувати етапи, створювати нові завдання, переміщати картки, та передивлятися повний зміст завдання. У ході тестуванн не виявлено некоректної роботи модуля Kanban Board.

4.4 Створення інструкції для запуску програмного додатку

Програмний додаток, який розроблений в ході даної дипломної роботи, містить багато компонентів сервер, базу даних, клієнтську частини. Кожен з цих компонентів повинен бути налаштований, коректно. Тому необхідно детально описати процес запуску програми. Інструкція буде лежати в корні проекту в файлі «README.md».

Для роботи програмного додатку, необхідні встановлений на комп’ютерів наступні додатки Node.js, NPM, Yarn, Docker, docker-compose. Також для користувачів Windows необхідно встановити bash емулятор, та запускати скрипти за допомогою саме нього.

Було прийнято рішення описати два можливих варіанта запуску проекту, в режимі розробника та в режимі використання.

Для запуску проекту в режимі використання необхідні встановити необхідні програмні додатки, що описані вище. Перейти в корінь директорії з проектом та виконати команду «./start.sh» (рис. 4.16).

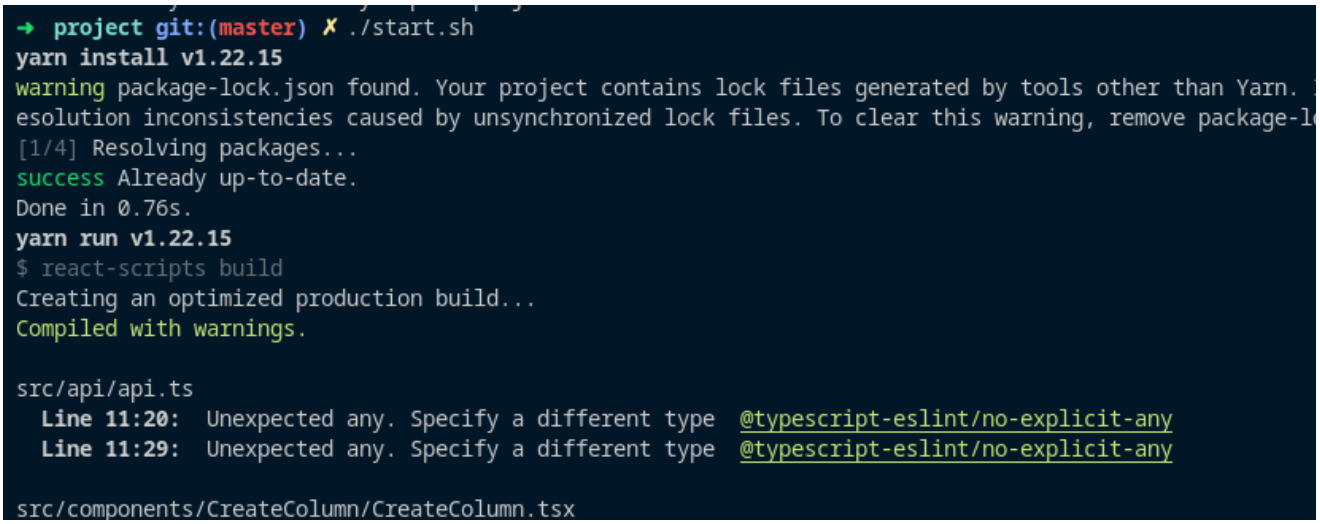


Рисунок 4.16 — Скріншот результату виконання команди «./start.sh»

Необхідно почекати деякий час, поки завантажуються Docker образи та відбудеться збірка клієнту та серверу. Програмний додаток буде запущений з параметрами по замовчуванню, дані параметри можна переглянути в файлі «.example.env». Після того, як ініціалізація завершиться в терміналі користувач буде виведено «[server]: Server is running at https://localhost:3000» (рис. 4.17).

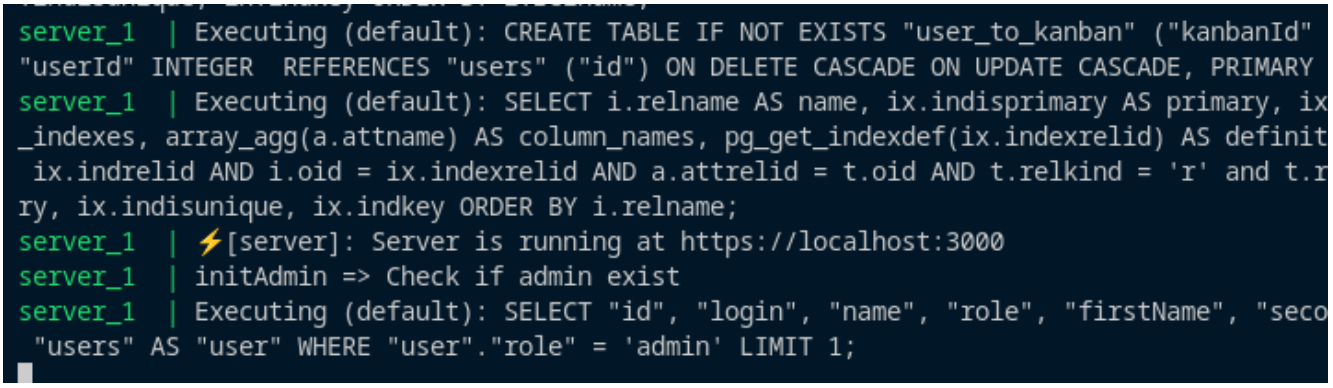


Рисунок 4.17 — Скріншот результату звершення ініціалізації додатку

За замовчуванням після ініціалізації програмний додаток буде доступний за посиланням «http://localhost:8001». Та автоматично створений користувач з ролью «admin». Логін «admin», пароль «admin».

Для запуску програмного додатку в режимі розробника необхідно виконати наступні кроки. Встановити необхідні програмні додатки, які описувалися вище.

Перше необхідно підняти базу даних, для цього необхідно виконати наступні кроки.

1. Перейти в папку ./server

2. Виконати команду - npm run start:db

3. Дочекатися розгортання docker контейнеру

Перевірити, стан контейнера можна виконавши команду - docker-compose ps База даних піднята та працює. Тепер необхідно запустити сервер. Для цього необхідно перейти в папку ./server. Виконати команду «yarn install», дочекатися встановлення всіх необхідних пакетів. В папці серверу створити файл «.env», та заповнити його по аналогії з файлом «.example.env». Даний файл містить параметри, які будуть передані серверу при запуску.

Повний опис всіх параметрів які може приймати сервер (рис. 4.18).

Опис аргументів файлу .env

Назва поля	Опис	Значення з .example.env
POSTGRES_USER	Логін бази даних	postgres
POSTGRES_USER	Пароль бази даних	postgres
POSTGRES_DB_NAME	Назва таблиці в базі даних	fkan
POSTGRES_HOST	Посилання на Posgres з контейнера	db
POSTGRES_PORT	Порт для Postgres	5432
ADMIN_LOGIN	Логін адміністратора	admin
ADMIN_PASSWORD	Пароль адміністратора	admin
SERVER_PORT	Порт серверу	3000
SERVER_SECRET	Секркетний ключ для генерації JWT	example_string

Рисунок 4.18 — Опис аргументів файлу .env для серверу

Після налаштування параметрів. Необхідно виконати команду «npm start», та дочекатися коли в консолі з'явиться текст «[server]: Server is running at https://localhost:3000». Порт буде змінюватися в залежності від того який вказаний в файлі «.env». Тепер сервер працює. При необхідності вносити зміні в код серверу необхідно завершити поточний процес, та запустити його знову. Тільки після перезапуску код сервера оновиться.

Наступним кроком необхідно запустити клієнт. Для цього потрібно виконати наступні кроки: перейти в папку ./client/, виконати команду - yarn install, створити

файл .env файл повинен знаходитися за шляхом ./client/.env, Заповнити файл .env приклад заповнення .env файлу знаходится за шляхом ./client/.example.env, виконати команду - npm run start

Опис доступних параметрів файлу «.env» для клієнту (рис. 4.19).

Назва поля	Опис	Значення з .example.env
CLIENT_PORT	Порт клієнту	8001
REACT_APP_API_URL	URL серверу за яким звертається клієент	'http://localhost:3000'

Рисунок 4.19 — Опис аргументів файлу .env для клієнту

Внесення змін в код клієнту відбувається на льоту. Вносяться зміни, сторінка автоматично перезавантажується та відображається з урахуванням оновленого коду.

4.5 Висновки

В даному розділі було проведено повне тестування програмного додатку, для виявлення некоректної поведінки програми.

Тестування серверної частини додатку відбулося за рахунок написання автоматизованих тестів, які перевіряли коректність роботи REST API. Для написання тестів використовувався фреймворк «Jest». Було створено 4 набори тестів, та 21 тест кейс. Усі автоматизовані тести проходять успішно.

Тестуванн клієнтської частини додатку відбувалося мануально. Перевірено коректність всіх модулів, та було приділено особливу увагу під час тестування модуля авторизації та Kanban Board. В ході тестування критичних помилок в роботі не виявлено.

Також було створено інструкцію по запуску програмного додатку у режимі для використання, та в режимі для розробника. Копія інструкції, була збережена в корні проекту в файлі «README.md» та містить всю необхідну інформації для роботи з програмним додатком.

ВИСНОВКИ

В першому розділі було виконано аналіз стану проблеми. Було обґрунтовано чому використання Kanban Board може покращити навчальний процес. Оскільки маючи, Kanban Board з завданнями, та наочно відслідковуючий на якій стадії знаходиться кожне з завдань, значно спрощується їх менеджмент. Було проведено порівняльний аналіз аналогів та постановку задачі.

В другому розділі були детально описані вимоги до програмного додатку. Створені макети графічного інтерфейсу, що дозволили виправити помилки дизайну інтерфейсу, ще до реалізації в коді. Було розроблено ефективний алгоритм зміни послідовності об'єктів оскільки в системі відбувається безліч подібних операцій. У порівнянні з наївним алгоритмом, алгоритмічна складність якого $O(\ n^2\ )$ ефективний алгоритм працює за $O(n\ *\ log(n)\ )$.

В третьому розділі описаний процес розробки програмного додатку. Проведений аналіз можливих архітектурних рішень, та обране найкраще для данного проекту використання клієнт-серверної архітектури, та REST API для комунікації між клієнтом та сервером.

Також був проведено аналіз можливих техноогій для реалізації клієнту та серверу. Було обрано розробляти клієнтську частину додатку, як веб додаток з використання мови програмування TypeScript та сучасних бібліотек веб розробки таких, як React та Redux. Серверну частину було прийнято рішення реалізовувати також застосовуючи мову прграмування TypeScript. Дане рішення, дало змогу винести типи REST API за межі клієнту та серверу, та забезпечити безпеку на рівні api.

Реалізація модуля, авторизації поступово ускладнювалася черз не досконаліть рішень. Остаточним рішенням було реалізувати стандарт JWT, який дозволяє однозначно ідентифікувати користувача без зайвого навантаження на базу даних.

Після реалізації клієнтської частини проведена оптимізація часу завантаження. Вдалося зменшити розмір початкового банду майже в полову за

допомогою застосування лінивого завантаження. Дана оптимізація зменшила час між переходом користувача по посиланню та початком роботи на 38%.

В четвертому розділі було проведено повне тестування програмного додатку.

Серверна частина програмного додатку була перевірена за рахунок написання автоматичних тестів із застосування «Jest». Автоматичні тести перевіряють коректність роботи окремих модулів, а також в цілому коректність роботи REST API. Було написано 4 набори тестів та 21 тест кейс. Всі тести проходять успішно.

Також було проведено мануальне тестування клієнтської частини програмного додатку. Перевірені основні модулі не помилок в роботі клієнту не виявлено.

Крім тестування в четвертій частині була створена інструкція по запуску програми. Інструкція містить опис двох варіантів запуску в режимі використання, та в режимі розробника. Для спрощення інструкції, було використано засіб контейнеризації Docker, що дозволило окремо не описувати запуск та налаштування бази даних. Інструкція була продубльована в файл README.md та містить всю необхідну інформацію для початку роботи з програмним додатком.

Розроблений програмний додаток може бути впроваджений в реальний навчальний процес.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Означення навчального процесу. [Електронний ресурс]. URL: http://education-ua.org/navchalniy-proces.php
2. Examples of Kanban Boards for Education. [Електронний ресурс]. URL: https://www.meistertask.com/blog/3-examples-of-kanban-boards-for-education-and-how-to-use-them/
3. W. Jackson, What Is Education? – University of Chicago Press, 2015 – 122с.
4. Edgar Faure. Learning to Be: The World of Education Today and Tomorrow 1979. 239с.
5. Jeffrey Liker. Toyota Way(Toyota:14 принципов ведения бизнеса), 2003. 400с.
6. Japan Management Association. Kanban Just-in Time at Toyota, 1986. 212с.
7. Mike Burrows, Kanban from the Inside, 2014. 270с.
8. What Is a Kanban Board and How to Use It?. [Електронний ресурс]. URL: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban-board
9. About Trello [Електронний ресурс]. URL: https://trello.com/en/about
10. What Is Jira?. [Електронний ресурс]. URL: https://www.productplan.com/glossary/jira/
11. Hygger Kanban Board. [Електронний ресурс]. URL: https://hygger.io/guides/agile/kanban/kanban-boards/ 1. Online Project
12. Management Software - Zoho. [Електронний ресурс]. URL: https://www.zoho.com/projects/
13. Online Project Management Software - Zoho. [Електронний ресурс]. URL: https://www.zoho.com/projects/
14. Мартін Фаулер, UML. Короткий посібник по UML, 1997. 305с.
15. Visual Studio Code. [Електронний ресурс]. URL: https://code.visualstudio.com/
16. Документація бібліотеки React. [Електронний ресурс]. URL: https://uk.reactjs.org/

17. Документація бібліотеки Redux. [Електронний ресурс]. URL: https://redux.js.org/
18. Документація Node.js. [Електронний ресурс]. URL: https://nodejs.org/en/about/
19. Гленфорд Майерс, Искусство тестирования программ, 2005. 272с
20. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті /Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх, – Вінниця: ВНТУ, 2006. – 60 с.
21. Книга Алгоритмы Построение и анализ. Клиффорд Штайн, Рональд Л. Ривест, Томас Х. Кормен, Чарльз И. Лейзерсон, 2000р. 1323с.
22. Client Server Architecture. [Електронний ресурс]. URL: https://cio-wiki.org/wiki/Client_Server_Architecture
23. API. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/API
24. Comparing API Architectural Styles. [Електронний ресурс]. URL: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/
25. Peer-to-peer - https://www.wiki.uk-ua.nina.az/Peer-to-peer.html
26. A tour of the C# language. [Електронний ресурс]. URL: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
27. What is Java technology and why do I need it? [Електронний ресурс]. URL:https://www.java.com/en/download/help/whatis_java.html
28. About_JavaScript programming language. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
29. TypeScript. [Електронний ресурс]. URL: https://www.typescriptlang.org/
30. Specification for HTML & CSS. [Електронний ресурс]. URL: https://www.w3.org/standards/webdesign/htmlcss#:~:text=HTML%20(the%20Hypertext%20Markup%20Language,for%20a%20variety%20of%20devices.
31. What And Why React.js. [Електронний ресурс]. URL: https://www.c-sharpcorner.com/article/what-and-why-reactjs/
32. Node.js. [Електронний ресурс]. URL: https://nodejs.org/en/about/
33. Express. [Електронний ресурс]. URL: https://expressjs.com/

34. Git. [Електронний ресурс]. URL: https://git-scm.com/
35. Database. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Database
36. About Object–relational mapping. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping
37. Using HTTP cookies. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
38. JWT. [Електронний ресурс]. URL: https://jwt.io/introduction
39. DBeaver. [Електронний ресурс]. URL: https://dbeaver.io/about
40. Architecture pattern Model-View-Controller. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC
41. Documentation DevTools network profiling. [Електронний ресурс]. URL: https://developer.chrome.com/docs/devtools/network/
42. Documentation webpack-bundle-analyzer. [Електронний ресурс]. URL: https://github.com/webpack-contrib/webpack-bundle-analyzer
43. Documentation extension minification. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Minification_(programming)
44. Article about lazy loading. [Електронний ресурс]. URL: https://www.imperva.com/learn/performance/lazy-loading
45. Модульне тестування. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Unit_testing
46. Documentation test framework Jest. [Електронний ресурс]. URL: https://jestjs.io/
47. Integration Testing. [Електронний ресурс]. URL: https://www.guru99.com/integration-testing.html
48. Демченко В.С. Використання Kanban Board в робочих процесах // Тези доповідей «LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії 2022», Вінниця: Вінницька політехніка, 2022р.

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф.
О. Н. Романюк
“31” березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу
«Програмний засіб щодо впровадження Kanban board до навчального процесу»
студенту Демченку Віталію Сергійовичу
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доц. каф. ПЗ Хошаба О.М.
"31" березня 2022 р.
Виконав: студент гр.ПІ-18б(з)
Демченко В. С.
"31" березня 2022 р.

Вінниця - 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: Програмний засіб щодо впровадження Kanban board до навчального процесу.

Галузь застосування – навчальний процес

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №13 від « 7 » лютого 2022 р.

3. Мета та призначення розробки.

Метою роботи є вирішення проблеми менеджменту великої кількості завдань під час навчального процесу за рахунок впровадження інструменту Kanban board в навчальний процес.

Призначення роботи – розробка програмного додатку релізуючого інструмент Kanban board.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Examples of Kanban Boards for Education. [Електронний ресурс]. URL: https://www.meistertask.com/blog/3-examples-of-kanban-boards-for-education-and-how-to-use-them/
2. Документація бібліотеки React. [Електронний ресурс]. URL: https://uk.reactjs.org/
3. Comparing API Architectural Styles. [Електронний ресурс]. URL: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/
4. Демченко В.С. Використання Kanban Board в робочих процесах // Тези доповідей «LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії 2022», Вінниця: Вінницька політехніка, 2022р.

5. Технічні вимоги

На пристрої клієнту необхідний встановлений один з сучасних браузерів Google Chrome, Edge, Safari, Opera та доступ в інтернет.

Для роботи серверної частини рекомендується використання комп'ютера з процесором на архітектурі x86-64, операційною Linux та встановленими засобами контейнеризация Docker.

6. Конструктивні вимоги.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та зручним для використання.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

a. пояснювальна записка до БДР;
b. технічне завдання;
c. лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обгрунтування актуальності розробки системи та постановка задач	26.03.2022-10.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи системи	11.04.2022-26.04.2022	Вик.

3	Вибір середовища та мови розробки	27.04.2022-4.05.2022	Вик.
4	Розробка програмного продукту	5.05.2022-24.05.2022	Вик.
5	Тестування роботи системи	25.05.2022-27.05.2022	Вик.
6	Оформлення матеріалів до захисту БДР	27.05.2022-10.06.2022	Вик.

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

ДОДАТОК Б

Протокол перевірки проєкту

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний засіб щодо впровадження Kanban board до навчального процесу

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: О.М.Хошаба

Оригінальність	93,9%
Схожість	6,1%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

☒ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☒ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Демченко В. С.

Керівник роботи _____ Хошаба О.М.

ДОДАТОК В

Лістинг програмного додатку

Код файлу ./index.tsx:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './styles/index.css';
import App from './App';
import { store } from './store'
import { Provider } from 'react-redux';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import { LoginPage, Main } from './routers';
import './i18n';
import { FocusStyleManager } from "@blueprintjs/core";
import KanbanBoard from './components/KanbanBoard/KanbanBoard';
import CreateUser from './components/CreateUser/CreateUser';
import CreateKanban from './components/CreateKanban/CreateKanban';
FocusStyleManager.onlyShowFocusOnTabs();
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.Fragment>
    <Provider store={store}>
      <BrowserRouter>
        <Routes>
          <Route path="login" element={<LoginPage />} />
          <Route path="/" element={<App />}>
            <Route path="/" element={<Main />} >
              <Route path="registration" element={<CreateUser/>} />
```

```
                    <Route path="create_kanban" element={<CreateKanban/>} />
                </Route>
                <Route path="/kanban/:id" element={<KanbanBoard />} />
            </Route>
            <Route
                path="*"
                element={
                    <main style={{ padding: "1rem" }}>
                        <p>There's nothing here!</p>
                    </main>
                }
            />
        </Routes>
    </BrowserRouter>
</Provider>
</React.Fragment>
);
```

Код файлу ./api/api.ts:

```
import axios, { AxiosResponse } from "axios";
import { REACT_APP_API_URL } from "../config";
import { store } from "../store";
export type RequestType = 'POST' | 'GET';
axios.defaults.baseURL = REACT_APP_API_URL;
class Api {
        public get<T = any, K = any>(url: string, params?: T): Promise<AxiosResponse<K>> {
    return axios.get(url, { params, headers: this.headers });
}
```

```
    public post<T, K>(url: string, params: T): Promise<AxiosResponse<K>> {
      return axios.post(url, { ...params }, { headers: this.headers });
    }
    private get headers() {
      const token = store.getState().user.accessToken;
      if (!token) {
        throw Error("User do not authorized");
      }
      return {
        "x-access-token": token,
        "content-type": "application/json",
        'Access-Control-Allow-Origin': '*'
      };
    }
  }
  const api = new Api();
  export default api;
```

Код файлу ./api/kanban.ts:

```
  import api from "./api";
  export const create = async (name: string, users: number[]): Promise<KanbanCreateResponse> => {
      const res = await api.post<KanbanCreateRequest, KanbanCreateResponse>('/api/kanban/create', { name, users });
    return res.data;
  }
  export const get = async (id: number): Promise<KanbanCreateResponse> => {
    const res = await api.get<null, KanbanCreateResponse>(`/api/kanban/${id}`);
    return res.data; }
```

```
export const reorder_columns = async (kanbanId: number, col_orders: number[]) => { const res = await api.post<{ col_orders: number[] }, IColumn[]>(`/api/kanban/${kanbanId}/reorder_columns`, { col_orders }); return res.data; }
export const create_task = async (kanbanId: number, data: CreateTaskRequest) => {
const res = await api.post<CreateTaskRequest, CreateTaskResponse>(`/api/kanban/${kanbanId}/create_task`, { ...data }); return res.data; }
export const get_task = async (kanbanId: number, data: GetTaskRequest) => {
const res = await api.get<GetTaskRequest, GetTaskResponse>(`/api/kanban/${kanbanId}/get_task`, { ...data });
return res.data;
}
export const move_task = async (kanbanId: number, data: MoveTaskRequest) => { const res = await api.post<MoveTaskRequest, MoveTaskResponse>(`/api/kanban/${kanbanId}/move_task`, { ...data });
return res.data;
}
export const create_column = async (kanbanId: number, data: CreateColumnRequest) => {
const res = await api.post<CreateColumnRequest, CreateColumnResponse>(`/api/kanban/${kanbanId}/create_column`, { ...data });
return res.data;
}
export const add_message = async (kanbanId: number, data: AddMessageRequest) => {
const res = await api.post<AddMessageRequest, AddMessageResponse>(`/api/kanban/${kanbanId}/add_message`, { ...data });
return res.data; }
```

```
export const get_messages = async (kanbanId: number, data: GetTaskMessagesRequest) => { const res = await api.get<GetTaskMessagesRequest, GetTaskMessagesResponse>(`/api/kanban/${kanbanId}/get_messages`, { ...data });
    return res.data;
}
export default {
    create,
    get,
    reorder_columns,
    create_task,
    move_task,
    create_column,
    get_task,
    add_message,
    get_messages
}
```

Код файлу ./components/CreateTask/CreateTask.tsx:

```
import style from './CreateTask.module.scss';
import { Button, FormGroup, Dialog, DialogProps, InputGroup, Text, Alert, TextArea, Intent, Classes } from "@blueprintjs/core";
import { useTranslation } from 'react-i18next';
import { useState } from 'react';
import { useFormik } from 'formik';
import api from '../../api';
import { useParams } from 'react-router-dom';
import MultiSelectUsers from '../MultiSelectUsers';
import { LazyDateInput } from '../LazyDateInput';
import { LazyTextEditor } from '../LazyTextEditor';
```

```
export interface ICreateTaskFormValues {
    title: string;
    assignedIds: number[],
    description: string;
    deadline: Date,
}
interface ICreateTaskErrors {
    name?: string,
    general?: string,
}
export interface ICreateTaskProps extends DialogProps {
    onCreateComplete: (kanban: KanbanObj) => any,
    userIds?: number[],
}
const CreateTask = (props: ICreateTaskProps) => {
    const [t] = useTranslation();
    const [description, setDescription] = useState<string>("***Hello***");
    const [errors, setErrors] = useState<ICreateTaskErrors>({});
    const params = useParams();
    const formik = useFormik<ICreateTaskFormValues>({
        initialValues: {
            title: '',
            description: '***Hello World***',
            assignedIds: [],
            deadline: new Date(),
        },
        onSubmit: async (values) => {
                    api.kanban.create_task(Number(params.id), { ...values, deadline:
values.deadline.toISOString(), description: formik.values.description })
```

```
                .then(kanban => {
                    props.onCreateComplete(kanban.kanban);
                })
                .catch(err => {
                    console.error("CreateTask => onSubmit", err.response);
                    switch (err.response.status) {
                        default:
                            return setErrors({ ...errors, general: t("Сталася помилка") })
                    }
                });
            },
        });
        return (
            <Dialog {...props} usePortal={false} className={`${style.taskWrapper}
${Classes.OVERLAY_SCROLL_CONTAINER}`} >
                <Alert
                    isOpen={!!errors.general}
                    onClose={() => setErrors({ ...errors, general: undefined })}
                    intent="danger">
                    <p> {t("Сталася помилка")}</p>
                </Alert>
                <div className={style.body}>
                    <FormGroup label={t('Назва')} inline={true} >
                        <InputGroup
                            name="title"
                            placeholder='"task title"'
                            onChange={formik.handleChange}
                            value={formik.values.title}
                            intent={errors.name ? 'danger' : 'none'}
```

```
                    onFocus={() => { errors.name ? setErrors({ ...errors, name: undefined }) : '' }}
                />
                {errors.name ? <Text className={style.errorText}> {errors.name} </Text> : <></>}
            </FormGroup>
            <FormGroup
                label={t("Виконавець")}
                inline={true}
            >
                <MultiSelectUsers
                    userIds={props.userIds}
                    onChange={(items: IUser[]) => formik.values.assignedIds = items.map(item => item.id)}
                />
            </FormGroup>
            <FormGroup
                label={t("Виконати до")}
                inline={true}
            >
                <LazyDateInput onChange={(date: Date) => formik.values.deadline = date} />
            </FormGroup>
            <FormGroup label={t("Опис завдання")}>
                <LazyTextEditor
                    value={formik.values.description}
                    onChange={(description) => formik.setFieldValue('description', description || '')}
                />
            </FormGroup>
```

```
                                    <Button  onClick={formik.submitForm}  fill={true}
intent={Intent.PRIMARY}> {t("Створити")} </Button>
                </div>
            </Dialog>
        );
    }
    export default CreateTask;
```

Код файлу ./components/CreateTask/CreateTask.module.scss:

```
.body{ margin: 10px; }
.errorText{ color: red;}
.taskWrapper{ width: 90%;}
```

Код файлу ./server/index.ts:

```
import app from '@root/app';
import './models';
import db from '@root/db';
import config from '@root/config';
import { initAdmin } from '@dir/src/init-admin';
const init = async () => {
    await db.sync().catch(console.error);
    const port = config.PORT;
    return new Promise((resolve: any) => app.listen(port, () => {
        console.log(`⚡[server]: Server is running at https://localhost:${port}`);
        resolve();
    }));
}
init().then(initAdmin);
```

Код файлу ./config.ts:

```
import dotenv from 'dotenv';
import { SequelizeOptions } from 'sequelize-typescript';
dotenv.config();
const ENV = process.env.NODE_ENV;
const PORT = process.env.SERVER_PORT;
const SECRET = process.env.SERVER_SECRET;
if (!SECRET) {
    throw Error("JWT SECRET Not found! Please add 'SECRET=${example}' to your .env file");
}
function getSequelizeConfig(env: string | undefined): { dialect?: string, config: SequelizeOptions } {
    const ormConfig = {
        timestamps: false
    };
    switch (env) {
        case 'test':
            return {
                dialect: 'sqlite::memory',
                config: {
                    ...ormConfig,
                    storage: process.env.TEST_DB_STORAGE,
                    logging: false} }
        default:
            return {
                config: {
                    ...ormConfig,
                    dialect: 'postgres',
```

```
            host: process.env.POSTGRES_HOST,
            port: Number(process.env.POSTGRES_PORT),
            username: process.env.POSTGRES_USER,
            password: process.env.POSTGRES_PASSWORD,
            database: process.env.POSTGRES_DB_NAME,
        }
    }
  }
}
export const ADMIN_USER = {
  login: process.env.ADMIN_LOGIN as string,
  password: process.env.ADMIN_PASSWORD as string
}
export default {
  ENV,
  PORT,
  SECRET,
  SEQUELIZE: getSequelizeConfig(ENV),
}
```

Код файлу ./server/src/controller/kanban-controller.ts:

```
import sequelize from "@dir/src/db";
import Card from "@dir/src/models/card-model";
import Column from "@dir/src/models/column-model";
import Kanban from "@dir/src/models/kanban-model";
import User from "@dir/src/models/user-model";
import { Op } from 'sequelize';
import { Request, Response } from "express";
import { reorderObjs } from "@dir/src/utils/reorder-elements";
```

```
import { saveInstances } from "@dir/src/utils/save-instances-list";
import Message from "@dir/src/models/message-model";
// Usage example { where: {id: 2}, ...KANBAN_QUERY }
const KANBAN_QUERY = {
  attributes: { exclude: ['createdAt', 'updatedAt'] },
  include: [{
    model: Column,
    include: {
      // @ts-ignore
      model: Card,
      attributes: ['id', 'ord', 'title', 'deadline', 'assignedId', 'creatorId'],
      order: [
        ['ord', 'ASC'],
      ],
    },
    order: [
      ['ord', 'ASC'],
    ]
  }, {
    model: User,
      attributes: ['id', 'name', 'firstName', 'lastName', 'secondName', 'position',
'role', 'login'],
    through: {
      attributes: [],
    },
  }]
};
// @ts-ignore
const getKanban = async (id: number): Promise<KanbanObj> => {
  // @ts-ignore
```

```
        const kanban = await Kanban.findOne({
            where: { id },
            ...KANBAN_QUERY
        });
        if (!kanban) {
            throw { error: `getKanban => Kanban with ${id} not found` }
        }
        // @ts-ignore
        return { ...kanban.get({ plain: true }) }
    }
    export const create = async (req: Request<null, null, KanbanCreateRequest>, res: Response<KanbanCreateResponse | IErrorResponse>) => {
        try {
            const kanbanInstanse = await sequelize.transaction(async (t) => {
                console.log("kanban/create userId: ", req.userId);
                console.log("kanban/create userId: ", req.userId, "check uniq name");
                const existKanban = await Kanban.findOne({
                    where: { name: req.body.name }
                });
                if (existKanban) {
                    return res.status(409).send({ error: "Kanban with same name already exist" });
                }
                console.log("kanban/create userId: ", req.userId, "create kanban");
                    const [ownerUser, users, kanban, columnStart, columnEnd] = await Promise.all([
                        User.findOne({ where: { id: req.userId } }),
                        User.findAll({ where: { id: [...req.body.users, req.userId!] } }),
                        Kanban.create({ name: req.body.name }, { transaction: t }),
                        Column.create({ title: 'До виконання', ord: 1 }, { transaction: t }),
```

```
                Column.create({ title: 'Виконано', ord: 2 }, { transaction: t }),
            ]);
            console.log("kanban/create userId: ", req.userId, "set kanban relations");
            await Promise.all([
                // @ts-ignore
                kanban.setOwner(ownerUser, { transaction: t }),
                // @ts-ignore
                kanban.addColumns([columnStart, columnEnd], { transaction: t }),
                // @ts-ignore
                kanban.addUsers(users, { transaction: t })
            ]);
            return kanban;
        });
        // @ts-ignore
        res.send({ kanban: (await getKanban(kanbanInstanse.id)) });
    } catch (e: any) {
        console.error("kanban/create error: ", e);
        res.status(500).send({ error: e });
    }
}
export const get_kanban = async (req: Request<{ id: number }, null, null, null>,
res: Response<KanbanCreateResponse | IErrorResponse>) => {
    try {
        const data = await getKanban(req.params.id);
        if (!data) {
            return res.status(404).send({ error: "Kanban not found" });
        }
        res.send({ kanban: data });
    } catch (e: any) {
        res.status(500).send({ error: e });
```

```
    }
}
export const reorder_columns = async (
    req: Request<{ id: number }, null, { col_orders: number[] }>,
    res: Response<{ columns: IColumn[] } | IErrorResponse>
) => {
    try {
        const col_orders = req.body.col_orders.map(el => Number(el));
        const query = {
            where: {
                id: { [Op.or]: col_orders }
            }
        };
        const columns = (await req.kanban.getColumns(query) as Column[])
            .sort((col1, col2) => col1.id - col2.id);
        if (col_orders.length !== columns.length) {
            const correctAmount = await req.kanban.countColumns();
                return res.status(409).send({ error: `Incorrect amount of columns ids.
Correct amount is ${correctAmount}` });
        }
        await sequelize.transaction(async (t) => {
            reorderObjs(col_orders, columns);
            await saveInstances(columns, t);
        });
        const resColumns = columns.map(col => col.get())
        // @ts-ignore
        res.send({ columns: resColumns });
    } catch (e: any) {
        res.status(500).send({ error: e });
    }
```

```
}
export const create_task = async (
    req: Request<{ id: number }, null, CreateTaskRequest>,
    res: Response<CreateTaskResponse | IErrorResponse>
) => {
    try {
        const [[firstColumn], user] = await Promise.all([
            req.kanban.getColumns({ where: { ord: 1 } }),
            User.findOne({ where: { id: req.userId } })
        ]);
        const cards = await firstColumn.getCards({
            attributes: ['id', 'ord'],
            order: [
                ['ord', 'ASC'],
            ]
        });
        const cradsOrder = cards.map((card: Card) => card.id);
        await sequelize.transaction(async (t) => {
            const assignedUsers = await User.findAll(
                {
                    where: { id: req.body.assignedIds },
                    transaction: t,
                });
            for (const assigned of assignedUsers) {
                const card = await Card.create({
                    title: req.body.title,
                    ord: 1,
                    description: req.body.description || '',
                    deadline: req.body.deadline,
                }, { transaction: t });
```

```
            await firstColumn.addCard(card, { transaction: t });
            // @ts-ignore
            await card.setCreator(user, { transaction: t });
            // @ts-ignore
            await card.setAssigned(assigned, { transaction: t });
            cards.unshift(card);
            cradsOrder.unshift(card.id);
        }
        reorderObjs(cradsOrder, cards);
        await saveInstances(cards, t);
    });
    res.send({ kanban: (await getKanban(req.params.id)) });
  } catch (e: any) {
    console.error("kanban-controller => create_task error: ", e);
    res.status(500).send({ error: e });
  }
}
const move_task_in_column = async (
  req: Request<{ id: number }, null, MoveTaskRequest>,
  res: Response<MoveTaskResponse | IErrorResponse>
) => {
  await sequelize.transaction(async (t) => {
    const [column] = await req.kanban.getColumns({
      where: { id: req.body.columnStart.id },
      inlude: {
        model: Card,
        attributes: ['id', 'ord'],
      }
    }, { transaction: t }) as Column[];
    // @ts-ignore
```

```
        const cards = await column.getCards({
            attributes: ['id', 'ord'], order: [
                ['id', 'ASC'],
            ]
        }, { transaction: t })
        reorderObjs(req.body.columnStart.cards, cards);
        await saveInstances(cards, t);
    });
}
const move_tasK_between_column = async (
    req: Request<{ id: number }, null, MoveTaskRequest>,
    res: Response<MoveTaskResponse | IErrorResponse>
) => {
    const columns = await req.kanban.getColumns({
        inlude: {
            model: Card,
            attributes: ['id', 'ord'],
        }
    }) as Column[];
    const startCol = columns.find(col => col.id === req.body.columnStart.id)!;
    const endCol = columns.find(col => col.id === req.body.columnEnd.id)!;
    const query = {
        where: { id: { [Op.not]: req.body.cardId } },
        attributes: ['id', 'ord'], order: [
            ['id', 'ASC'],
        ]
    };
    await sequelize.transaction(async (t) => {
        // @ts-ignore
```

```
            const [card] = await startCol.getCards({ where: { id: req.body.cardId } }, {
transaction: t });
            // @ts-ignore
            await endCol.addCard(card, { transaction: t });
            const [startColCards, endColCards] = await Promise.all([
               // @ts-ignore
               startCol.getCards(query, { transaction: t }),
               // @ts-ignore
               endCol.getCards(query, { transaction: t })
            ]);
            endColCards.push(card);
            reorderObjs(req.body.columnStart.cards, startColCards);
            reorderObjs(req.body.columnEnd.cards, endColCards);

            await Promise.all([
               saveInstances(startColCards, t),
               saveInstances(endColCards, t)
            ]);
         });
      }
      export const move_task = async (
         req: Request<{ id: number }, null, MoveTaskRequest>,
         res: Response<MoveTaskResponse | IErrorResponse>
      ) => {
         try {
            if (req.body.columnStart.id === req.body.columnEnd.id) {
               await move_task_in_column(req, res);
            } else {
               await move_tasK_between_column(req, res);
            }
```

```
            res.send({ kanban: (await getKanban(req.params.id)) });
        } catch (e: any) {
            console.error(e);
            res.status(500).send({ error: e });
        }
    }
    export const create_column = async (
        req: Request<{ id: number }, null, CreateColumnRequest>,
        res: Response<CreateColumnResponse | IErrorResponse>) => {
        try {
            await sequelize.transaction(async (t) => {
                const countCol = await req.kanban!.countColumns() as number;
                  const column = await Column.create({ title: req.body.title, ord: countCol
+ 1 }, { transaction: t });
                await req.kanban.addColumn(column, { transaction: t });
            });
            res.send({ kanban: (await getKanban(req.params.id)) });
        } catch (e: any) {
            console.error(e);
            res.status(500).send({ error: e });
        }
    }
    export const get_task = async (
        req: Request<{ id: number }, null, GetTaskRequest>,
        res: Response<GetTaskResponse | IErrorResponse>) => {
        try {
            const card = await Card.findOne({ where: { id: Number(req.query.id) } });
            if (!card) {
                return res.status(404).send({ error: "Card not found!" });
            }
```

```
            res.send({ card: card.get({ plain: true }) });
        } catch (e: any) {
            console.error(e);
            res.status(500).send({ error: e });
        }
    }
    export const add_message = async (
        req: Request<{ id: number }, null, AddMessageRequest>,
        res: Response<AddMessageResponse | IErrorResponse>) => {
        try {
            await sequelize.transaction(async (t) => {
                const [card, message, user] = await Promise.all([
                    Card.findOne({ where: { id: Number(req.body.taskId) }, transaction: t
}),
                    Message.create({ text: req.body.text }, { transaction: t }),
                    User.findOne({ where: { id: req.userId } }),
                ]);
                await Promise.all([
                    // @ts-ignore
                    card.addMessage(message, { transaction: t }),
                    // @ts-ignore
                    message.setCreator(user, { transaction: t })
                ]);
                res.send({ message: message.get({ plain: true }) });
            });
        } catch (e: any) {
            console.error(e);
            res.status(500).send({ error: e });
        }
    }
```

```
export const get_messages = async (
    req: Request<{ taskId: number }, null, GetTaskMessagesRequest>,
    res: Response<GetTaskMessagesResponse | IErrorResponse>) => {
    try {
        const card = await Card.findOne({
            where: {
                id: Number(req.query.taskId)
            }
        });
        // @ts-ignore
        const messages = await card.getMessages();

        res.send({ messages: messages.map((m: any) => m.get({ plain: true })) });
    } catch (e: any) {
        console.error(e);
        res.status(500).send({ error: e });
    }
}
```

Код файлу ./server/src/controller/user-controller.ts:

```
import User from "@models/user-model";
import { Request, Response } from "express";
import jwt from "jsonwebtoken";
import bcrypt from "bcrypt";
import config from "@root/config";
import Kanban from "@dir/src/models/kanban-model";
export const root = (req: Request, res: Response) => {
   res.send({ data: 'User' });
}
```

```
const SECRET = config.SECRET;
const TOKEN_TIME_EXPIRE = 86400 * 7; // Default 7 days
export const createUser = (data: SignupRequest) => {
   const name = data.role === 'admin' && data.lastName === 'admin'
      ? 'admin'
      : `${data.secondName} ${data.firstName} ${data.lastName}`;
   return User.create({
      login: data.login,
      name,
      role: data.role,
      firstName: data.firstName,
      secondName: data.secondName,
      lastName: data.lastName,
      position: data.position,
      // @ts-ignore
      password: bcrypt.hashSync(data.pass, 8)
   });
};
export const userDataFilter = (userData: User): IUser => {
   return {
      id: userData.id,
      login: userData.login,
      name: userData.name,
      role: userData.role,
      firstName: userData.firstName,
      secondName: userData.secondName,
      lastName: userData.lastName,
      position: userData.position,
   }}
```

```
export const login = async (req: Request, res: Response<IErrorResponse | { user: IUser,
accessToken: string }>) => {
  try {
    const user = await User.findOne({
      where: {
        login: req.body.login
      },
      raw: true,
    });
    if (!user) {
      return res.status(404).send({ error: "User Not found." });
    }
    const isValidUser = await bcrypt.compare(
      req.body.password,
      user.password
    );
    if (!isValidUser) {
      return res.status(401).send({
        error: "Invalid Password!"
      });
    }
    const token = jwt.sign({ id: user.id }, SECRET, {
      expiresIn: TOKEN_TIME_EXPIRE
    });
    res.status(200)
      .send({
        user: { ...userDataFilter(user) },
        accessToken: token
      });
  } catch (err: any) {
```

```
        res.status(401).send({ error: err.message });
    }
}
export const logout = async (req: Request, res: Response) => {
    res.send({ data: "logout" });
}
export const signup = async (req: Request<null, null, SignupRequest>, res: Response<IErrorResponse | SignupResponse>) => {
    try {
        const userData = await User.findOne({
            where: { login: req.body.login }
        });
        if (userData) {
            return res.status(409).send({ error: "User already exist" });
        }
        const createdUser = await createUser(req.body);
        res.send({ user: { ...userDataFilter(createdUser) } });
    } catch (err: any) {
        res.status(500).send({ error: err.message });
    }
}
export const find = async (req: Request, res: Response<IErrorResponse | { user: IUser }>) => {
    const findResult = await User.findOne({ where: req.query });
    if (!findResult) {
        return res.status(404).send({ error: "User not found" });
    }
    res.send({ user: { ...userDataFilter(findResult) } });
}
```

```
export const find_all = async (req: Request<null, null, null, UserFindAllRequest>, res: Response<IErrorResponse | UserFindAllResponse>) => {
    const query = req.query.ids
        ? { where: { id: req.query.ids } }
        : {};
    const userList = await User.findAll(query);
    res.send({ users: userList.map(userDataFilter) });
}
export const get_kanbans = async (req: Request<UserGetKanbansRequset>, res: Response<IErrorResponse | UserGetKanbansResponse>) => {
    const kanbans = await Kanban.findAll({
        include: [{
            model: User,
            where: { id: req.userId },
            required: true,
        }], attributes: ['name', 'id', 'ownerId']
    });
    res.send({ kanbans: kanbans.map(kanban => ({
            id: kanban.id,
            name: kanban.name,
            // @ts-ignore
            ownerId: kanban.ownerId
        }))
    })}
```

Код файлу ./server/src/utils/reorder-elements.ts

```
const binarySearch = (arr: { id: number }[], x: number) => {
    let start = 0, end = arr.length - 1;
```

```
    while (start <= end) {
        const mid = Math.floor((start + end) / 2);
        if (arr[mid].id === x) return mid;
        else if (arr[mid].id < x)
            start = mid + 1;
        else
            end = mid - 1;
    }
    return -1;
}
// Note this function affected objects inside source list
export const reorderObjs = (orderList: number[], objList: { id: number, ord: number }[])
=> {
    const sortObjList = objList
        .sort((obj1, obj2) => obj1.id - obj2.id);
    for(let i = 0; i < orderList.length; i++){
        const index = binarySearch(sortObjList, orderList[i]);
        sortObjList[index].ord = i + 1;
    }
    return sortObjList;
}
```

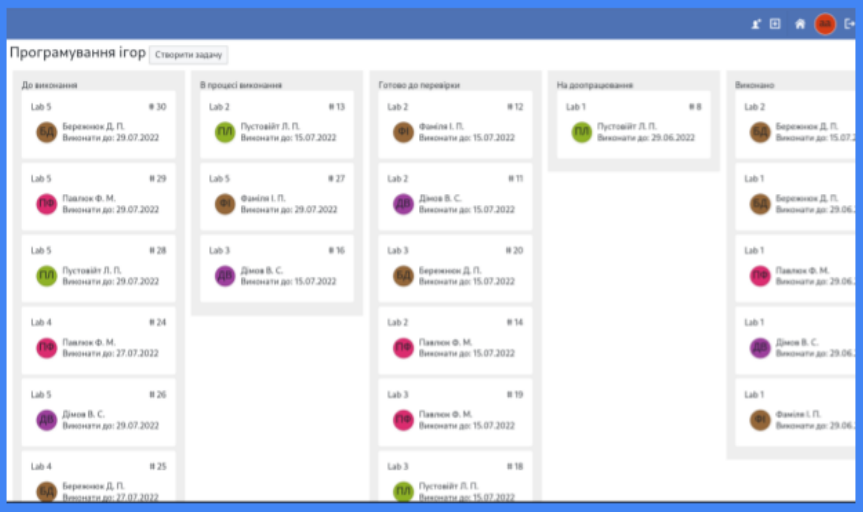
ДОДАТОК Г

Графічна частин**а**

ГРАФІЧНА ЧАСТИНА

ПРОГРАМНИЙ ЗАСІБ ЩОДО ВПРОВАДЖЕННЯ KANBAN BOARD ДО НАВЧАЛЬНОГО ПРОЦЕСУ

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ



ПРОГРАМНИЙ ЗАСІБ ЩОДО ВПРОВАДЖЕННЯ KANBAN BOARD ДО НАВЧАЛЬНОГО ПРОЦЕСУ

Студент гр. ПІ-18б(з)
Демченко В. С.

Науковий керівник к.т.н., доц. каф. ПЗ Хошаба О.М.

ВІННИЦЯ – 2022

Рисунок Г.1 – Слайд презентації 1

МЕТА І ЗАВДАННЯ

Мета і завдання: Метою роботи є підвищення ефективності навчального процесу. За рахунок впровадження програмного додатку на основі інструменту Kanban Board.

Об'єкт дослідження – процес видачі та виконання завдань студентами.

Предмет дослідження – методи та програмні засоби, що дозволяють покращити навчальний процес.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

- методи розробки графічного інтерфейсу;
- методи менеджменту великої кількості завдань;
- методи розробки клієнт серверних систем;
- методи для серіалізації текстових даних;
- методи авторизації користувача;

Рисунок Г.2 – Слайд презентації 2

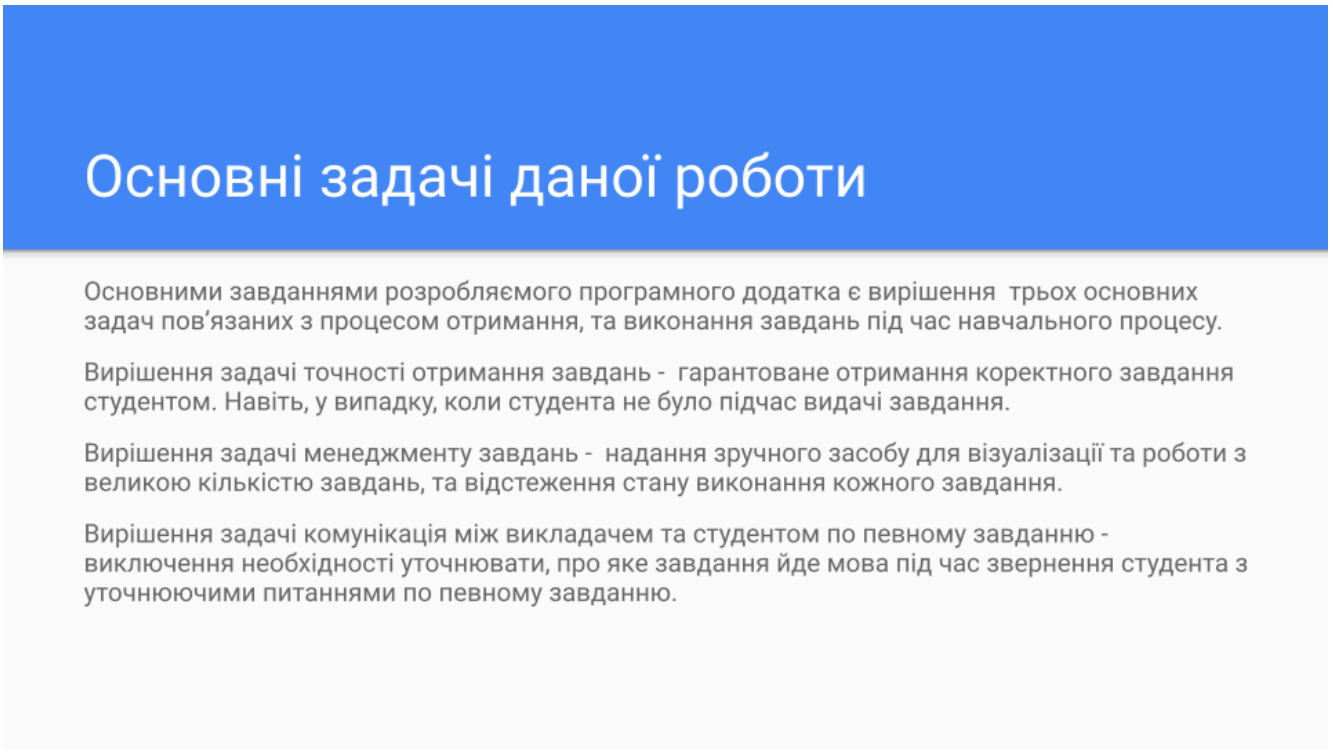


Рисунок Г.3 – Слайд презентації 3

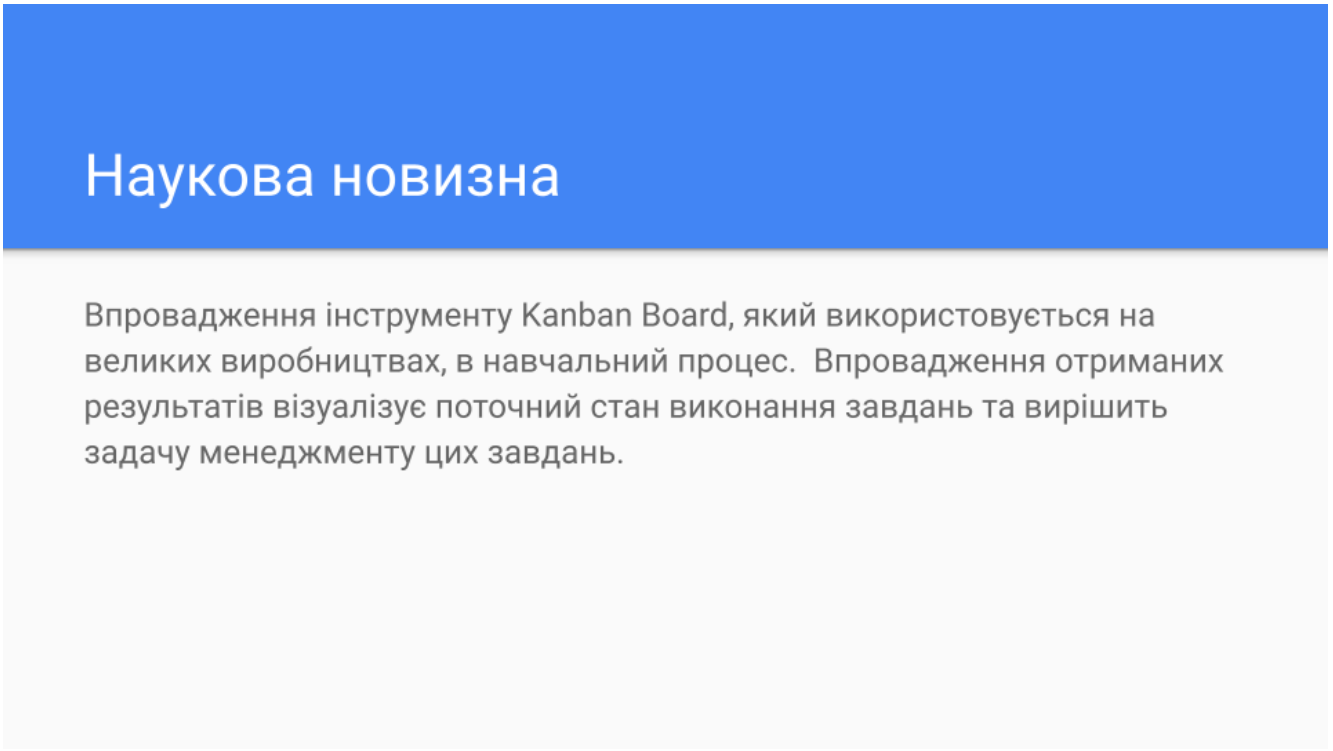
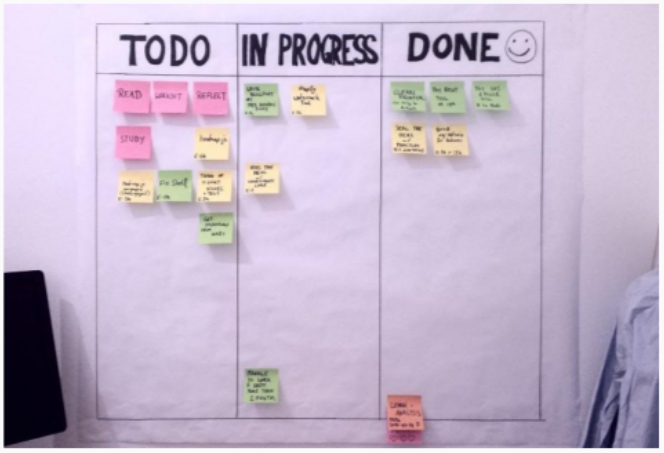


Рисунок Г.4 – Слайд презентації 4

Kanban Board



Kanban Board - це візуальний інструмент, який дає огляд поточного стану роботи.

Містить два основні елементи це стовпчики та картки.

На картках може, бути описана додаткова інформація, яка необхідна для виконання завдання.

Рисунок Г.5 – Слайд презентації 5

Технічні особливості реалізації

Було обрано використовувати клієнт серверну архітектуру програмного додатка та REST API, для комунікації між клієнтом і сервером.

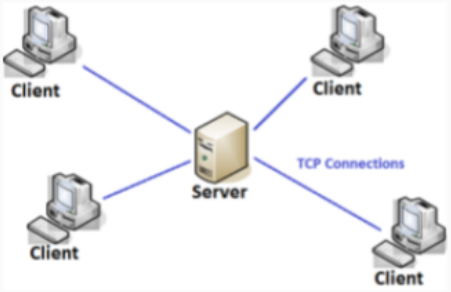


Рисунок Г.6 – Слайд презентації 6

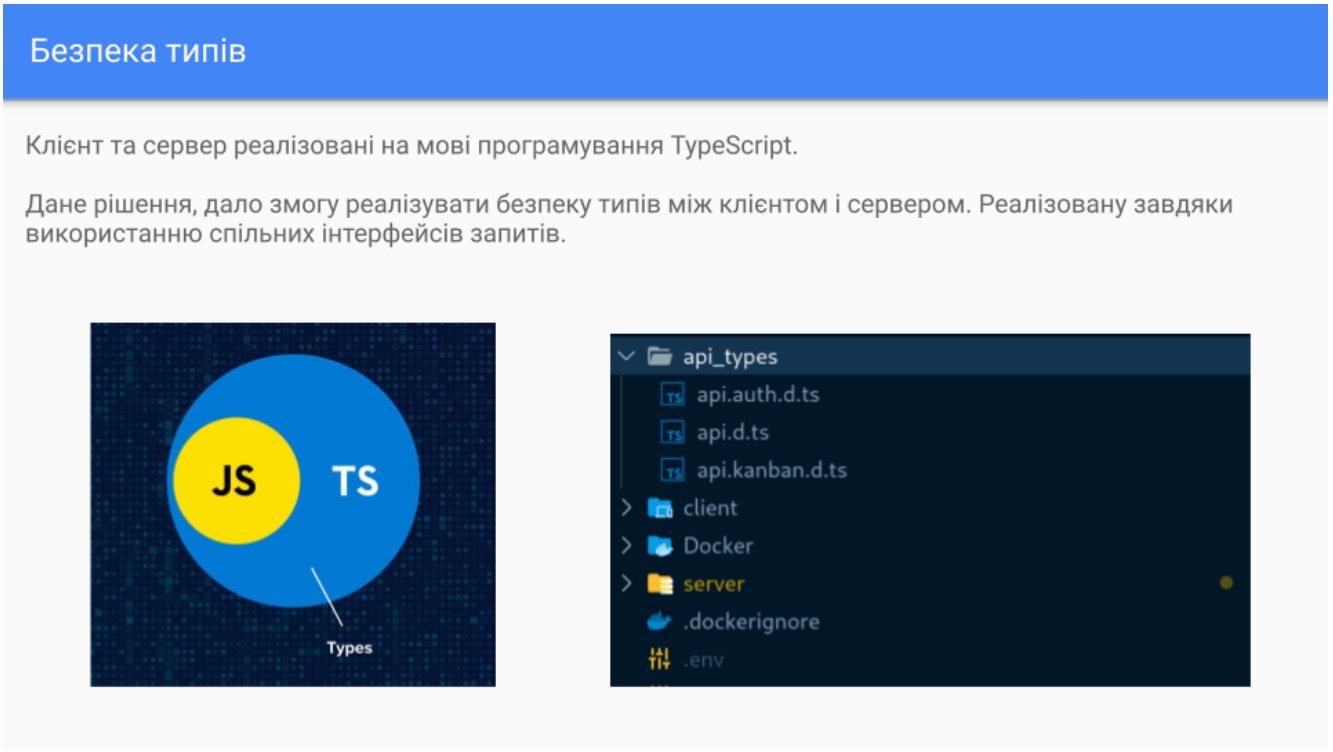


Рисунок Г.7 – Слайд презентації 7

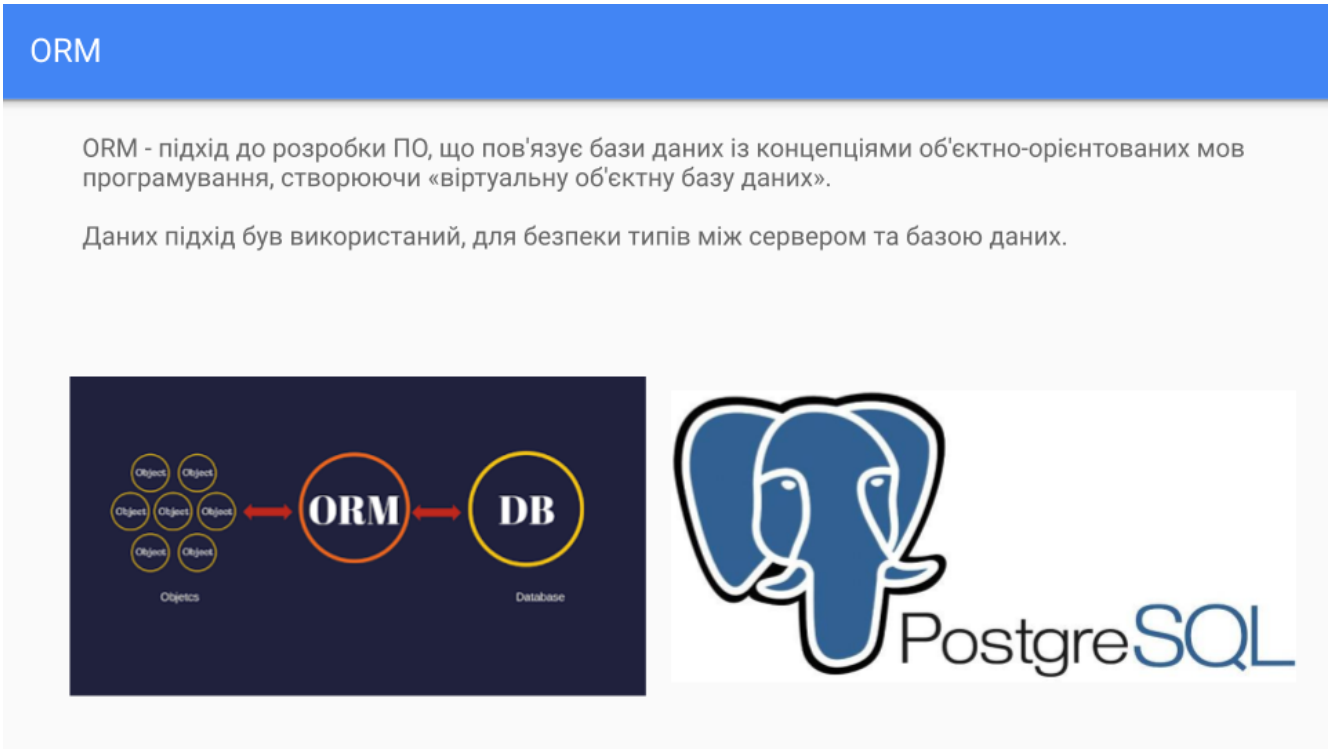


Рисунок Г.8 – Слайд презентації 8

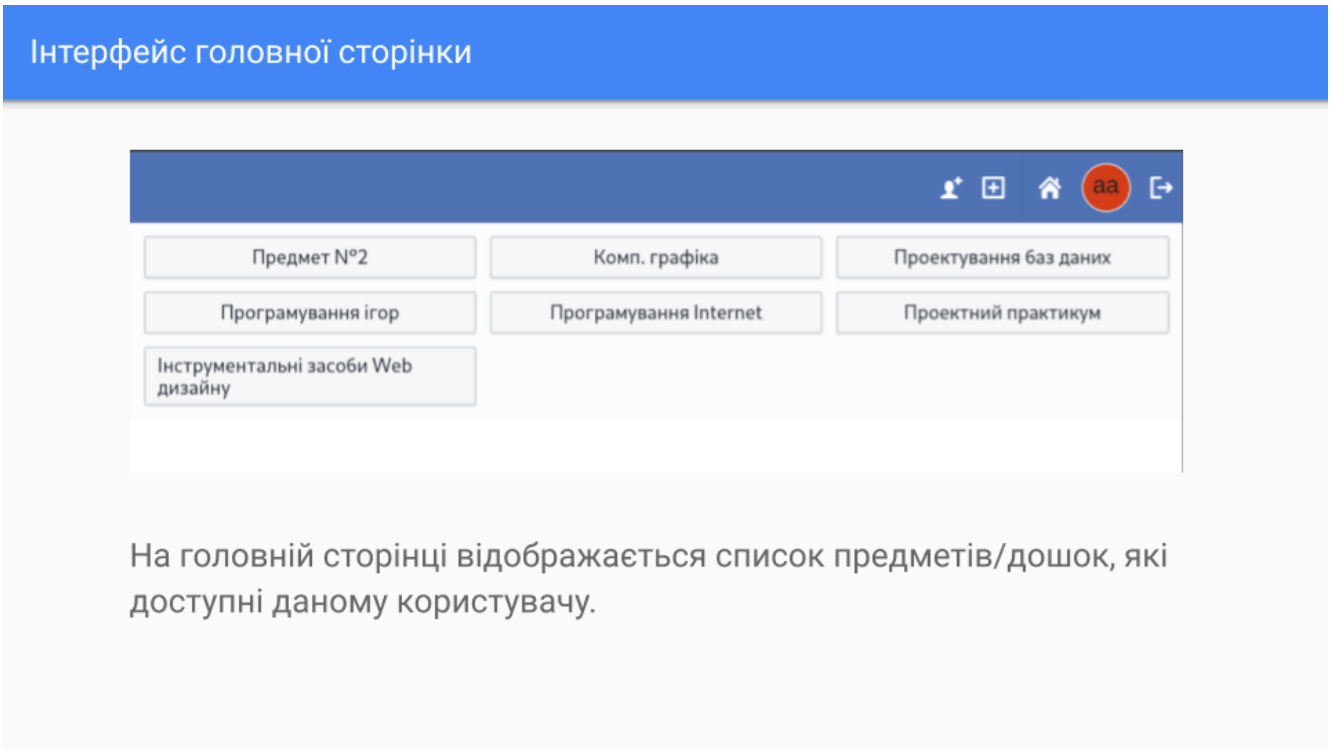


Рисунок Г.9 – Слайд презентації 9

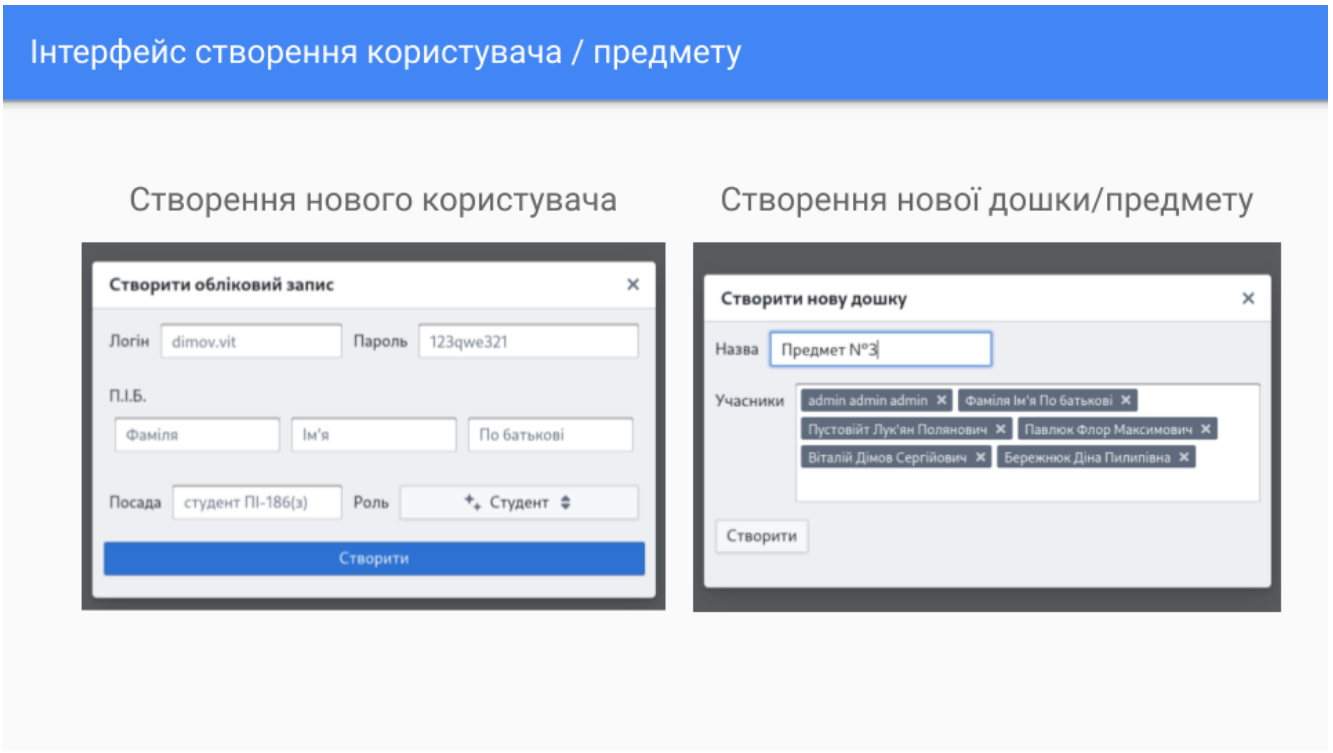


Рисунок Г.10 – Слайд презентації 10

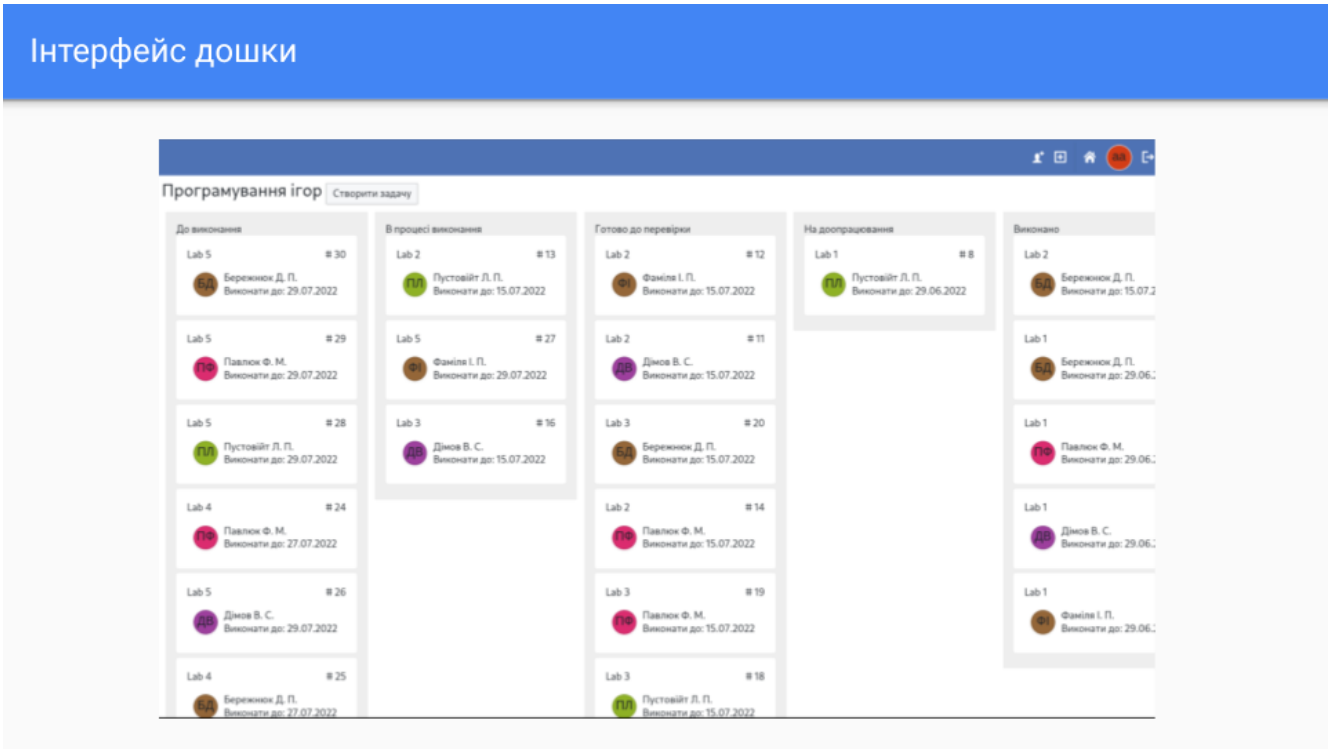


Рисунок Г11 – Слайд презентації 11

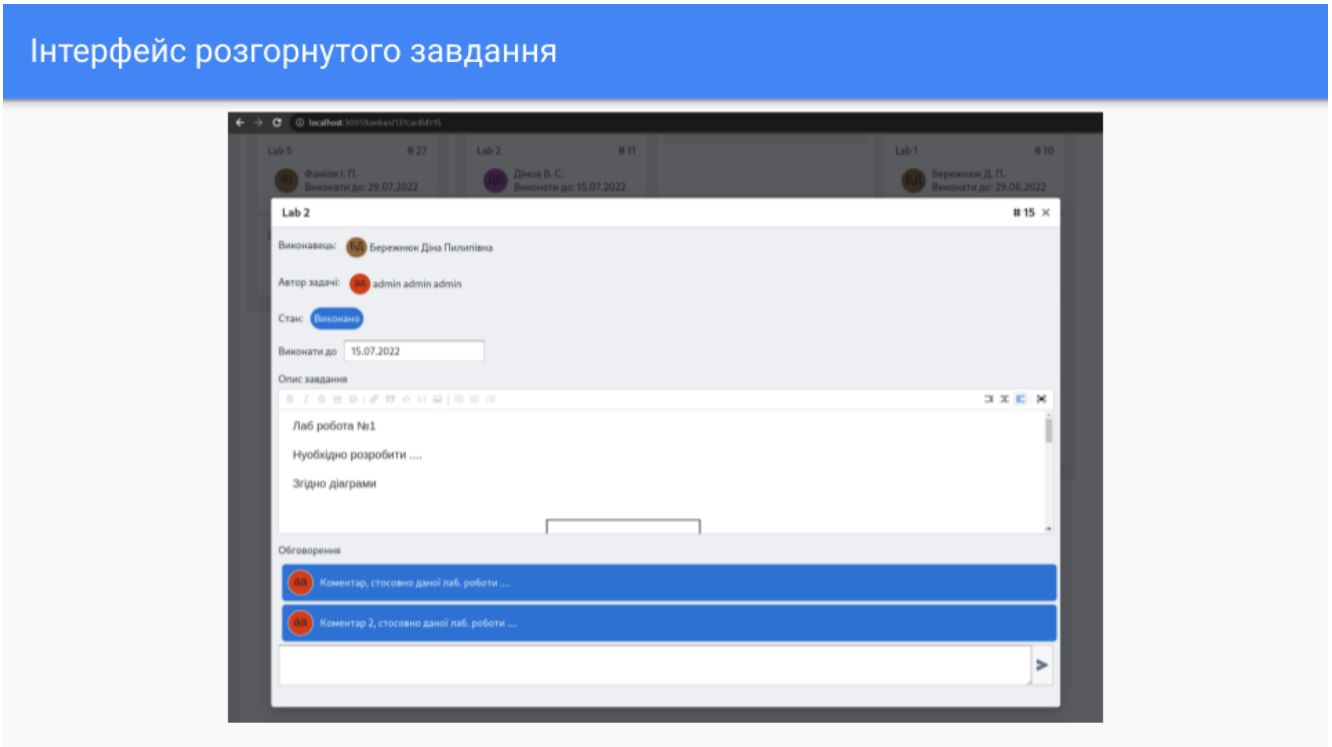


Рисунок Г.12 – Слайд презентації 12

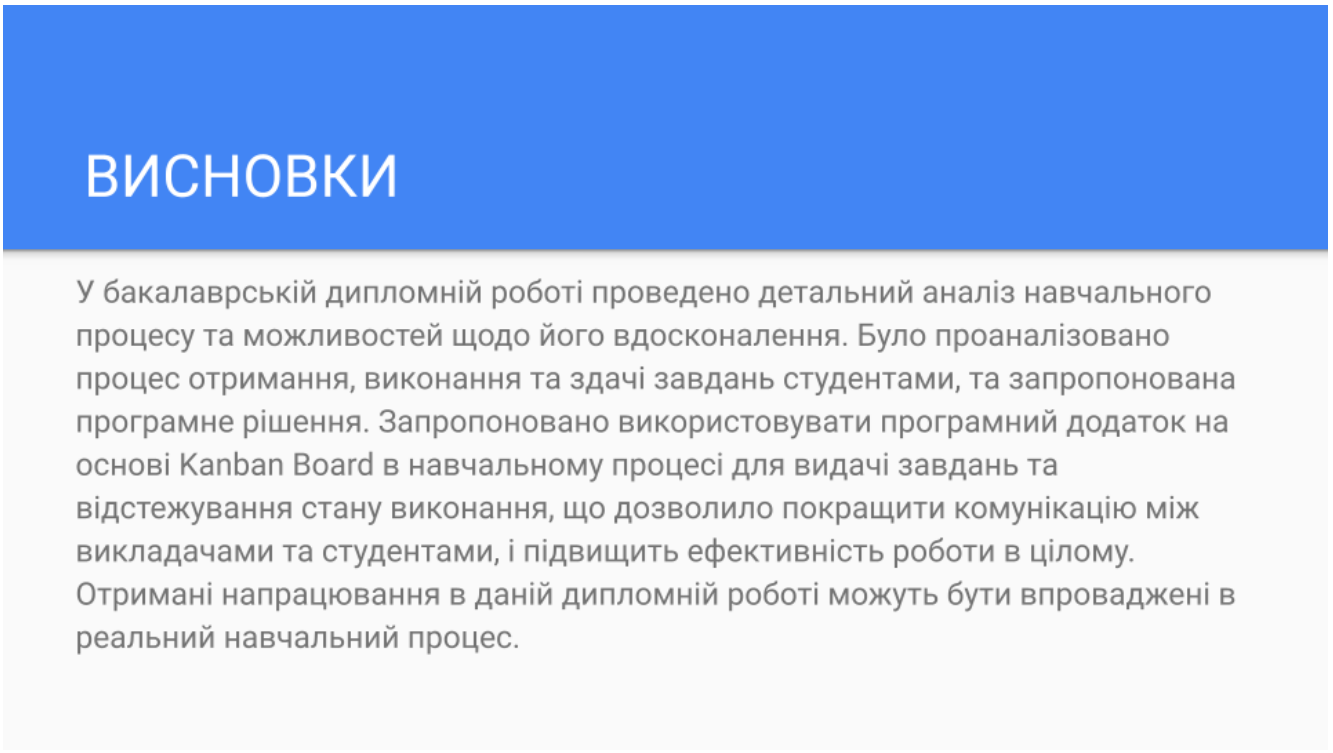


Рисунок Г.13 – Слайд презентації 13

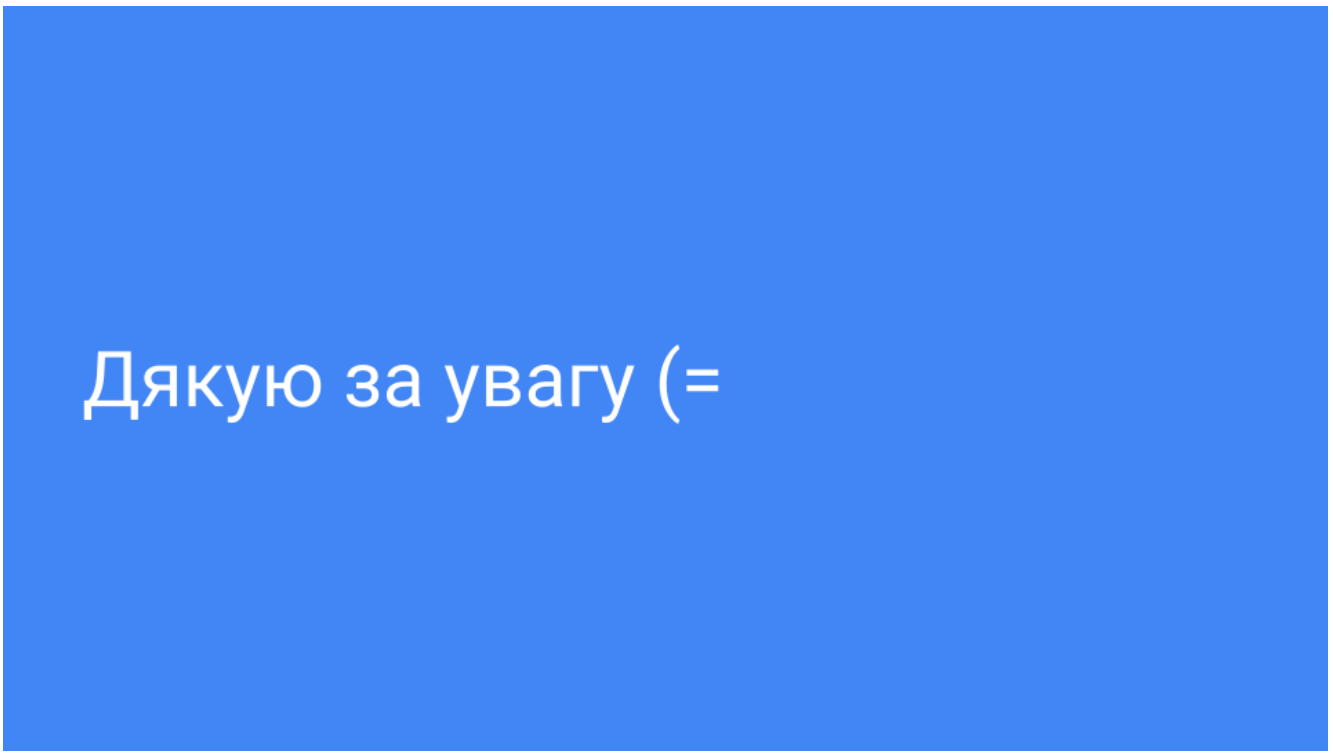


Рисунок Г.14 – Слайд презентації 14