МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНА СИСТЕМА ТРЕКІНГУ ВИКОНАННЯ ІНДИВІДУАЛЬНИХ НАВЧАЛЬНИХ ЗАВДАНЬ

Здобувач освіти гр. ІН-83 Олексій ГАЙДАБРУС

Науковий керівник,
кандидат фізико-математичних наук,
асистент кафедри комп'ютерних наук Олег БЕРЕСТ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-83 спеціальності «122 – Комп'ютерні науки» денної форми навчання Гайдабруса Олексія Андрійовича.

Тема: «ІНФОРМАЦІЙНА СИСТЕМА ТРЕКІНГУ ВИКОНАННЯ ІНДИВІДУАЛЬНИХ НАВЧАЛЬНИХ ЗАВДАНЬ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) інформаційний огляд; 2) вибір методів вирішення задачі; 3) програмна реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Олег БЕРЕСТ
Завдання прийняв до виконання _____ Олексій ГАЙДАБРУС

РЕФЕРАТ

Записка: стор. 59 , рис. 31, табл. 6, додаток 1, джерел 13.

Об'єкт дослідження – "Організація виконання індивідуальних навчальних завдань".

Мета роботи створення – інформаційної системи трекінгу для виконання індивідуальних навчальних завдань.

Методи дослідження – для реалізації Frontend частини було обрано HTML/CSS, Angular, Bootstrap, TypeScript , для реалізації Backend частини було обрано Spring, SpringBoot, Hibernate, PostgreSQL, Java.

Результати – було спроектовано та інформаційної системи трекінгу для виконання індивідуальних навчальних завдань. Після інтеграції клієнтської частини користувачу надається можливість авторизації та реєстрації, можливість створення проектів, можливість використовувати канбан-дошку, можливість відстеження термінів виконання завдань.

Зміст

Вступ

В сучасній сфері освіти чітко прослідковується тенденція впровадження інформаційних технологій(IT) в навчальний процес. Системи структуризації та відстеження навчальних робіт підвищують ефективність навчання та спрощують процес планування, як для вчителів так і студентам. Гарна структуризація навчального процесу є запорукою успішної самоосвіти, тому система відстеження виконання навчальних завдань буде користуватися попитом серед широкої аудиторії користувачів.

Головним функціоналом навчальних систем є збереження та структуризація завдань. Можливість візуалізації переліку завдань допомагає структурувати задачі за пріоритетом та складністю. Також такий підхід може допомогти для організації розробки власних проектів та самоосвіті.

Оскільки простота та доступність web-додатків надає можливість роботи з ними в будь-якому місці з доступом до мережі інтернет та обладнанням підтримуючим роботу з браузером, ефективно буде розробити інформаційну систему трекінгу виконання індивідуальних навчальних завдань саме web-орієнтованою. В умовах глобальної пандемії та переходу більшості навчальних закладів в Україні на дистанційну роботу саме web-орієнтоване рішення буде актуальне та затребуване в своїй області.

Отже метою даного проекту є розробка інформаційної системи трекінгу виконання індивідуальних навчальних завдань в web-орієнтованому рішенні.

Задачами проекту є підвищення якості та швидкості організації навчального процесу за рахунок інтеграції IT технологій в процеси моніторингу виконання індивідуальних навчальних завдань.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Дослідження предметної області

У програмного забезпечення, як у живої істоти є свій життєвий цикл. Життєвий цикл ПЗ [1] – стадії, що проходить програмний продукт від появи ідеї до її реалізації в коді, імплементації у бізнес і подальшої підтримки. Моделі життєвого циклу багато в чому зумовлюють і методології розробки ПЗ.

Зазвичай до етапів життєвого циклу відносять: аналіз вимог, проектування, програмування, тестування і налагодження, експлуатацію і підтримку.

Методологія розробки ПЗ [1] – це система, яка визначає порядок виконання завдань, методи оцінки та контролю. Методологія потрібна, щоб робота була структурована, щоб всі учасники команди розуміли, що зараз відбувається в компанії, над якими завданнями хто працює. Методології розробки, гнучкі і жорсткі, прийнято асоціювати з розробкою програмного забезпечення.

Дошка Kanban [2] – це інструмент управління проектами, який допомагає наочно уявити завдання, обмежити обсяг незавершеною роботи і домогтися максимальної ефективності (або швидкості). Вона може впорядкувати повсякденну роботу. За допомогою карток і стовпців на дошці Kanban можна зрозуміти, який обсяг роботи слід взяти на себе, і виконати цей обсяг, дотримуючись принципів безперервного вдосконалення.

Дошка – це обов'язковий елемент для гнучкої методології. Вона є в Scrum, є і в Kanban. Кожен член команди отримує до неї доступ в будь-який час і бачить, на якому етапі перебуває завдання.

Kanban-дошка підлаштовується під будь-який процес і застосовується в будь-якій області. Наприклад, щоб скласти список справ. Приклад простої канбан-дошки зображено на рисунку 1.1.

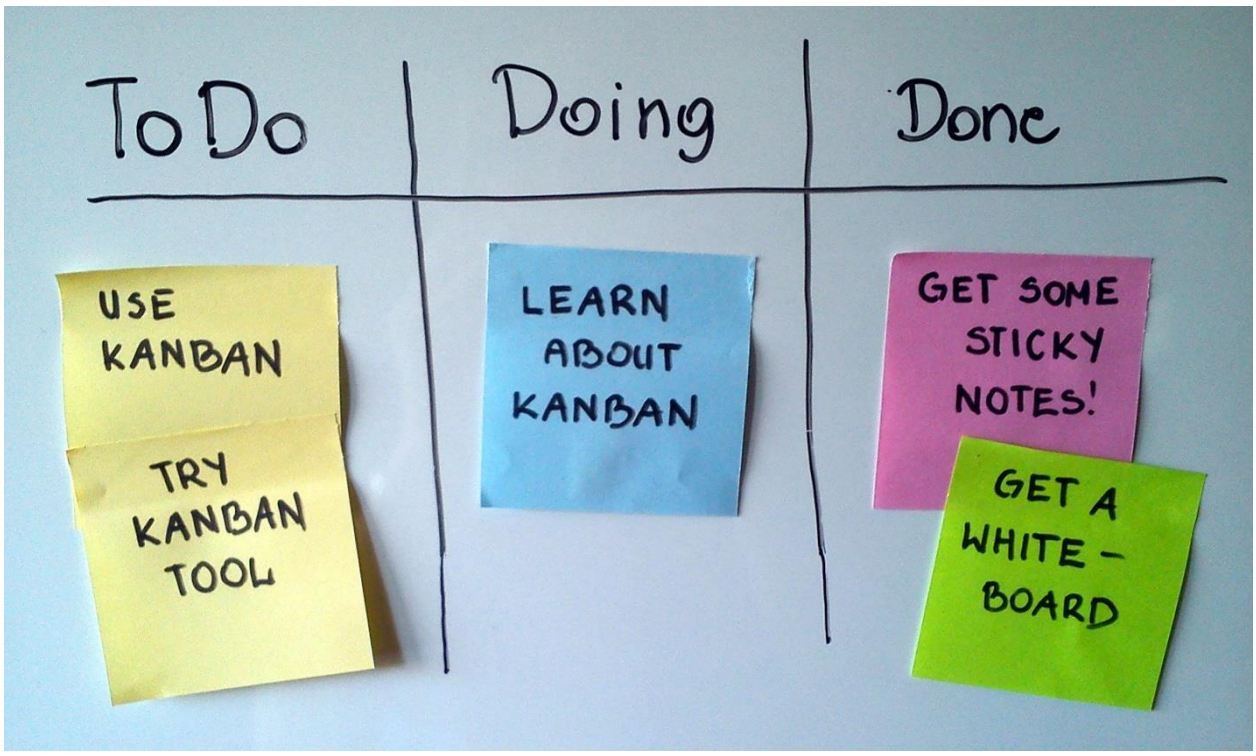

Рисунок 1.1 – Приклад канбан-дошки

У кожного проекту є план процесу робіт. Спочатку проект аналізуєьтся і дошка ділиться на стовпці, які відображають етапи. Наприклад, для процесу створення IT-проекту етапи можуть бути такими: на черзі, в роботі, аналіз, проектування, розробка, тестування, готово. Імена стовпців змінюються в залежності від проекту, але важливо зберігати. Kanban-картки [2] – це завдання, які рухаються по потоку і перетікають в інші стовпці в залежності від їх стану. На картці або стікері пишуть назву завдання і прикріплюють в початок дошки. На дошці відображаються всі процеси. Команда їх аналізує і усуває слабкі місця. У Kanban це називається управлінням потоком.

Візуалізація допомагає бачити картину цілком і коригувати окремі її частини, розуміючи, як зміни торкнуться весь проект. Отримати результат точно в срок можливо, якщо контролювати команди. Визначте кількість завдань: скільки яку кількість задач можна вирішувати в установлені терміни. Наприклад, в «Проектуванні» одночасно – не більш двох завдань, а на «Тестування» – тільки одна. Важливо знайти баланс: вибрати зручний темп роботи, який не шкодить термінах проекту. Для цього в Kanban враховують

час виконання кожного завдання. Так можна розуміє, що займає більше часу, а що – менше, і тим самим правильно організувати роботу.

1.2 Постановка задачі

Метою роботи є створення web-додатку інформаційної системи трекінгу виконання індивідуальних навчальних завдань. Він призначений для забезпечення можливості контролю рівня виконання індивідуальних завдань.

Для досягнення поставленої мети треба вирішити наступні задачі:

− проаналізувати існуючі технології розробки та обрати необхідну для реалізації проекту;

− виконати аналіз аналогів;

− розробити дизайн web-додатка;

− спроектувати базу даних інформаційної системи;

− реалізувати алгоритми серверної частини web-додатка;

− протестувати роботу розробленого web-додатку.

Цільовою аудиторією використання розроблюваного web-додатку є працівники в інтелектуальній сфері або учні які мають потребу в структуризації та відстеженні навчальних завдань.

Розробка призначена для підвищення якості навчального процесу та автоматизації моніторингу виконання навчальних завдань.

1.3 Огляд аналогів

Trello [3] – це інструмент управління завданнями, який дозволяє проявляти творчий підхід. Він використовує візуалізацію карток, списків і дощок в стилі Канбан, щоб інформувати про завдання, які необхідно виконати. Проекти в Trello представляють собою «дошки» зі списками завдань і картками, які призначаються користувачам і містять їх завдання або проекти. Картки можна переміщати і редагувати і навіть синхронізувати вміст з календарем. Trello дозволяє працювати над проектами як індивідуально, так і в команді. Trello доступний на комп'ютерах у вигляді деськтопной версії, в

браузерної версії і версії для мобільних пристроях. Картки мають безліч можливостей. Вони призначені для обговорень, голосувань, завантаження файлів і даних. Є можливість задавати дедлайни, призначати текстові та колірні мітки.

Мабуть головна причина чому Trello став популярний – це дуже простий та інтуїтивний інтерфейс, де користувач може швидко зорієнтуватися та почати вже працювати за лічені хвилини. Інтерфейс Trello зображено на рисунку 1.2.

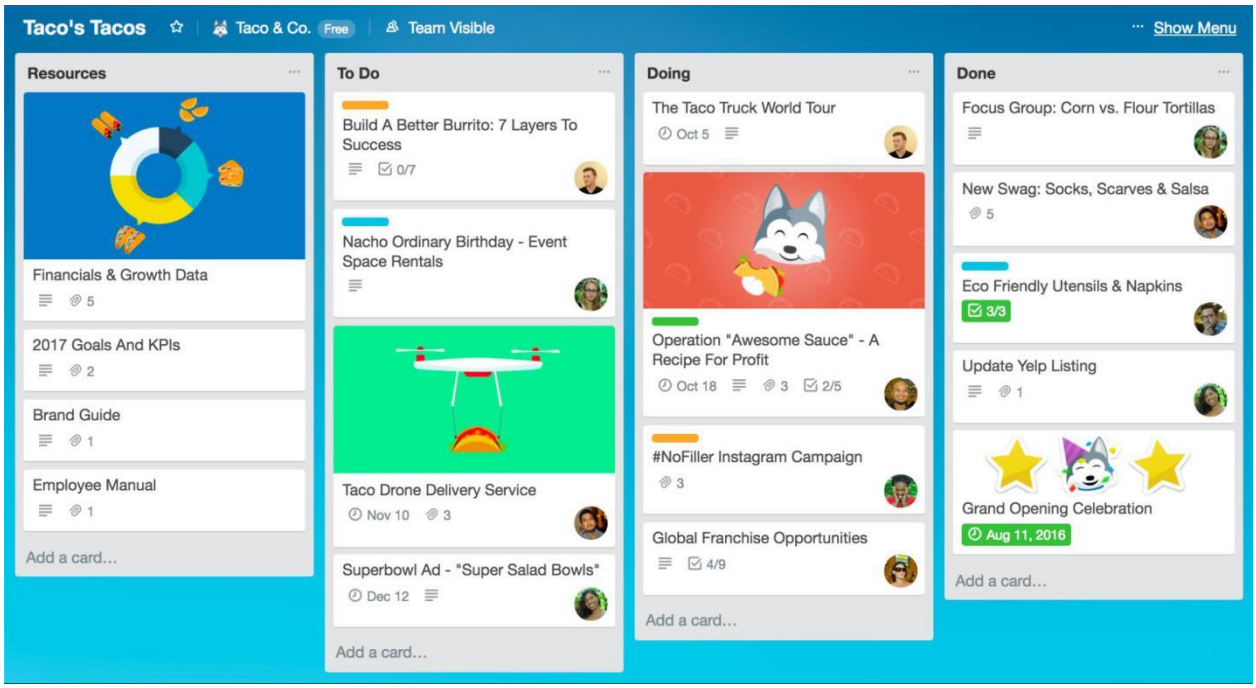

Рисунок 1.2 – Інтерфейс Trello

Але Trello має значні мінуси:

– Відсутня можливість відстеження часу;

– Немає можливості додавати опису проектів;

Basecamp – онлайн-сервіс для управління проектами з полегшеним призначеним для користувача інтерфейсом. Basecamp дозволяє збільшити продуктивність і організувати роботу над проектами. Сервіс об'єднує в собі інструменти для комунікації, відстеження завдань, планування і передачі файлів. Користувачам доступні версії в браузері і на мобільних пристроях. Робота в Basecamp проходить в єдиному просторі, де розміщені спільні

завдання і чат компанії, команди і проекти. Дані компоненти включають в себе 6 елементів для управління проектами: стрічка з новинами по проекту, лист завдань, документи і файли, загальний чат, календар і автоматичний чек-ін. У вкладці компанії розміщені загальні елементи, що відносяться до всієї команди. Інтерфейс Basecamp зображено на рисунку 1.3.

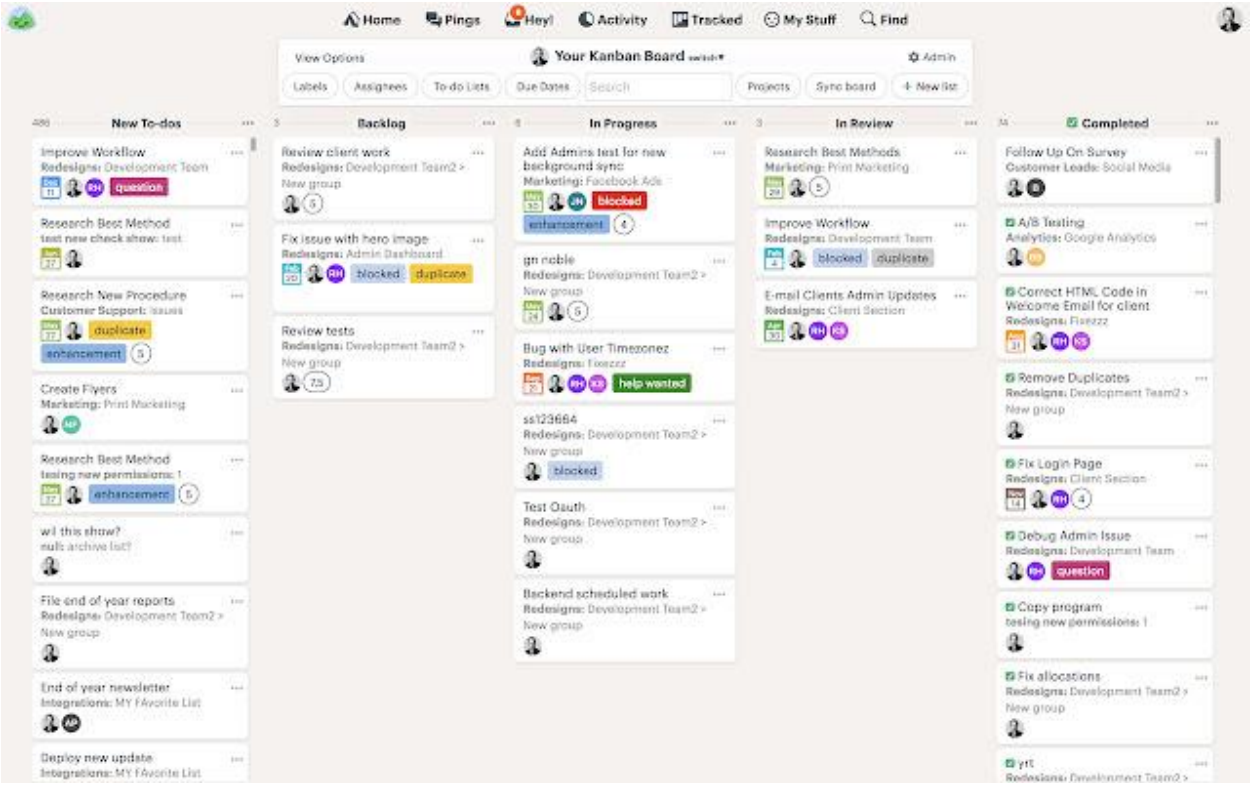

Рисунок 1.3 – Інтерфейс Basecamp

Мінуси Basecamp:

− Немає можливості відстеження часу;

− Зручність користування падає при великої кількості завдань та проектів;

− При огляді аналогів зрозуміло, що головним мінусом цих веб-сайтів являється те, що вони не мають можливості відстеження часу. Тому для покращення власного веб-сайту буде надана можливість відстеження часу.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір інструментів проектування веб-дизайну

Adobe illustrator надає ідеальне піксельне дизайнерське середовище для створення гнучких і вільних веб-елементів. Він пропонує все необхідне для створення чистого й чіткого веб-макета – векторну графіку, медіа-значки з реагуванням, масштабовані компоненти, генерацію CSS, експорт SVG, каркаси та багаторазові символи. Інтерфейс Adobe illustrator зображено на рисунку 2.1.

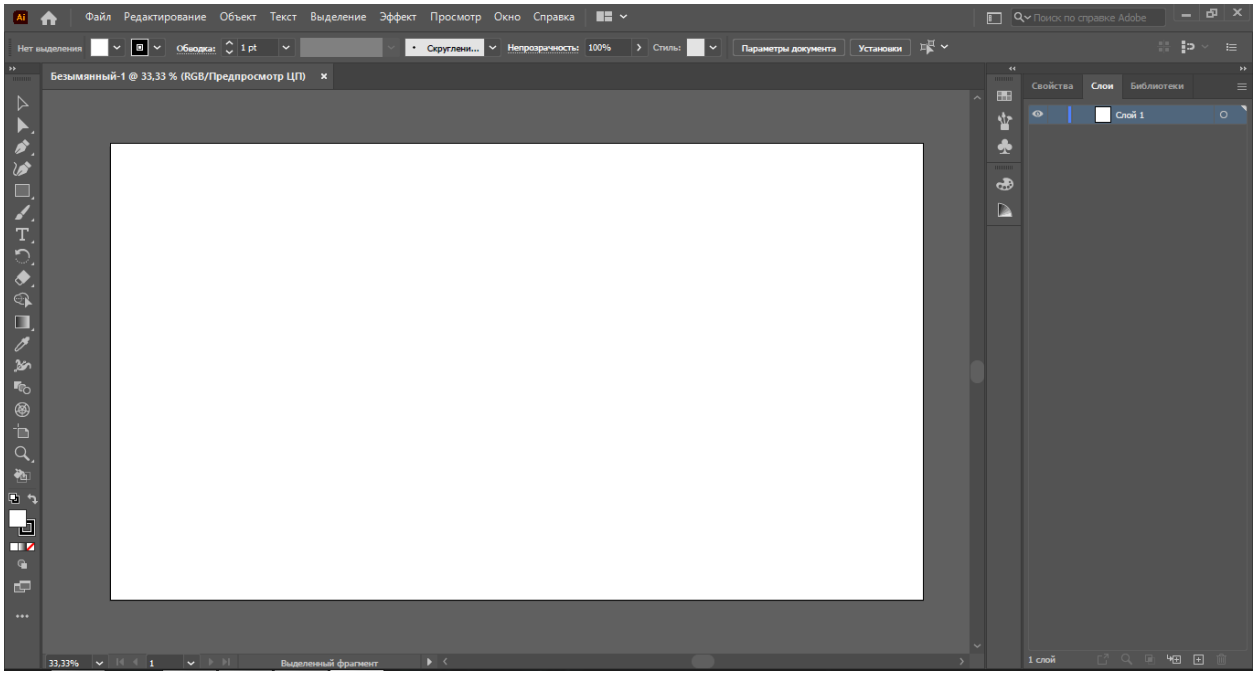

Рисунок 2.1 – Програма Adobe illustrator

Тому саме Adobe illustrator було обрано для проектування веб-дизайну.

2.2 Вибір інструментів для розробки Frontend частини

HTML, CSS, JavaScript — це найпопулярніша зв'язка технологій для створення сайтів. Близько 90% всіх сайтів працює саме завдяки цьому набору технологій. Кожна з цих технологій має різне призначення, цілі та функції. Але більшу цінність вони представляють, коли працюють разом, а не окремо.

HTML (HyperText Markup Language) — це мова гіпертекстової розмітки сторінки. Він використовується для того, щоб дати браузеру зрозуміти, як потрібно відображати завантажений сайт. Мова складається з тегів — це своєрідні команди, які перетворюються в візуальні об'єкти в браузері користувача. Наприклад, тег <img> використовується для розміщення зображень на сторінці. У нього є обов'язковий атрибут src, в якому вказується посилання на файл.

CSS (Cascading Style Sheets) — мова опису зовнішнього вигляду HTML-документа. CSS працюює з HTML, але це зовсім інша мова. HTML структурує документ і впорядковує інформацію, а CSS взаємодіє з браузером, щоб надати документу оформлення. Використання CSS направлено на те, щоб створити візуальне оформлення WEB-сторінки. При виконанні розмітки елементів мовою HTML документ отримає основну структуру сторінки. Це заголовки різного рівня, параграфи, марковані та нумеровані списки, відступи, картинки та інше. Але, якщо все залишити як є, то користувачеві на сторінці буде виведений безликий текст, який буде погано сприйматися. CSS, на відміну від HTML, може змінювати налаштування текстових блоків. За допомогою CSS задається шрифт, розмір цього шрифту, колір тексту і фону. Подібних властивостей дуже багато, а сучасні версії CSS дозволяють навіть працювати з анімацією. При створенні сторінки на сайті спочатку прописується HTML-код, що саме там буде відображатися. А за допомогою коду CSS можна задати візуальні налаштування текстової основи.

Bootstrap — це відкритий і безкоштовний HTML, CSS і JS фреймворк, який використовується веб-розробниками для швидкої верстки адаптивних дизайнів сайтів та веб-додатків. Bootstrap дозволяє верстати сайти в кілька разів швидше, ніж на «чистому» CSS і JavaScript. Ще один його аспект — доступність. Фреймворк Bootstrap — це набір набір CSS і JavaScript файлів. Щоб його використовувати ці файли необхідно просто підключити до сторінки. Після цього будуть доступні інструменти даного фреймворка: колоночная система (сітка Bootstrap), класи і компоненти. Bootstrap

складається з: інструментів для створення макета (обгортковий контейнерів, потужної системи сіток, гнучких медіа-об'єктів, адаптивних утилітних класів); класів для стилізації базового контенту: тексту, зображень, коду, таблиць; готових компонентів: кнопок, форм, горизонтальних і вертикальних навігаційних панелей, слайдерів, випадаючих списків, акордеонів, модальних вікон, спливаючих підказок і ін.; утілітних класів для вирішення традиційних завдань найбільш часто виникають перед веб-розробниками: вирівнювання тексту, відображення та приховування елементів, завдання кольору, фону, margin і padding відступів, і т.д.

Angular — один з найпопулярніших і сучасних JavaScript фреймворків, це набір готових рішень, що дозволяють швидко створювати складні й динамічні інтерфейси нHTML, CSS, JavaScript — це найпопулярніша зв'язка технологій для створення сайтів. Близько 90% всіх сайтів працює саме завдяки цьому набору технологій. Кожна з цих технологій має різне призначення, цілі та функції. Але більшу цінність вони представляють, коли працюють разом, а не окремо.

HTML (HyperText Markup Language) — це мова гіпертекстової розмітки сторінки. Він використовується для того, щоб дати браузеру зрозуміти, як потрібно відображати завантажений сайт. Мова складається з тегів — це своєрідні команди, які перетворюються ва сайтах. Angular є структурною основою для динамічних веб-додатків, яка дозволяє використовувати HTML у якості шаблону мови, а потім розширювати синтаксис HTML для виявлення компонентів додатків. За допомогою прив’язки даних та впровадження залежності можна виключити велику частину коду, яку можна було написати. Angular включає в себе:

− основний на компонентах фреймворка для створення масштабних веб-приложений;

− набор добре інтегрованих бібліотек, які охоплюють широкий спектр функцій: маршрутизація, управління формами, клієнт-серверне взаємодія і т.д;

− набор інструментів для розробки, які допомогають розробляти, збирати, тестувати та обновляти код.

2.3 Вибір інструментів для розробки Backend частини

Java — є універсальною мовою програмування, яка досягла широку популярність у сферах мобільних застосунків та веб-додатків. Особливості Java:

Кросплатформеність — це можливість запуску програмного коду на будь-якому пристрої, який має віртувальну машину Java або Java Virtual Machine, ця спеціальна програма яка виконує код. Якщо програмний код написано один раз, то він працює з будь-якою апаратною платформою або операційною системою: від смарткарт до застосунків для розумних будинків;

Ком'юніті. Java — набула широкої популярності та велика кількість розробників використовує цю мову програмування, за допомогою Java можна знайти вирішення практично будь-якої проблеми. У пригоді стануть тисячі бібліотек та форумів. На GitHub, наприклад, є відкриті проекти і документація, а на форумі Stack Overflow можна звернутися за допомогою до ком'юніті;

Надійність. Мова Java має строгу типізацію, це означає, що будь-яка змінна або вираз має певний тип вже на момент компіляції, що спрощує виявлення якихось помилок та проблем. Компілятор сам підказує розробнику, де припущені помилки;

Об'єктно-орієнтованість. Бібліотеки написані на Java — це набір класів, які відповідають за функціональність той чи іншой частини мови. Ці класи - це просто набір полів та методів, що описують об'єкти. Це добре, тому що дозволяє створювати складні, але прості у підтримці програми. І в цілому

Відносна простота. Java проста у використанні мова програмування, але має ряд важкий для розуміння концепцій, і в якойсь мірі складніша для вивчення ніж, наприклад, Python, але вона набагато простіша за мову C або C++. Функціональність мови оновлюється повільно, тому можна легко переходити на нові версії — заново вивчати не доведеться. Java — строго

типізована мова, це надає можливість відлідковувати помилки на ранніх етапах розробки.

Spring - один з найширше використовуваних фреймворків для розробки веб-додатків, що забезпечує продуману модель програмування і конфігурації. Метою створення цього фреймворка сприяло бажання спростити розробку додатків на популярному у той час Java EE стеку технологій від компанії Oracle, який на той момент був дуже складний і не завжди зручний у використанні.

На відміну від інших платформ, Spring фокусується на декількох областях функціонування додатків і надає для них широкий спектр додаткових функцій.

На відміну від інших платформ, Spring зосереджується на кількох областях функціонування додатків і надає для них широкий спектр додаткових функцій. Однією з основних особливостей Spring Framework є використання патерну Dependency injection (Di, впровадження залежності). Di допомагає набагато простіше реалізовувати необхідну додаткам функціональність, а також дозволяє розробляти слабо пов'язані класи, роблячи їх більше універсальними.

Якщо Spring Framework фокусується на наданні гнучкості, то Spring Boot прагне скоротити довжину коду і спростити розробку веб-додатку. Використовуючи конфігурацію за допомогою анотацій і стандартного коду, Spring Boot скорочує час, що витрачається на розробку додатків. Ця можливість допомагає створити автономні застосування з меншими або майже нульовими витратами на їх конфігурацію. Spring Boot має вбудований сервер, це означає, що не потрібно попередньо встановити його в середу розвертання. Spring Boot за умовчанням пропонує вбудований сервер Tomcat. Вбудовані сервери забезпечують більш ефективне розвертання і скорочують час перезапуску веб-додатку.

2.4 Проєктування веб-додатку

Перш ніж почати працювати з будь-яким сайтом, необхідно чітко продумати його структуру. Це безпосередньо впливає на ранжування ресурсу в пошукових системах, а також на його сприйняття користувачами. Ознайомившись з грамотною структурою, стає зрозуміло, чому два схожих за багатьма параметрами сайти мають абсолютно різні показники. Структура сайту — це схема розташування його сторінок. Це своєрідний план, в якому прослідковується логічний зв’язок між сторінками. З технічної точки зору навігація ресурсу являє собою набір URL, що розташовані в певній послідовності. Вона нерозривно пов’язана з семантичним ядром. Структура веб-додатку зображено на рисунку 2.2.

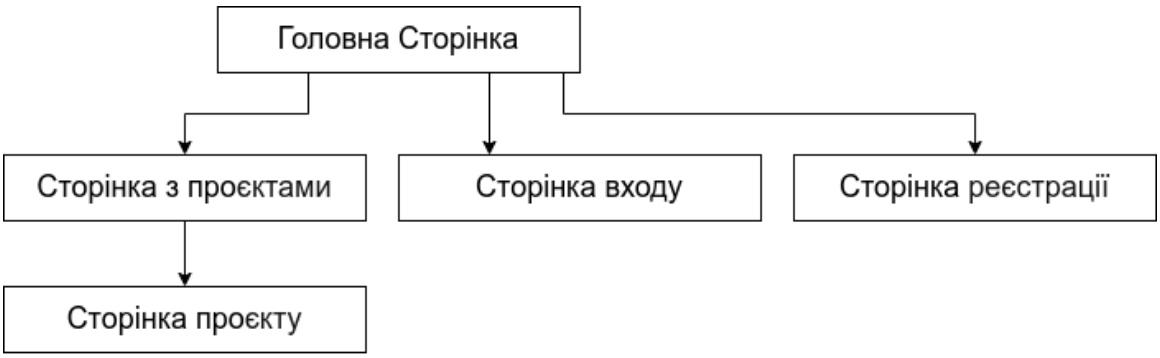

Рисунок 2.2 – Структура веб-додатку

Веб-додаток буде включати в себе 5 сторінок:

− головна сторінка;

− сторінка реєстрації;

− сторінка авторизації;

− сторінка з проектами.

2.5 Прототипування веб-додатку

Шаблон веб-сайту [4] було створено за допомогою таких інструментів як: прямокутник, відрізок лінії, еліпс. Стиль дизайну веб-сайта було розроблено в темних відтінках дотримуючись мінімалізму.

Головна сторінка складається з таких елементів як: шапка, контент, підвал. В шапці розміщені кнопки авторизації та реєстрації та назва веб-сайту та сам логотип. В контенті розташований опис веб-сайту та посилання на реєстрацію. Прототип головної сторінки зображено на рисунку 2.3.

Рисунок 2.3 – Прототип головної сторінки

Сторінки авторизації та реєстрації мають схожу будову: посередині розміщено форму з полями, які треба заповнити для виконання авторизації та реєстрації. Прототип сторінки авторизації на рисунку 2.4.

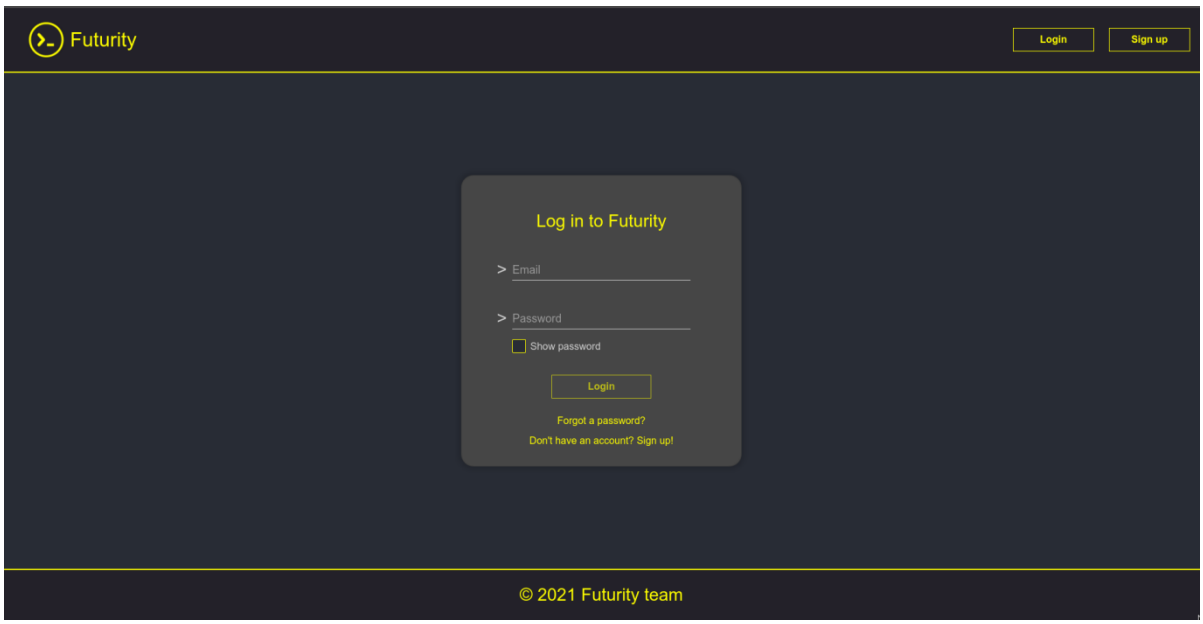

Рисунок 2.4 – Прототип сторінки авторизації

Сторінка реєстрації містить 2 форми для заповнення. Перша форма для перевірки почти зображена на рисунку 2.5.

Рисунок 2.5 – Прототип сторінки реєстрації (1-ша форма)

Друга форма для заповнення реєстраційної інформації зображена на рисунку 2.6.

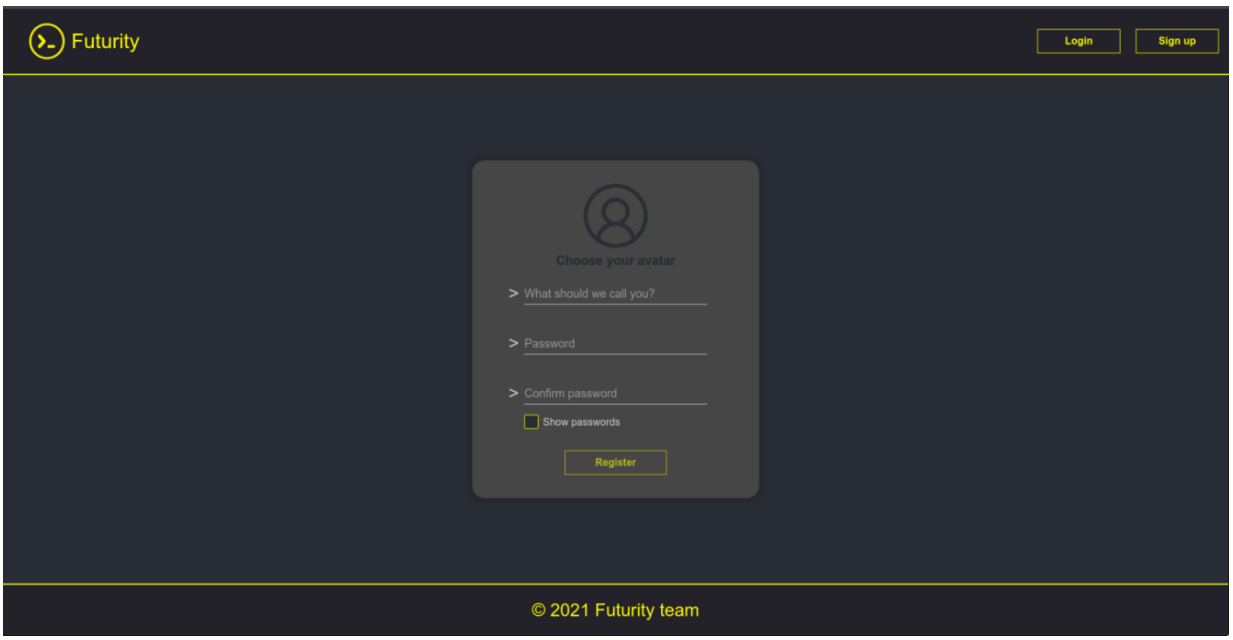

Рисунок 2.6 – Прототип сторінки реєстрації (2-га форма)

Сторінка з проектами складається з таких елементів як: шапка та контент. В шапці розташовано аватар користувача, логотип та назва сайту, кнопка виходу. В контенті розташовані таблички з проектами, які створив користувач та кнопка додавання нового проекту. Табличка проекту складається з картинки проекту, назви та опису. Прототип сторінки з проектами зображено на рисунку 2.7.

Рисунок 2.7 – Прототип сторінки з проектами

Сторінка проекту складається з таких елементів як: шапка та контент. В шапці розташовано аватар користувача, логотип та назва сайту, кнопка виходу. В контенті розташована канбан дошка з колонками, які створив користувач. Прототип сторінки з проектами зображено на рисунку 2.8.

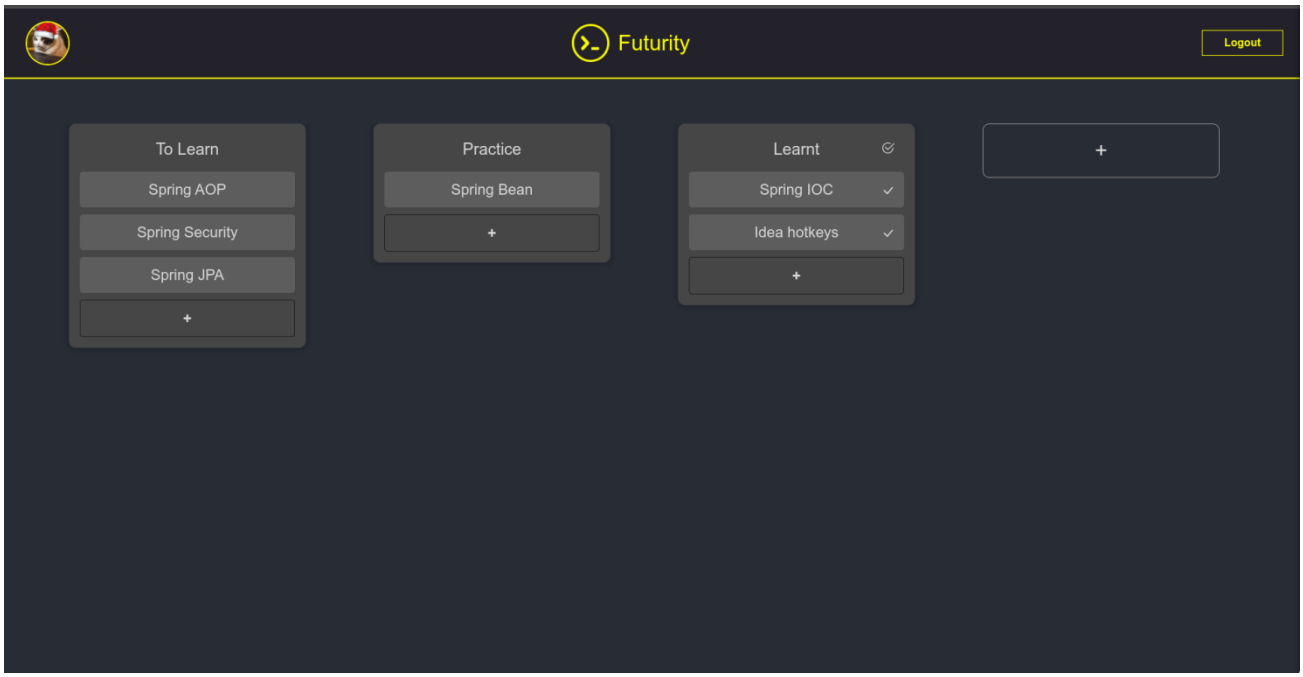

Рисунок 2.8 – Прототип сторінки проекту

Дизайн web-додатку оформлено в мінімалістичному стилі, для забезпечення інтуїтивної зрозумілості для будь-якого типу користувачів. Результатом проектування є прототип сторінок web-додатку інформаційної системи трекінгу виконання індивідуальних навчальних завдань.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка архітектури

Для реалізації веб-додатку було обрано мікросервісну архітектуру. Архітектура веб-додатку зображена на рисунку 3.1.

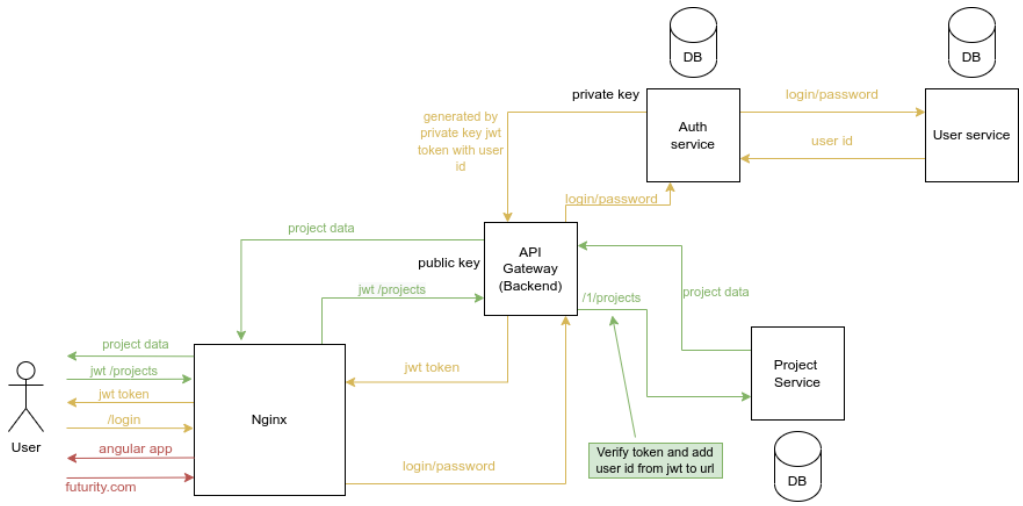

Рисунок 3.1 - Архітектура веб-додатку

Веб-додаток оснований на клієнт-серверної архітектури, в основі якої лежить 2 поняття - клієнт та сервер. Клієнт[5] – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій. Сервер[5] – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних. Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше. Модель клієнт-серверної архітектури зображено на рисунку 3.2

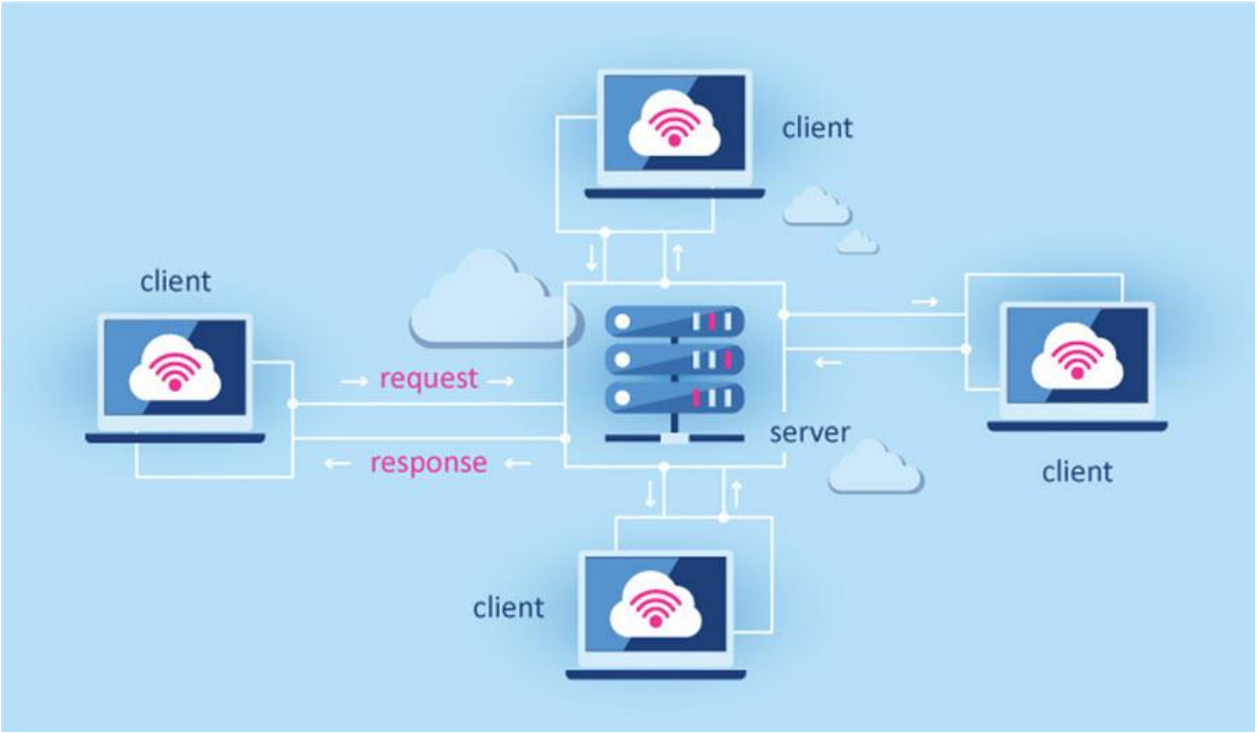

Рисунок 3.2 - Клієнт-серверна архітектура

Сервер виконує такі функції, як:

– зберігання, доступ, захист і резервне копіювання даних;

– обробка клієнтського запиту;

– відправлення результату (відповіді) клієнту.

Клієнт виконує такі функції, як:

– надання користувальницького інтерфейсу;

– формулювання запиту до сервера і його відправка;

– отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

Такий підхід має досить значні переваги:

– Виконання більшої частини роботи на сервері при мінімумі навантаження на клієнта.

– Більшість даних зберігаються на сервері, який, зазвичай, краще захищений від різноманітних загроз, ніж звичайний клієнтський ПК.

– Можливість чіткішого розмежування повноважень доступу до різних рівнів інформаційної системи. Кожному клієнту – свій рівень доступу.

− Кроссплатформеність, тобто будь-який клієнт може працювати з ресурсами сервера незалежно від операційної системи, що використовується.

3.2 Розробка Backend частини

Backend частина буде складатися з 4 мікросервісів. Кожен мікросервіс відповідальний за свою частину роботи. Мікросервіс “API Gateway” реалізує один з популярних паттернів в мікросервісній архітектурі “API Gateway”, мета цього мікросервісу - це розділення клієнтського програмного інтерфейсу від внутрішньої реалізації та виконання аунтентификації. Цей мікросервіс приймає всі запити та відпрацьовує як зворотний проксі, витягуючи ресурси із внутрішніх мікросервісів від імені клієнтської програми. Мікросервіс “Auth service” Відповідальний за авторизацію, тобто цей мікросервіс виконує такі функції, як: логін, реєстрація, розсилка повідомлень на пошту. Мікросервіс “User service” відповідальний за керування користувачами та іх даними, які зареєстровані в системі: збережння користувачів, додавання нових користувачів, пошук вже існуючих, та різні маніпуляції з іх даними (логін, пошта, аватар і т.д.). “Project service” відповідальний за керування проектами та задчами, які створив користувач: додавання проекту, видалення проекту, зміна опису, додавання нових задач до проекту, тощо. Лістинг головних модулів backend частини наведено в додатку А.

Доступ до захищених ресурсів буде відбуватися за допомогою JWT токену, який буде генеруватись Auth мікросервісом за допомогою раніше створеного приватного ключа, також цей JWT токен буде перевірятися Api-Gateway мікросервісом за допомогою публічного ключа.

Одним із принципів REST [6] є незалежність від стану (stateless). Це означає, що клієнт повинен сам подбати про аутентифікацію при кожному запиті. Як раз саме для цього використовується JWT токен. Найпростіший підхід для аутентифікації в REST це надсилання логіну та пароля користувача при кожному запиті. Зрозуміло, що такий спосіб не є безпечним, особливо якщо клієнт використовує незахищений протокол. Більш звичайне рішення -

зіставлення користувача нікому унікальному ідентифікатору - токену. При першому логіні клієнту від сервера видається токен, утворений хеш-функцією від якихось унікальних даних користувача (id, логін, пароль). У основу заноситься пара токен - id. За наступних запитів клієнт передає цей токен, а сервер шукає у базі запис. Якщо запис знайдено, авторизується. Часто, для більшої безпеки, токену дають певний час життя, після якого він стає недійсним. Такий підхід зображений на рисунку 3.3.

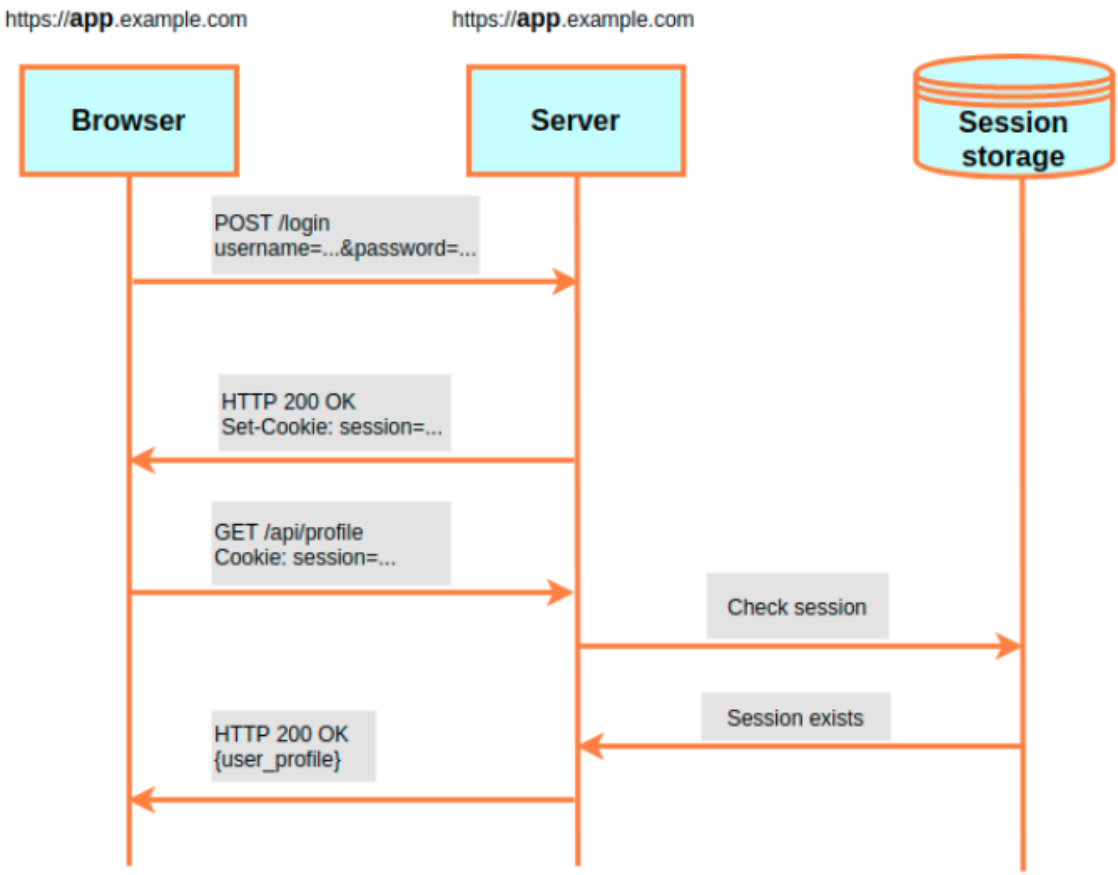

Рисунок 3.3 - Аутентифікація на основі сесій

JSON Web Token працює схоже зі звичною реалізацією. Але JWT має деякі переваги — він є самодостатнім, всі необхідні для аутентифікації дані можна зберігати в самому токені. JWT складається з трьох основних частин: заголовка (header), навантаження (payload) та підпису (signature). Заголовок та навантаження формуються окремо, а потім на їх основі обчислюється підпис. Зазвичай заголовок (header) складається з двох полів: типу токена (в даному

випадку JWT) та алгоритму хешування підпису. Payload – це будь-які дані, які можна передати в токені. Підпис (signature) обчислюється на основі заголовка та навантаження. Таким чином, якщо хтось спробує змінити дані в токені, він не зможе змінити підпис, не знаючи приватного ключа. При аутентифікації приватним ключем може бути пароль користувача (або хеш від пароля). Після першого логіну клієнту повертається згенерований сервером JWT. При кожному наступному запиті клієнт повинен передавати JWT встановленим API способом (наприклад, через заголовок або як параметр запиту). Сервер декодує header та payload та перевіряє зарезервовані поля. Якщо все гаразд, за вказаним у header алгоритмом складається підпис. Якщо отриманий підпис співпадає з переданим, користувача авторизують. Такий підхід зображений на рисунку 3.4.

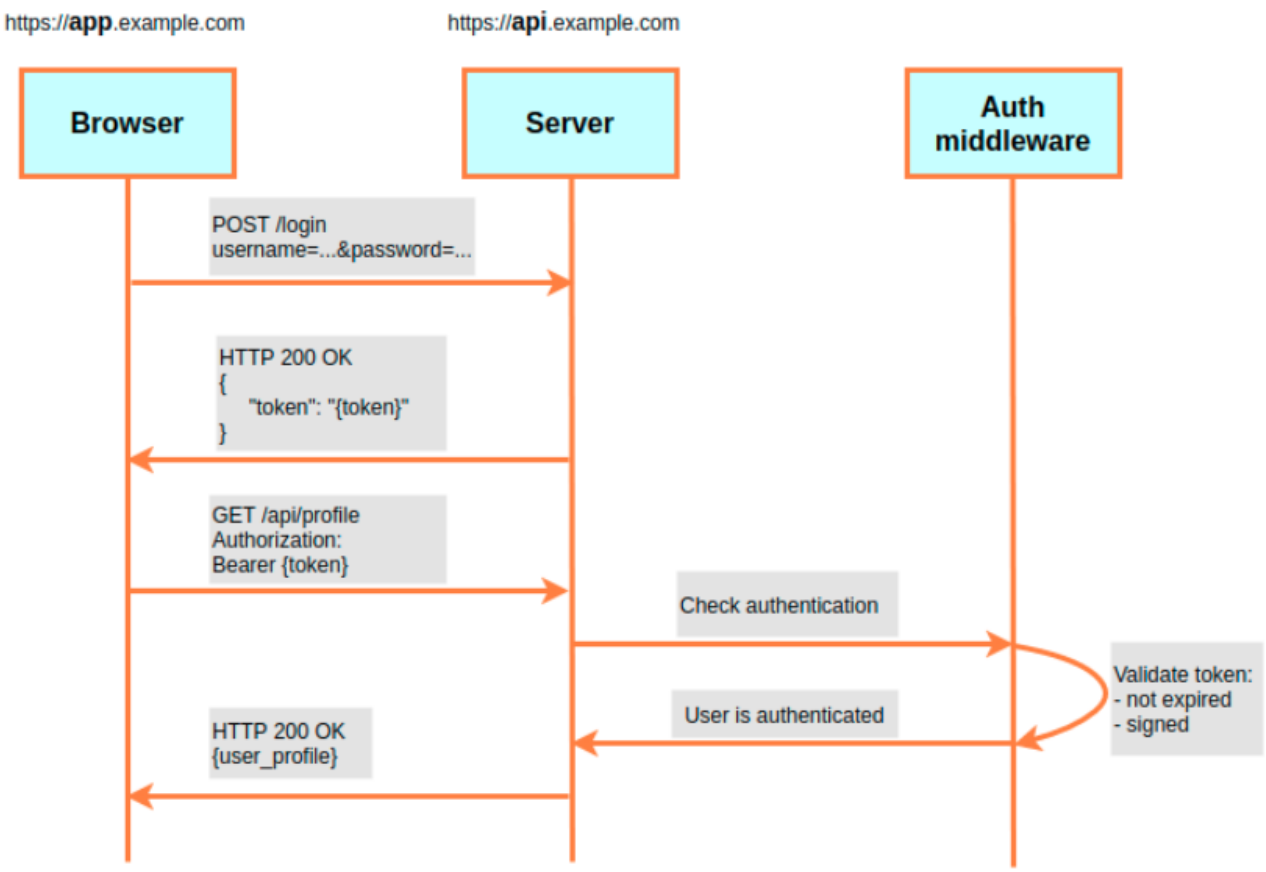

Рисунок 3.4 - Аутентифікація на основі токенів

Для більшого захисту, Auth мікросервіс буде генерувати пару токенів, а саме access токен та refresh токен, access токен буде “закінчюватися” через 15 хвилин, що значить при його викрадені у того, хто його викрав буде лише 15 хвилин щоб скористатися викраденим токеном, але користувачу прийдеться кожні 15 хвилин авторизуватися, що досить не зручно. Цю проблему вирішує refresh токен, який буде зберігатися в кукі з флагом “httpOnly” і за допомогою refresh токену можна генерувати нові access токени після того як access токен закінчиться без участі користувача. Діаграма послідовності access та refresh токену зображена на рисунку 3.5, також послідовність роботи зображено на рисунку 3.1 жовтим кольором.

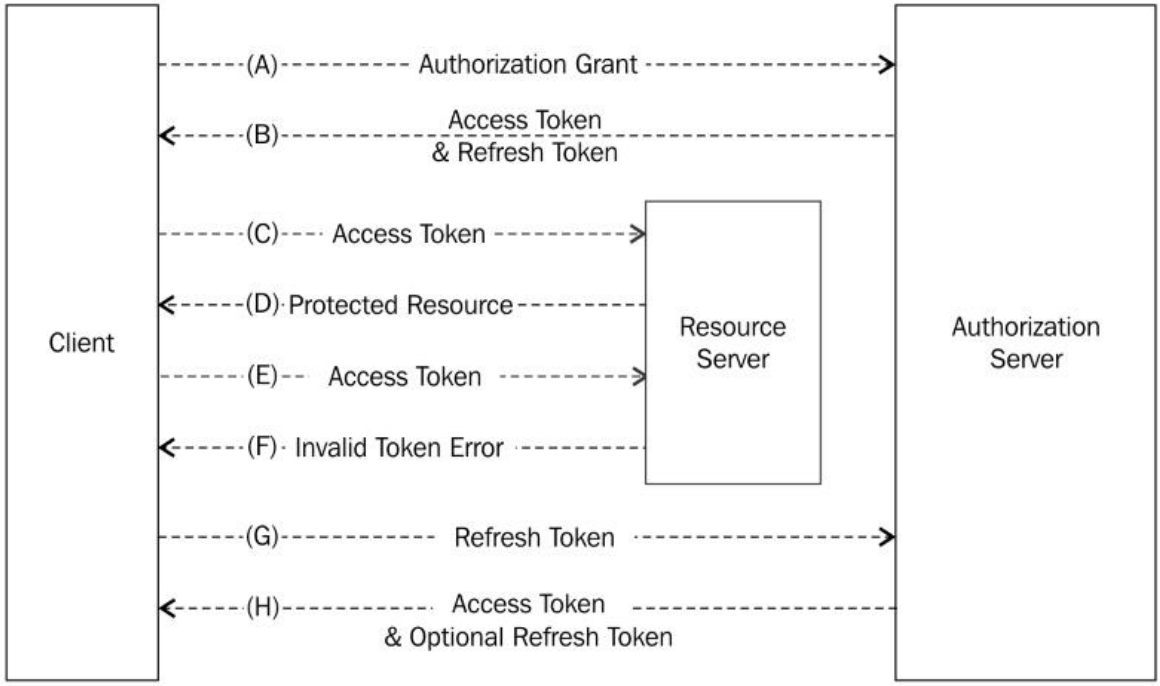

Рисунок 3.5 - Діаграма послідовності access та refresh токену

Користувач з кожним запитом посилає access токен і Api-Gateway перевіряє цей токен на коректність, чи не змінив його користувач, якщо токен вірний, то Api-Gateway дістає ID користувача і додає його в URL запиту і надсилає цей запит далі до захищенного ресурсу, тому другим мікросервісам не потрібно перевіряти чи має той чі інший користувач доступ до ресурсу, все це робить Api-Gateway. Діаграма послідовності доступу до захищеного

ресурсу зображена на рисунку 3.6, також послідовність запиту зображено на рисунку 3.1 зеленим кольором.

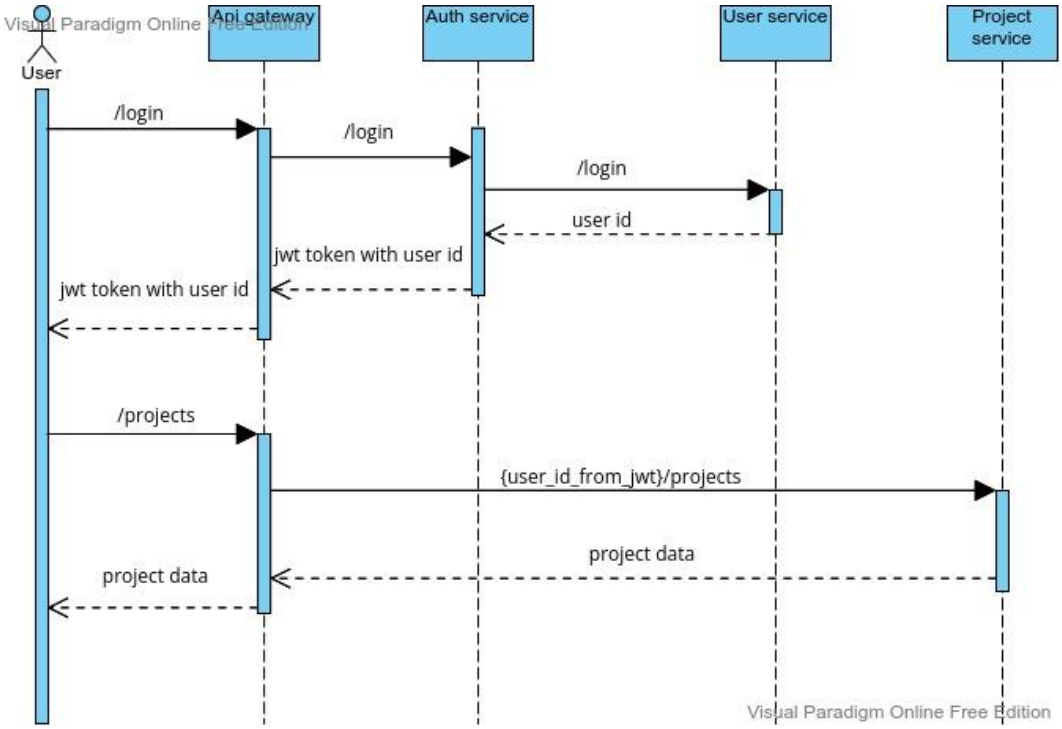

Рисунок 3.6 - Діаграма послідовності доступу до захищеного ресурсу

Реєстрація нового користувача буде проходити в 2 етапи, перший етап - це верифікація електронної адреси користувача за допомогою повідомлення з кодом підтвердження, другий етап - це безпосередньо реєстрація користувача та збереження його авторизаційних даних. Діаграма послідовності реєстрації зображена на рисунку 3.7.

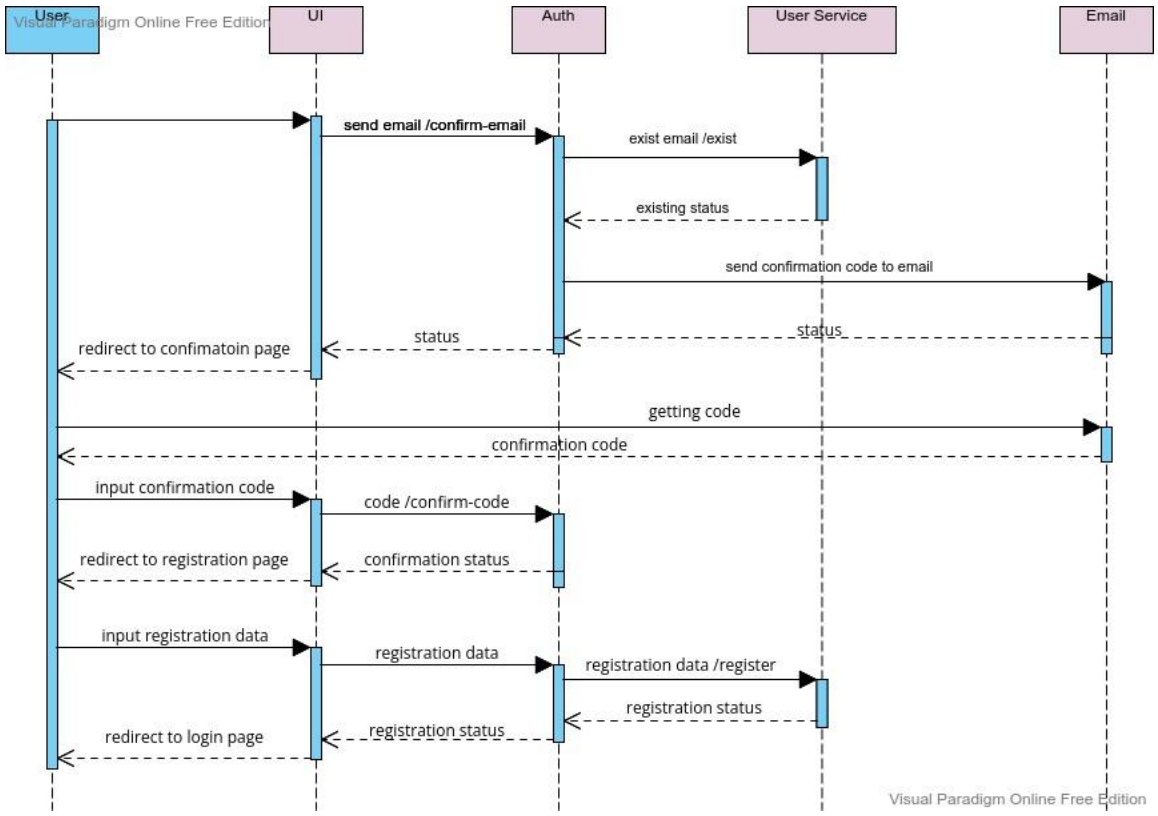

Рисунок 3.7 - Діаграма послідовності реєстрації користувача

REST (Representational State Transfer) – це стиль архітектури API для розподілених систем. REST визначає, як дані надаються клієнту в зручному для нього форматі. Обмін даними відбувається у форматі JSON або XML. REST підхід володіє наступними принципами:

− Відділення клієнта від сервера (Client-Server)[6]. Клієнт — це інтерфейс сайту або програми, наприклад. У REST API код запитів залишається на стороні клієнта, а код доступу до даних - на стороні сервера. Це спрощує організацію API, дозволяє легко переносити інтерфейс користувача на іншу платформу і дає можливість краще масштабувати серверне зберігання даних.

− Відсутність запису стану клієнта (Stateless)[6]. Сервер не повинен зберігати інформацію про стан (проведені операції) клієнта. Кожен запит від клієнта повинен містити тільки інформацію, яка потрібна для отримання даних від сервера.

− Єдність інтерфейсу (Uniform Interface). Усі дані повинні запитуватись через одну URL-адресу стандартними протоколами, наприклад, HTTP. Це спрощує архітектуру сайту або програми та робить взаємодію з сервером зрозумілішим.

− Надання коду на запит (Code on Demand). Сервери можуть надсилати клієнтові код (наприклад, скрипт для запуску відео). Так загальний код програми або сайту стає складнішим лише за необхідності.

Backend частина працює на основі архітектури Rest API, обмін даними відбувається у форматі JSON[6].

3.3 Проектування бази даних

Після створення архітектури необхідно спроектувати базу даних. Для зручного додавання та редагування інформації на web-ресурсі використовують системи управління базами даних (СУБД), а саме PostgreSQL. Результатом етапу проектування є визначена структура, яка складається з таблиць та логічних зв’язків між ними. Структуру таблиці бази даних визначає склад і послідовність стовпців, їх типи даних і розмір, первинний ключ.

Першим етапом проектування бази даних є виділення сутностей. Сутність – це об’єкт предметної області, який має зберігатися в базі даних, визначається набором атрибутів. У базі даних повинно бути представлено по одній таблиці на кожну сутність. Кожен стовпець таблиці містить атрибут, а рядок – екземпляр сутності.

Під час проектування даних було виділено наступні сутності:

− Користувач (User) - сутність, яка описує дані користувача;

− Код підтвердження (Confirmation token) - сутність, яка використовується для верифікації електронної пошти користувача;

− Refresh токен - сутність, яка використується для генерування нових access токенів;

− Проект (Project) - сутність, яка описує дані проекту;

− Колонка проекту (Project Column) - сутність, яка описує колонку проекту;

− Завдання (Task) - сутність, яка описує задачу.

Опис полів для сутності “Користувач” наведено в таблиці 3.1.

Таблиця 3.1 - Опис полів сутності “Користувач”

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації користувача
email	varchar(100)	Not null, Unique	Електронна пошта користувача
nickname	varchar(100)	Not null	Ім’я користувач
password	varchar(100)	Not null	Пароль користувача
avatar	blob	Not null	Аватар користувача

Опис полів для сутності “Код підтвердження” наведено в таблиці 3.2.

Таблиця 3.2 - Опис полів сутності “Код підтвердження”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації коду підтвердження
code	char(100)	Not null, Unique	Код, який надсилається на пошту для підтвердження
confirmed	boolean	Not null	Використовується для з’ясування чи підтверджена пошта користувача чи ні
confirmed_at	timestamp	Nulll	Дата, коли була підтверджена пошта користувача
expired_at	timestamp	Not Null	Дата, коли код підтвердження стане не дійсним

Опис полів для сутності “Refresh токен” наведено в таблиці 3.3.

Таблиця 3.3 - Опис полів сутності “Refresh токен”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації refresh токену
refresh_token	varchar(400)	Not null, Unique	Використується для генерування нових access токенів

Опис полів для сутності “Проект” наведено в таблиці 3.4.

Таблиця 3.4 - Опис полів сутності “Проект”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації проекту
user_id	bigint	Not null	Використовується для зв’язку з сутністю “Користувач” до якого відноситься проект
name	varchar(60)	Not null	Назва проекту
description	varchar(1000)	Not null	Опис проекту
preview	blob	Not null	Прев’ю проекту

Опис полів для сутності “Колонка проекту” наведено в таблиці 3.5.

Таблиця 3.5 - Опис полів сутності “Колонка проекту”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації колонки проекту
index	integer	Not null	Індекс колонки по відношенню до інших колонок в проекті
name	varchar(60)	Not null	Назва колонки
project_id	bigint	Not null	Використовується для зв’язку з сутністю “Проект” до якої відноситься колонка
preview	blob	Not null	Прев’ю проекту
done_column	boolean	Not null	Використовується для з’ясування чи колонка для виконаних задач чи ні.

Опис полів для сутності “Задача” наведено в таблиці 3.6.

Таблиця 3.6 - Опис полів сутності “Задача”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації задачі
index	integer	Not null	Індекс задачі по відношенню до інших колонок в проекті
name	varchar(60)	Not null	Назва задачі
column_id	bigint	Not null	Використовується для зв’язку з сутністю “Колонка проекту” до якої відноситься задача
deadline	timestamp	Not null	Термін виконання задачі

В мікросервісній архітектурі кожен мікросервіс має свою власну базу даних, але якщо упустити цей момент, то ERD діаграма виглядає наступним чином (Рисунок 3.8).

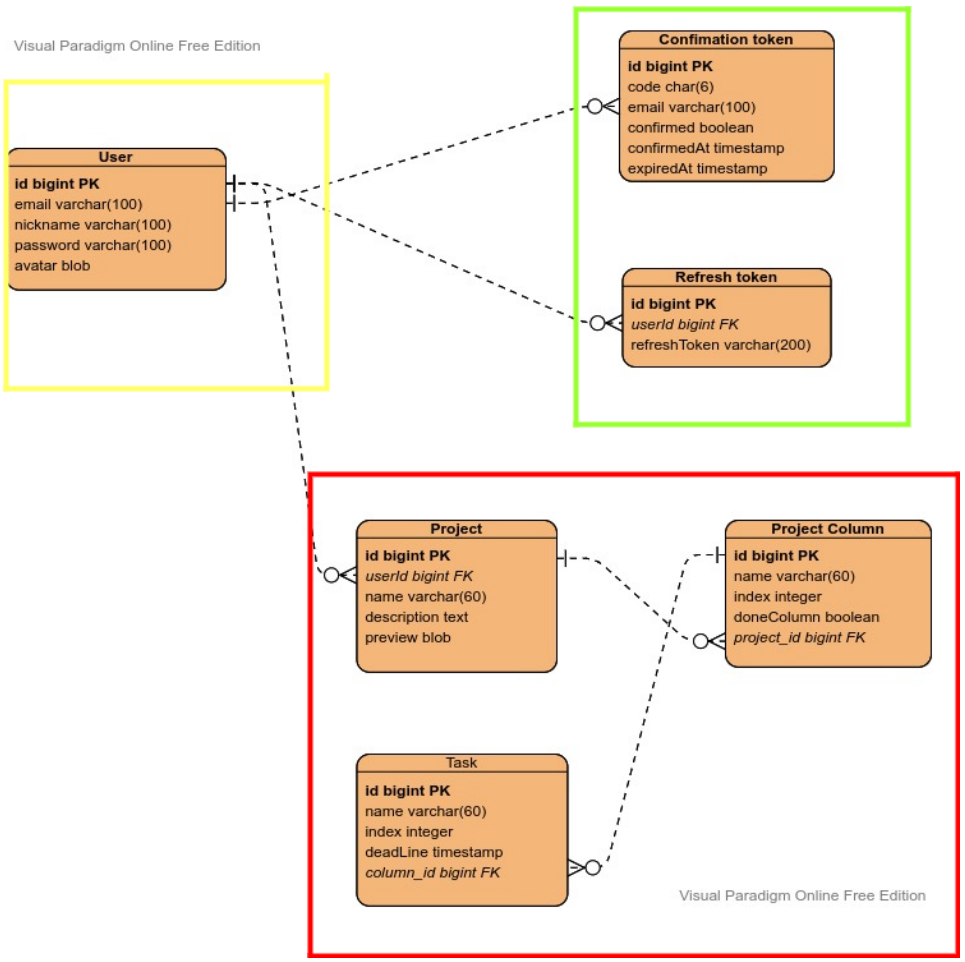

Рисунок 3.8 - ERD діаграма бази даних

Таблиця обведена жовтим кольором належить до “User” мікросервісу. Таблиці обведені зеленим належать до “Auth” мікросервісу. Таблиці обведені червоним належать до “Project” мікросервісу.

3.3 Розробка Frontend частини

Frontend частина построєна на основі Single Page Application (SPA). SPA - це web-додаток, розміщений на одній web-сторінці, яка для забезпечення роботи завантажує весь потрібний код разом із завантаженням самої сторінки. Звичайний сайт складається з безлічі HTML-сторінок. При натисканні на посилання, браузер завантажує нові сторінки за цими посиланнями, у користувача з'являється відчуття руху від однієї сторінки до іншої. Сторінки можуть лежати як файли на якомусь сервері або генеруватися під запит якоюсь серверною програмою. Але, умовно кажучи, кожен екран сайту — це окрема

технічна сутність, окремий документ. SPA працює так: коли користувач відкриває сторінку, браузер завантажує одразу весь код програми. Але показує лише конкретний модуль – частина сайту, яка потрібна користувачеві. Коли користувач переходить в іншу частину програми, браузер бере вже завантажені дані та показує йому. І, якщо потрібно, динамічно підвантажує потрібний контент з сервера без оновлення сторінки. З одного боку, такі програми працюють швидко та менше навантажують сервер. З іншого боку, вони вимагають більшого завантаження на старті. Різниця між SPA та MPA (Multi-Page Application) зображена на рисунку 3.9.

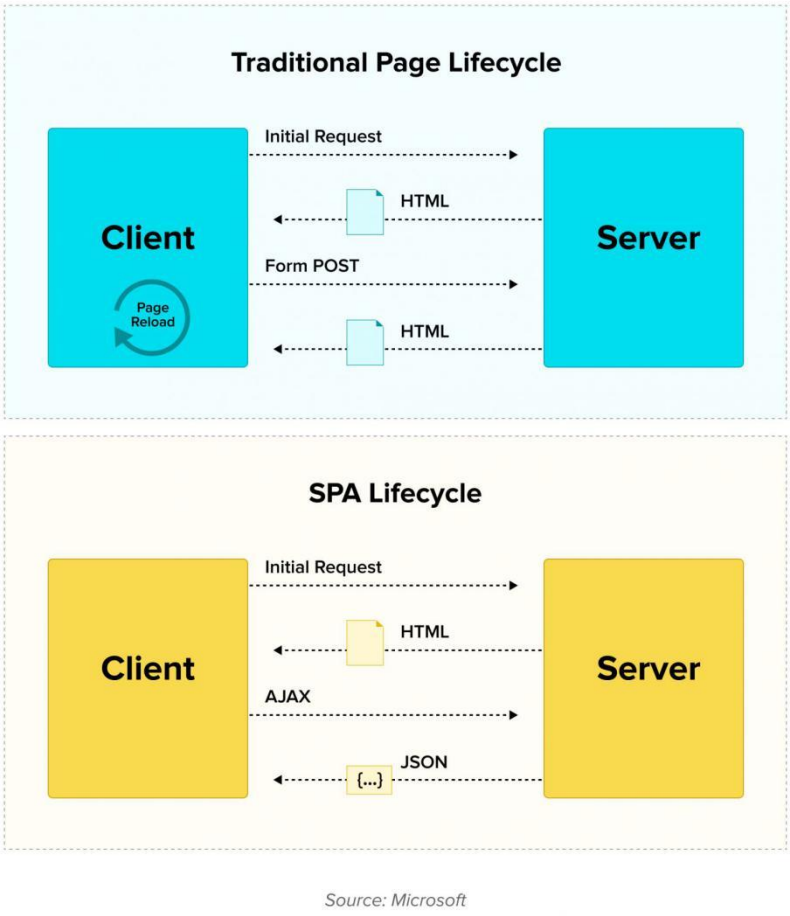

Рисунок 3.9 - Різниця між SPA та MPA

Можна виділити наступні переваги SPA:

– SPA швидкі. Перехід між модулями в веб-додатку відбувається швидше: потрібні ресурси вже завантажені, потрібно просто підставити дані,

які користувач запитав. Часто при цьому сервер повертає не важкий HTML, а легкий JSON або XML. Ще використання JSON спрощує розробку програми для різних платформ. Якщо для веб-версії розробити звичайний сайт, який приймає від сервера HTML, то для мобільного додатку доведеться писати доопрацювання, оскільки HTML не підійде. JSON робить відповідь сервера універсальною.

− SPA гнучкі. Якщо користувач весь час працює з однією сторінкою, простіше робити цікаві переходи та анімацію елементів. Можна працювати зі станом кнопок, вкладок та перемикачів. Таким чином, інтерфейс SPA може бути схожий швидше на повноцінний додаток, а не на простий сайт.

Frontend працює на основі рендерінгу на стороні клієнта (Client Side Rendering), цей підхід має на увазі рендеринг сторінок прямо у браузері за допомогою JavaScript. Вся логіка, отримання даних, шаблонизація та маршрутизація обробляються на клієнті, а не на сервері. Для передачі HTML/CSS файлів та JavaScript скриптів, використовується Nginx. Nginx - це веб-сервер з відкритим кодом, який також виступає як проксі-сервер електронної пошти, зворотний проксі-сервер і балансувальник навантаження. Структура програмного забезпечення є асинхронною та керованою подіями, що дозволяє обробляти багато запитів одночасно. Також Nginx буде використовуватись для проксування на запитів сервер від клієнта. Лістинг головних коду сторінок наведено в додатку А.

3.4 Розгортання веб-додатку

Запуск веб-додатку та налаштування оточення для запуску веб-додатку - це не така і легка справа, особливо для мікросервісної архітектури, у цьому процесі не виключена можливість людської помилки: користувач запустить скрипт два рази, переплутає послідовність або щось не зрозуміє. Тому для рішення цієї проблему було використано Docker. Docker[7] - Docker – це платформа для розробки, доставки та запуску контейнерних програм. Docker дозволяє створювати контейнери, автоматизувати їх запуск та розгортання,

керує життєвим циклом. Він дозволяє запускати безліч контейнерів на одній машині. Контейнери — це спосіб упакувати програму та всі її залежності в єдиний образ. Цей образ запускається в ізольованому середовищі, що не впливає на основну операційну систему. Контейнери дозволяють відокремити додаток від інфраструктури: розробникам не потрібно замислюватися, в якому оточенні працюватиме їхня програма, чи будуть там потрібні налаштування та залежності. Вони просто створюють програму, упаковують всі залежності та налаштування в єдиний образ. Потім цей образ можна запускати на інших системах, не турбуючись, що програма не запуститься. Контейнери дозволяють уникнути проблеми, коли, наприклад, не вистачає якоїсь залежності, тому що вони містять все необхідне для запуску програми. У класичному підході для встановлення програми може знадобитися виконати кілька дій: виконати скрипт, змінити файли налаштувань тощо. Контейнери дозволяють повністю автоматизувати цей процес, тому що включають всі необхідні залежності і порядок виконання дій.

Docker Compose — інструмент для запуску та керування мультиконтейнерними програмами. Він допомагає створити ізольоване середовище (пісочницю), в якому містяться всі необхідні залежності. Конфігурація цієї програми описується у файлі YAML. Запустити програму, раніше зібрану у контейнер, можна однією командою. Docker керує окремими сервісами, з яких складається програма. Docker Compose керує кількома контейнерами, що входять до складу програми. Можливості ті ж, що й Docker. Але можна вибудовувати комплексніші системи взаємодії.

3.5 Реалізований функціонал

Весь запланований функціонал був реалізований, а саме:

− Можливість реєстрації з підтвердженням електронної пошти користувача ;

− Можливість авторизації;

− Можливість створення проектів;

– Реалізація канбан-дошки;

– Можливість відстеження часу.

Реєстрація користувача поділяється на 2 етапи. Перший етап - користувач електронну пошту за допомогою повідомлення з кодом підтвердження (Рисунок 3.10). Тобто користувач вводить свою електронну пошту, система перевіряє цю пошту на предмет існування в системі, якщо ця пошта не існує, то на введену пошту відправляється повідомлення з кодом (Рисунок 3.11), після чого користувач повинен ввести цей код, і після перевірки цього кода системою, користувач може перейти на наступний етап - заповнення реєстраційних даних: аватар, нікнейм, пароль (Рисунок 3.12). Після чого користувач успішно реєструється в системі.

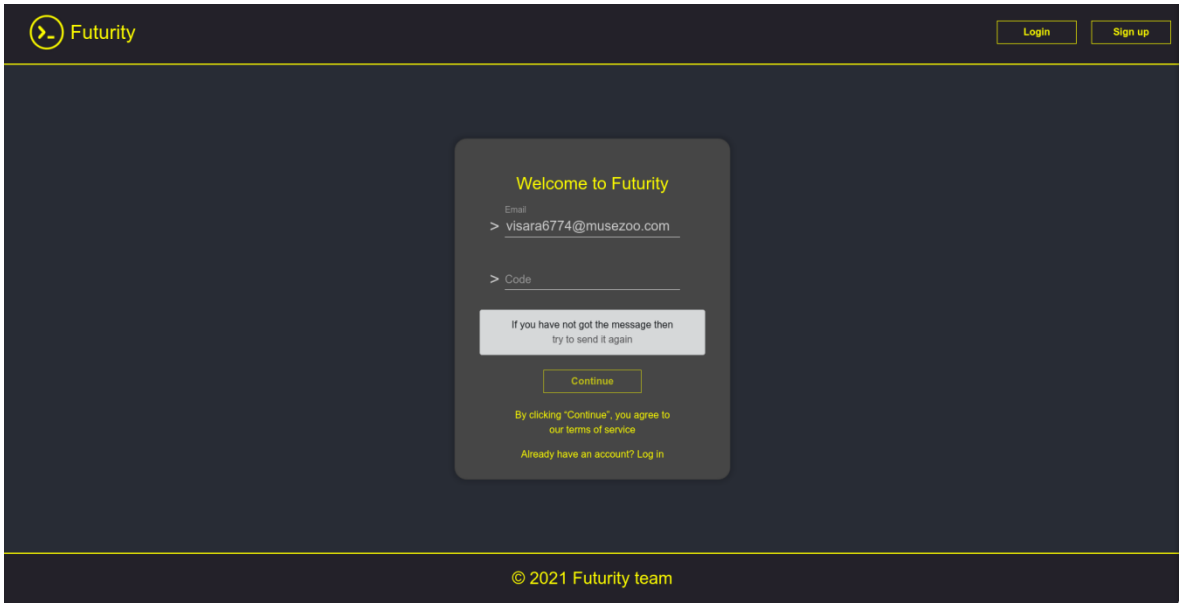

Рисунок 3.10 - Верифікація електронної пошти користувача

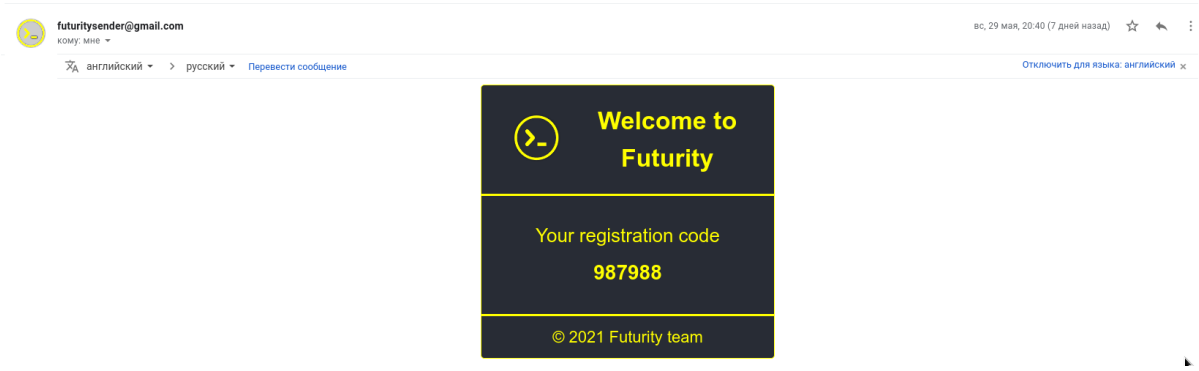

Рисунок 3.11 - Повідомлення з кодом

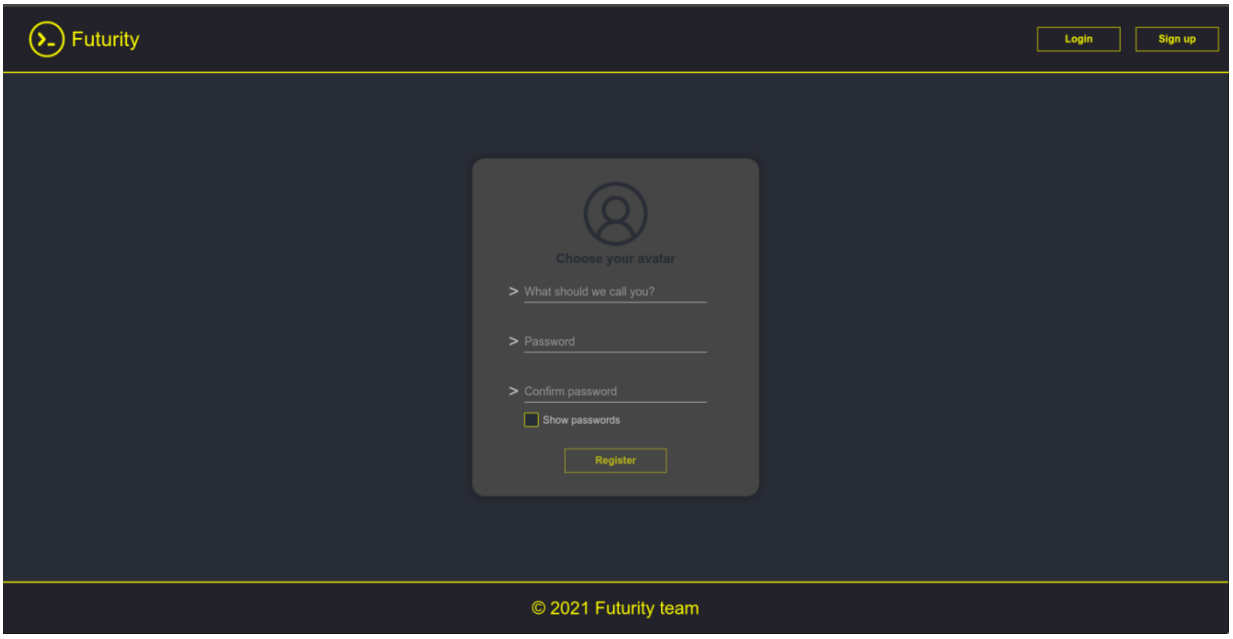

Рисунок 3.12 - Реєстрація даних користувача

Авторизація користувача відбувається звичним чином, а саме через електронну пошту та пароль. Сторінка логіну користувача зображена на рисунку 3.13.

Рисунок 3.13 - Сторінка логіну користувача

Сторінка з проектами має наступний функціонал:

− Відображення вже існуючих проекті;

− Створення нових проектів;

− Видалення існуючих проектів;

− Зміна назви та опису проекта.

Проект складається з таких полів: прев’ю, назва, опис. Функціонал сторінки з проектами зображений на рисунках 3.14-3.15.

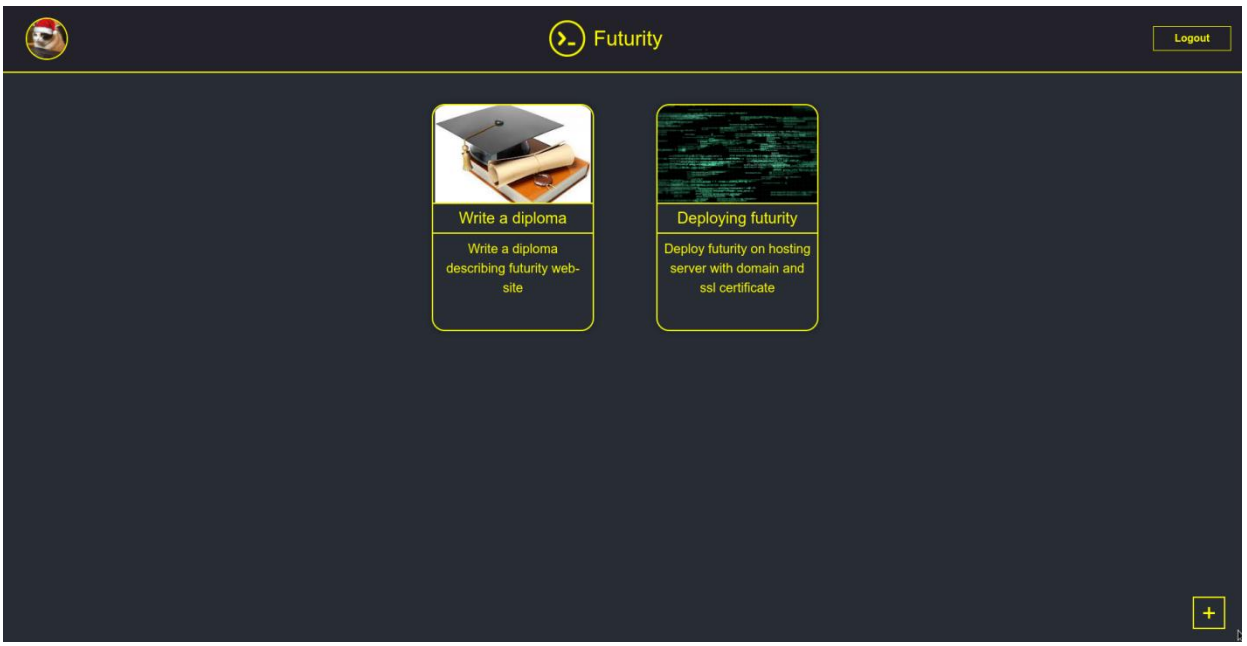

Рисунок 3.14 - Сторінка з проектами

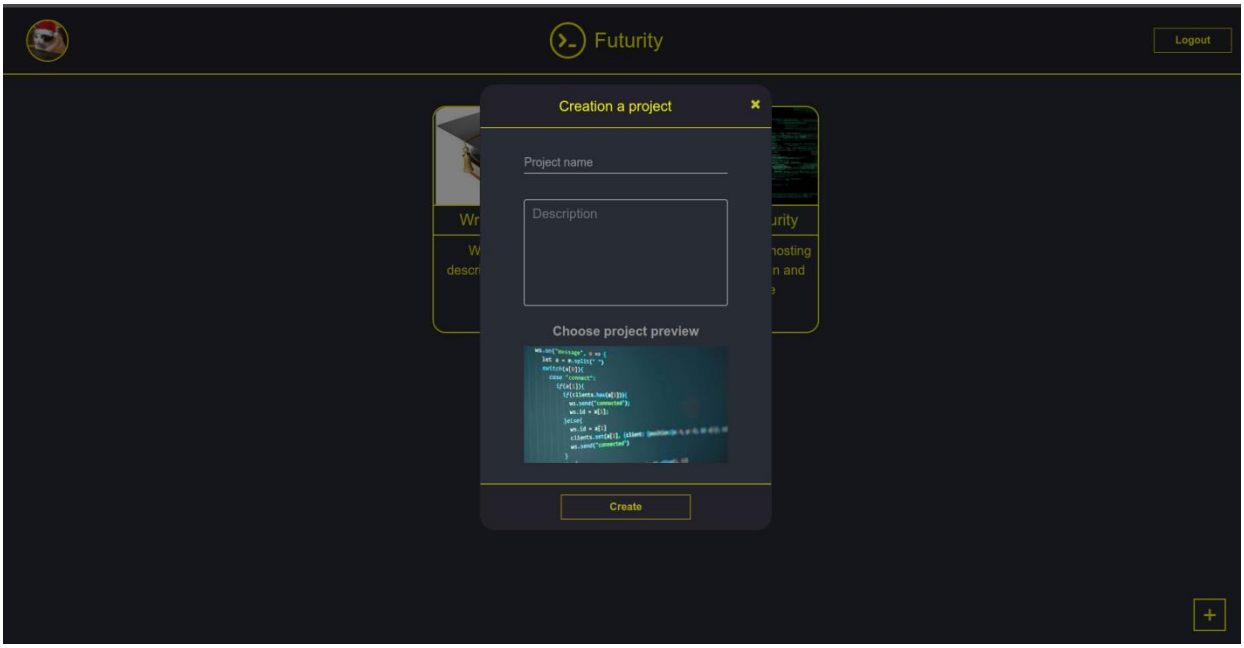

Рисунок 3.15 - Створення нового проекту

Сторінка проекта має вигляд канбан-дошки, так має наступний функціонал:

– Створення нових колонок;

– Створення нових задач;

– Редагування назви колонок та задач;

– Зміна позиції колонок та задач;

– Видалення існуючих колонок та задач;

– Можливість відмітити колонку як “Колонка, яка містить виконані задачі”;

– Можливість встановлення або зміна термінів виконання та відстеження часу.

Сторінка з проектами зображена на рисунку 3.16.

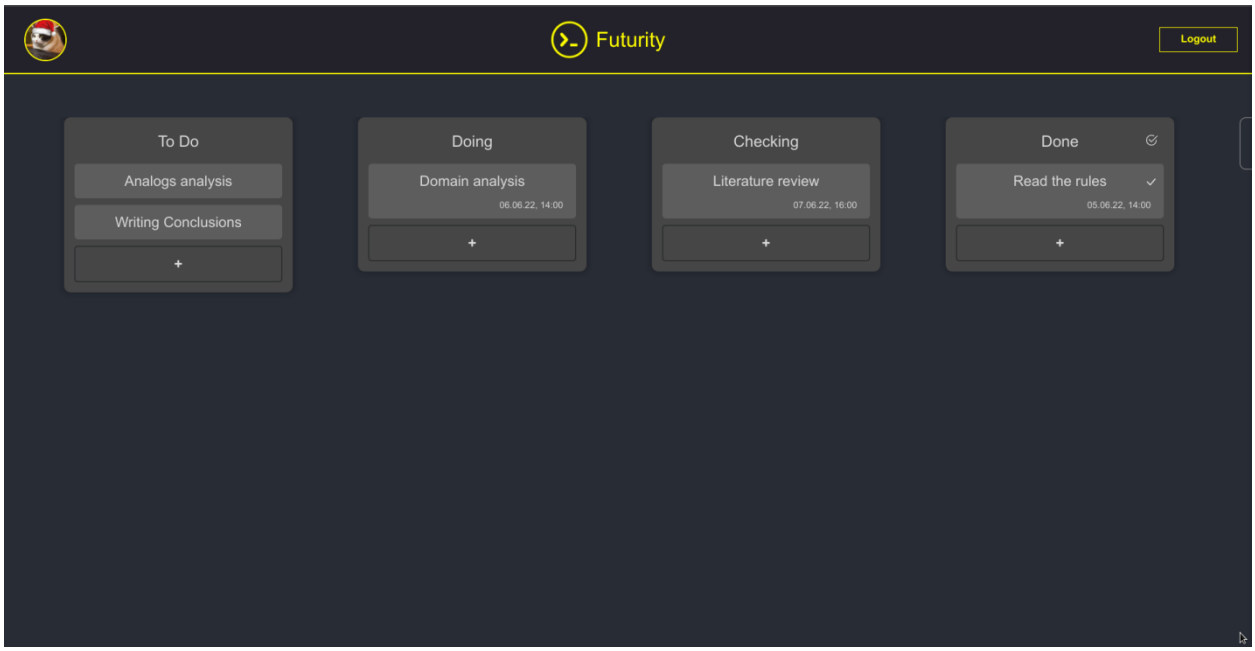

Рисунок 3.16 - Сторінка проекту

Як було сказано, одна з головних функціональних можливостей - це встановлення термнів та відстеження часу. При створення нової задачі користувачу надається можливість встановити термін виконання цієї задачі або створити цю задачу без термінів виконання. Цей функціонал зображено на рисунку 3.17.

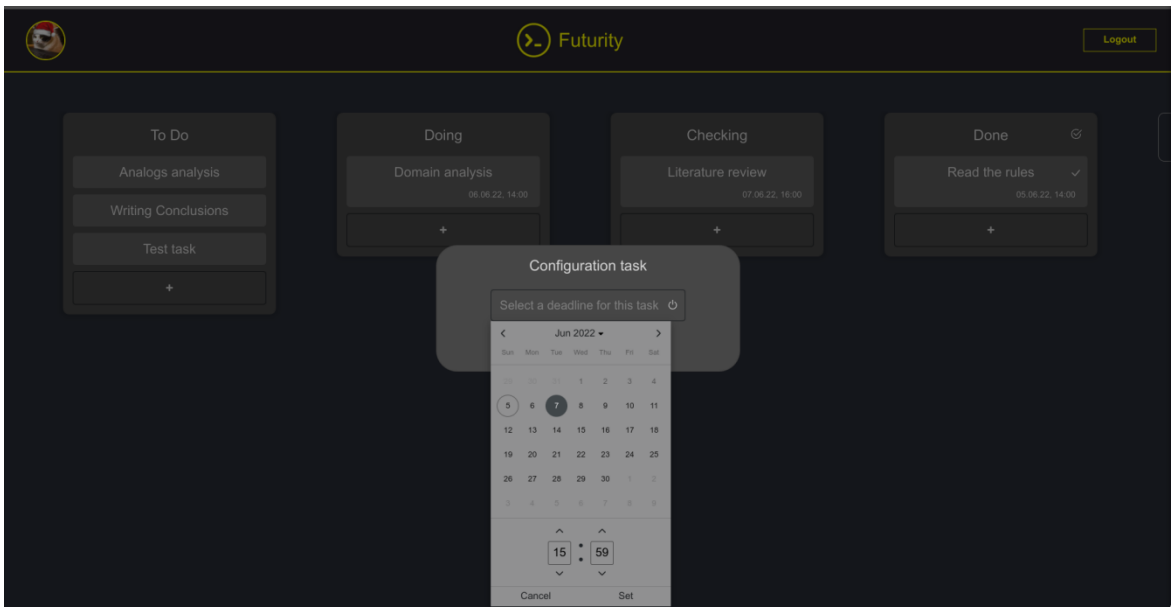

Рисунок 3.17 - Задання термінів виконання задачі

Після створення задачі з терміном (який можна змінити по бажанню), в задачі він буде відображатися і при наведенні на нього буде відображено скільки часу залишилось. Цей функціонал зображено на рисунку 3.18.

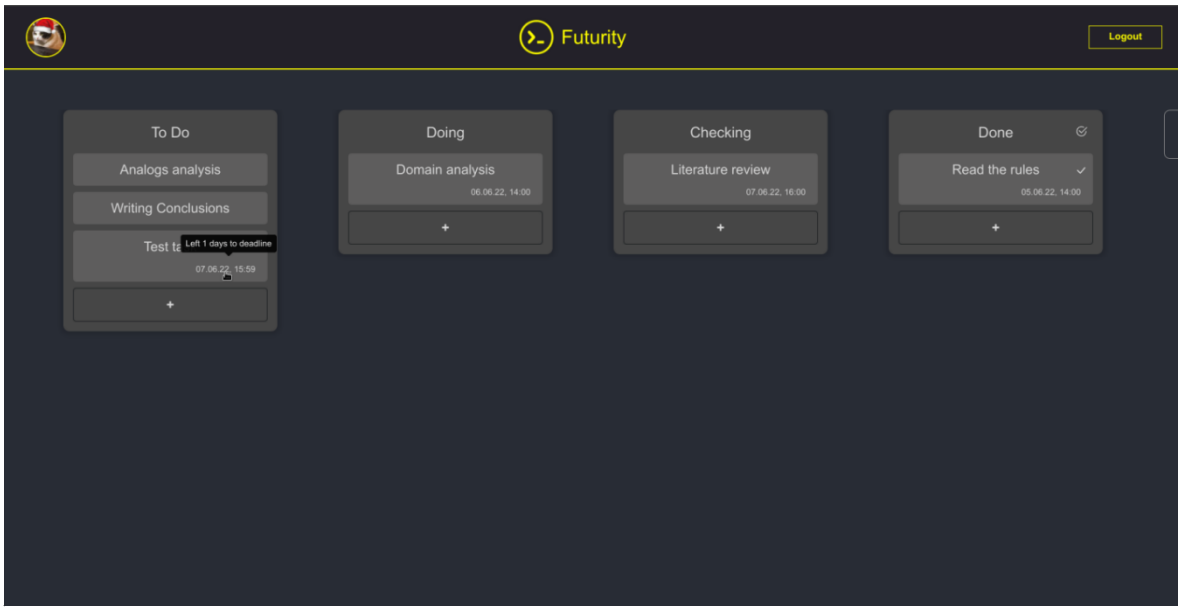

Рисунок 3.18 - Відстеження термінів виконання

Якщо терміни виконання порушені та задача не знаходиться в колонці для виконаних задач, то ця задача виділяється червоним кольором і при наведенні на термін виконання, відображено що термін виконання порушено. Цей функціонал зображено на рисунку 3.19.

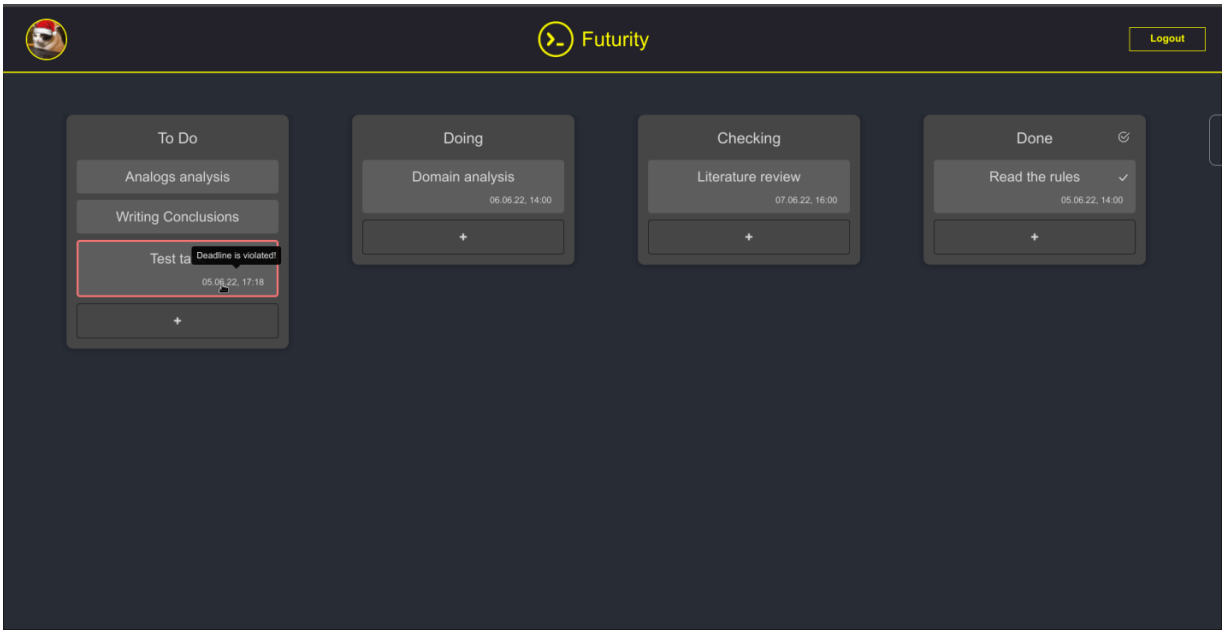

Рисунок 3.19 - Порушення термінів виконання

Якщо задача вважається виконаною, то її треба перенести в “Колонку для виконаних задач”, яку може довільно обрати користувач. На задачу, яку перенесено в колонку для виконаних задач терміни виконання не діють. Цей функціонал зображено на рисунку 3.20.

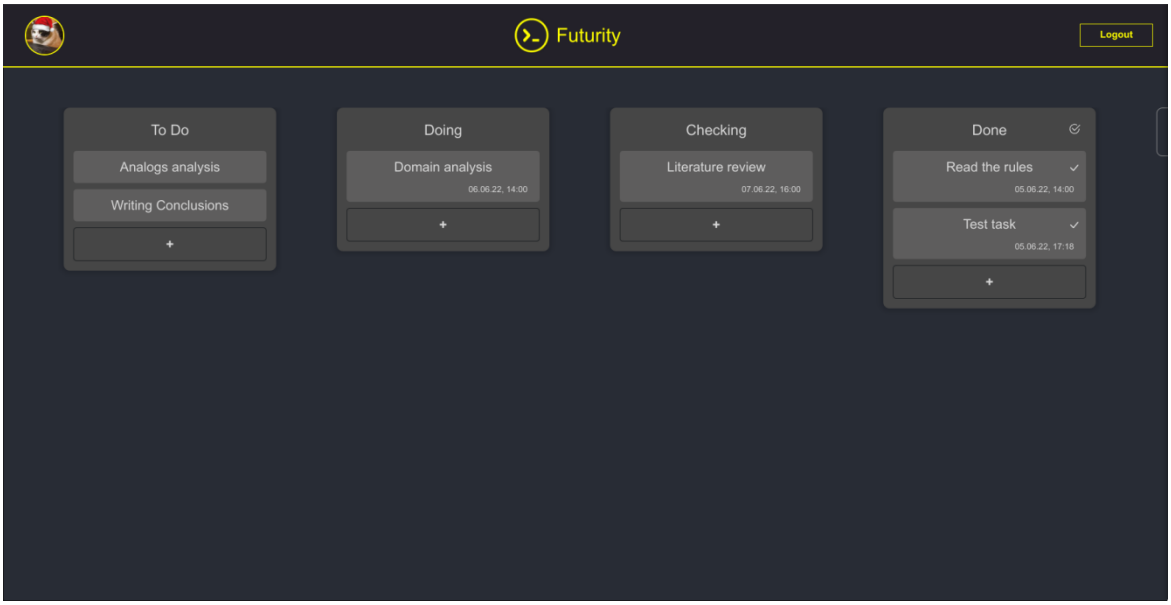

Рисунок 3.20 - Виконання задачі

Такий функціонал повністю імітує канбан-дошку та допомогає візуально організувати задачі, також був створений той функціонал, якого не вистачає аналогічним веб-додаткам, а саме відстеження часу.

Висновки

У результаті виконання кваліфікаційної роботи бакалавра було розроблено інформаційну систему трекінгу виконання індивідуальних навчальних завдань.

Під час реалізації дипломного проекту було проведено аналіз предметної області, та визначено її актуальність.

У проектній частині дипломної роботи обрав необхідні інструменти для практичної реалізації, проаналізував особливості їх застосування. Також було розроблено структуру web-додатку інформаційної системи трекінгу виконання індивідуальних навчальних завдань та спроектовано макет інтерфейсу.

У практичній частині дипломної роботи було розроблено архітектуру web-додатку, спроектовано та розроблено базу даних для підтримки роботи інформаційної системи. В завершенні було розроблено серверну частину web-додатку, для реалізації функціоналу інформаційної системи.

Розроблений web-додаток інформаційної системи трекінгу виконання індивідуальних навчальних завдань допомагає підвищити ефективність освітнього процесу та автоматизувати контроль за поставленими навчальними задачами.

Список літератури

1. Стадії циклу розробки ПЗ [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/.

2. Дошка Kanban [Електронний ресурс] – Режим доступу до ресурсу: https://happymonday.ua/ru/kanban-doska-dlja-raboty-kak-ispolzovat

3. Trello [Електронний ресурс] – Режим доступу до ресурсу: https://trello.com/ru.

4. Кисельов, С.В. Веб-дизайн: Навчальний посібник / С.В. Кисельов. - М .: Академія, 2018. - 416 с.

5. Дакетт, Джон Основи веб-програмування з використанням HTML, XHTML і CSS / Джон Дакетт. - М .: Ексмо, 2019. - 768 с.

6. REST архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://systems.education/what-is-rest

7. Docker documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/

8. Дакетт, Д. HTML і CSS. Розробка і дизайн веб-сайтів / Д. Дакетт. - М .: Ексмо, 2018. - 208 с.

9. Гоше, Хуан Дієго HTML5. Для професіоналів / Гоше Хуан Дієго. - М.:ЕКОМ Паблишерз, 2019. - 149 с.

10. Гарретт Джесс. Веб дизайн. Елементи досвіду взаємодії. - М .: Символ-Плюс, 2020. - 285 с.

11. Еккель, Брюс Філософія Java / Брюс Екель. - М .: Діалектика, 2016. - 809 с.

12. Spring documentation [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects

13. Angular documentation [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs

Додаток А

Лістинг програмного коду основних модулів

AuthController.java

```
package com.alex.futurity.authorizationserver.controller;

import com.alex.futurity.authorizationserver.dto.*;
import com.alex.futurity.authorizationserver.service.AuthService;
import lombok.AllArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;
import javax.validation.Valid;

@RestController
@AllArgsConstructor
public class AuthController {
    private static final String REFRESH_TOKEN_KEY = "refresh_token";

    private final AuthService service;

    @PostMapping("/singup")
    @ResponseStatus(value = HttpStatus.CREATED)
    public void singUp(@RequestPart MultipartFile avatar, @Valid @RequestPart
SingUpRequestDTO user) {
        service.singUp(user, avatar);
    }

    @PostMapping("/login")
    public ResponseEntity<JwtTokenDTO> login(@Valid @RequestBody LoginRequestDTO user,
HttpServletResponse response) {
        JwtRefreshResponseDTO dto = service.login(user);

        Cookie cookie = new Cookie(REFRESH_TOKEN_KEY, dto.getRefreshToken());
        cookie.setHttpOnly(true);
        cookie.setMaxAge(dto.getAge());

        response.addCookie(cookie);

        return ResponseEntity.ok(new JwtTokenDTO(dto.getAccessToken()));
    }
```

```
    @PostMapping("/confirm-code")
    @ResponseStatus(value = HttpStatus.OK)
    public void confirmCode(@Valid @RequestBody ConfirmCodeRequestDTO code) {
        service.confirmCode(code);
    }

    @PostMapping("/confirm-email")
    @ResponseStatus(value = HttpStatus.OK)
    public void confirmEmail(@Valid @RequestBody ConfirmEmailRequestDTO email) {
        service.confirmEmail(email);
    }

    @GetMapping("/refresh-token")
    public ResponseEntity<JwtTokenDTO> refreshToken(@CookieValue(REFRESH_TOKEN_KEY)
String refreshToken) {
        return ResponseEntity.ok(service.refreshToken(new JwtTokenDTO(refreshToken)));
    }
}
```

ColumnController.java

```
package com.alex.futurity.projectserver.controller;

import com.alex.futurity.projectserver.dto.ProjectColumnDto;
import com.alex.futurity.projectserver.dto.RequestStringDto;
import com.alex.futurity.projectserver.service.ColumnService;
import lombok.AllArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@AllArgsConstructor
@Log4j2
public class ColumnController {
    private final ColumnService columnService;

    @PostMapping("/{userId}/column/{projectId}/create")
    @ResponseStatus(HttpStatus.CREATED)
    public long createColumn(@PathVariable long userId, @PathVariable long projectId,
                             @Valid @RequestBody RequestStringDto columnName) {
        log.info("Handling creation column request. User id: {}, project id: {}, name:
{}", userId, projectId, columnName.getValue());
```

```
        return columnService.createColumn(userId, projectId, columnName.getValue());
    }

    @GetMapping("/{userId}/column/{projectId}")
    public List<ProjectColumnDto> getColumns(@PathVariable long userId, @PathVariable
long projectId) {
        log.info("Handling get columns request. User id: {}, project id: {}", userId,
projectId);

        return columnService.getColumns(userId, projectId);
    }

    @DeleteMapping("/{userId}/column/{projectId}/{columnId}/delete/")
    @ResponseStatus(HttpStatus.OK)
    public void deleteColumn(@PathVariable long userId, @PathVariable long projectId,
@PathVariable long columnId) {
        log.info("Handling deleting column request. User id: {}, project id: {}, column
id: {}",
                userId, projectId, columnId);

        columnService.deleteColumn(userId, projectId, columnId);
    }

    @PatchMapping("/{userId}/column/{projectId}/index/change")
    @ResponseStatus(HttpStatus.OK)
    public void changeIndexColumn(@PathVariable long userId, @PathVariable long
projectId,
                                  @RequestParam int from, @RequestParam int to) {
        log.info("Handling changing column index request. User id: {}, project id: {},
from {} to {}", userId, projectId,
                from, to);

        columnService.changeColumnIndex(userId, projectId, from, to);
    }

    @PatchMapping("/{userId}/column/{projectId}/{columnId}/name/")
    @ResponseStatus(HttpStatus.ACCEPTED)
    public void changeColumnName(@PathVariable long userId, @PathVariable long
projectId, @PathVariable long columnId,
                                 @Valid @RequestBody RequestStringDto columnName) {
        log.info("Handling changing column name request. User id: {}, project id: {},
column id: {}, columnName: {}",
                userId, projectId, columnId, columnName.getValue());

        columnService.changeColumnName(userId,          projectId,          columnId,
columnName.getValue());
    }
```

```
    @PatchMapping("/{userId}/column/{projectId}/mark")
    @ResponseStatus(HttpStatus.ACCEPTED)
    public void markColumnAsDone(@PathVariable long userId, @PathVariable long projectId,
                                 @RequestParam(value = "columnIdToUnmark", required = false) Long columnToUnmark,
                                 @RequestParam(value = "columnIdToMark") long columnToMark) {
        log.info("Handlong marking column request. User id: {}, project id: {}, column to mark: {}, column to unmark: {}",
                userId, projectId, columnToMark, columnToUnmark);

        columnService.markColumnAsDone(userId, projectId, columnToMark, columnToUnmark);
    }
}
```

UserService.java

```
package com.alex.futurity.userserver.service.impl;

import com.alex.futurity.userserver.entity.User;
import com.alex.futurity.userserver.exception.ClientSideException;
import com.alex.futurity.userserver.repo.UserRepository;
import com.alex.futurity.userserver.service.UserService;
import lombok.AllArgsConstructor;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepo;

    @Override
    public boolean isUserExist(String email) {
        return userRepo.findByEmail(email).isPresent();
    }

    @Override
    public void saveUser(User user) {
        userRepo.save(user);
    }
```

```
    @Override
    public Optional<User> findUserByEmail(String email) {
        return userRepo.findByEmail(email);
    }

    @Override
    public Resource findUserAvatar(long id) {
        User user = userRepo.findById(id)
                .orElseThrow(() -> new ClientSideException(String.format("User with id
\"%s\" is not found", id), HttpStatus.NOT_FOUND));

        return new ByteArrayResource(user.getAvatar());
    }
}
```

AuthService.java

```
package com.alex.futurity.userserver.service.impl;

import com.alex.futurity.userserver.dto.LoginRequestDTO;
import com.alex.futurity.userserver.dto.SingUpRequestDTO;
import com.alex.futurity.userserver.dto.LoginResponseDTO;
import com.alex.futurity.userserver.dto.UserExistRequestDTO;
import com.alex.futurity.userserver.entity.User;
import com.alex.futurity.userserver.exception.CannotUploadFileException;
import com.alex.futurity.userserver.exception.ClientSideException;
import com.alex.futurity.userserver.service.AuthService;
import com.alex.futurity.userserver.service.UserService;
import lombok.AllArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.io.IOException;

@Service
@AllArgsConstructor
@Log4j2
public class AuthServiceImpl implements AuthService {
    private final UserService userService;
    private final PasswordEncoder encoder;

    @Override
    @Transactional
    public User singUp(SingUpRequestDTO request) {
```

```
        try {
            if (userService.isUserExist(request.getEmail())) {
                throw new ClientSideException("Error. A user with the same email address
already exists", HttpStatus.CONFLICT);
            }
            User user = request.toUser();
            user.setPassword(encoder.encode(user.getPassword()));

            userService.saveUser(user);

            log.info("User {} have been registered", user);
            return user;
        } catch (IOException e) {
            log.warn("The avatar {} cannot be read: {}", request.getAvatar(),
e.getMessage());
            throw new CannotUploadFileException("The avatar cannot be read");
        }
    }

    @Override
    @Transactional
    public LoginResponseDTO login(LoginRequestDTO request) {
        User user = userService.findUserByEmail(request.getEmail())
                .orElseThrow(() -> new ClientSideException("Email or password is
incorrect. Check the entered data", HttpStatus.NOT_FOUND));

        if (!encoder.matches(request.getPassword(), user.getPassword())) {
            throw new ClientSideException("Email or password is incorrect. Check the
entered data", HttpStatus.NOT_FOUND);
        }

        log.info("The user with email {} was login in", request.getEmail());
        return new LoginResponseDTO(user);
    }

    @Override
    @Transactional
    public boolean isUserExist(UserExistRequestDTO request) {
        return userService.findUserByEmail(request.getEmail()).isPresent();
    }
}
```

Kanban-page.component.html

```
<div class="position-absolute h-100 w-100 container-absolute">
  <div cdkDropList cdkDropListGroup cdkDropListOrientation="horizontal"
       [cdkDropListData]="columns" (cdkDropListDropped)="dropColumn($event)"
```

```
       class="container-fluid d-flex align-items-start body-padding"
       [cdkDropListAutoScrollStep]="10" cdkScrollable #container dragScroll [drag-
scroll-disable]="['column']">
    <div class="spinner-border mx-auto" role="status" *ngIf="!columnsAreLoaded"></div>
    <div class="column text-center p-3" *ngFor="let column of columns; let i = index"
cdkDrag cdkScrollable
         [contextMenu]="columnMenu" [contextMenuSubject]="column"
    >
      <div class="d-flex flex-row-reverse justify-content-center">
        <i-bs  *ngIf="changingColumnIndex  !=  i  &&  column.isDone"  width="20px"
height="20px"
              class="cursor-pointer padding-right" name="check2-circle"
              [ngbTooltip]="'Move a task to this column to mark it as \'done\'"></i-
bs>
        <h4 class="mt-2 mb-2 mx-auto text-nowrap overflow-hidden text-truncate cursor-
pointer"
            (click)="startChangeColumnName(i)" *ngIf="changingColumnIndex !== i"
            [ngClass]="{
              'px-3': column.name.length > COLUMN_NAME_LENHGT,
              'padding-left-29': column.isDone
            }"
            [ngbTooltip]="column.name.length > COLUMN_NAME_LENHGT ? column.name : null">
          {{column.name}}
        </h4>
      </div>
      <input  type="text"  class="mt-2  mb-2  box-input"  [value]="column.name"
*ngIf="changingColumnIndex === i"
             (blur)="changeColumnName($event, column)"
             (keyup.enter)="changeColumnName($event, column)"
             (keyup.escape)="abortColumnChanging()"
             #input_column_change>
      <div      class="list"      cdkDropList      [cdkDropListData]="column.tasks"
(cdkDropListDropped)="dropTask($event, i)"
           cdkScrollable [cdkDropListAutoScrollStep]="10"
           [ngStyle]="{'max-height': (container.offsetHeight - 150) + 'px'}">
        <div class="box d-flex flex-column" *ngFor="let task of column.tasks; let ti =
index;" cdkDrag
             [contextMenu]="taskMenu"
             [contextMenuSubject]="task" [ngStyle]="{
              'border': this.isDeadlineViolated(task.deadline, column.isDone) ? '3px
solid rgb(248, 118, 118)' : 'rgb(94, 93, 93)'
             }">
          <div class="d-flex flex-row-reverse justify-content-center w-100">
              <i-bs  *ngIf="changingTaskIndex.taskIndex  !=  ti  &&  column.isDone"
class="cursor-pointer" name="check-lg"
                    [ngbTooltip]="this.TASK_IS_DONE" width="20px" height="20px"></i-bs>
```

```
            <div class="text-nowrap overflow-hidden text-truncate cursor-pointer mx-
auto"
                 *ngIf="i    !==    changingTaskIndex.columnIndex    ||    ti    !==
changingTaskIndex.taskIndex"
                 (click)="startChangeTaskName({columnIndex: i, taskIndex: ti})"
                 [ngClass]="{
                    'px-3': task.name.length > TASK_NAME_LENHGT,
                    'padding-left': column.isDone
                 }"
                 [ngbTooltip]="task.name.length > TASK_NAME_LENHGT ? task.name : null"
            >
              {{task.name}}
            </div>
          </div>
          <input type="text" class="box-input p-0" [value]="task.name" *ngIf="i ===
changingTaskIndex.columnIndex
           && ti === changingTaskIndex.taskIndex"
                 (blur)="changeTaskName($event, i, ti)"
                 (keyup.enter)="changeTaskName($event, i, ti)"
                 (keyup.escape)="abortTaskChanging()"
                 #input_task_change>
          <div class="deadline-text cursor-pointer" *ngIf="task.deadline != null"
               [ngbTooltip]="diffDeadline(task.deadline, column.isDone)"
          >
            {{showDeadline(task.deadline)}}
          </div>
        </div>
      </div>
      <div class="box" *ngIf="creationNewTask && i === creationColumnIndex">
        <input class="box-input" type="text"
               (blur)="creationTaskUnFocus($event)"
               (keyup.enter)="creationTaskUnFocus($event)"
               (keyup.escape)="abortTaskCreation()" #input_box>
      </div>
      <div class="box add" (click)="startCreateTask(i)"
           *ngIf="!creationNewTask || i !== creationColumnIndex"
      >+
      </div>
    </div>
    <div class="column text-center p-3" *ngIf="creationNewColumn">
      <input type="text" class="mt-2 mb-2 box-input"
             (blur)="creationColumnUnFocus($event)"
             (keyup.enter)="creationColumnUnFocus($event)"
             (keyup.escape)="abortColumnCreation()"
             #input_column>
    </div>
    <div class="column text-center p-3 column-add" (click)="startCreateColumn()"
```

```
        *ngIf="!creationNewColumn && columnsAreLoaded">
      <h2 class="mt-1 mb-1">+</h2>
    </div>
  </div>
</div>
<context-menu #columnMenu>
  <ng-template contextMenuItem let-item (execute)="deleteColumn($event.item)">Delete
{{item.name}}</ng-template>
  <ng-template contextMenuItem (execute)="markColumn($event.item)">Mark column as
'Done'</ng-template>
</context-menu>
<context-menu #taskMenu>
  <ng-template contextMenuItem (execute)="changeTaskDeadline($event.item)">Change
deadline</ng-template>
  <ng-template contextMenuItem let-item (execute)="deleteTask($event.item)">Delete
{{item.name}}</ng-template>
</context-menu>
```

Projects-page.component.html

```
<div class="container-fluid px-5 d-flex align-items-center justify-content-center mt-5
flex-wrap">
  <div class="card mx-5 mb-5" *ngFor="let project of projects; let i = index;"
[contextMenu]="basicMenu"
       [contextMenuSubject]="project"
       (click)="navigateToProject(project.project)"
  >
    <div class="card-img-top d-flex align-items-center justify-content-center">
      <img              class="preview"              [src]="project.project.previewUrl"
*ngIf="project.isPreviewLoad">
      <div           class="spinner-grow           text-warning"           role="status"
*ngIf="!project.isPreviewLoad"></div>
    </div>
    <div class="card-body p-0">
      <div class="card-title text-center w-100 py-1 mb-0">
        <div    class="text-nowrap    overflow-hidden    text-truncate    cursor-pointer"
*ngIf="changingProjectNameIndex !== i"
             [ngClass]="{'px-3': project.project.name.length > DESCRIBED_TEXT_LENGTH}"
             [ngbTooltip]="project.project.name.length   >   DESCRIBED_TEXT_LENGTH   ?
project.project.name : null"
        >
          {{project.project.name}}
        </div>
        <input     type="text"     class="box-input"     [value]="project.project.name"
*ngIf="changingProjectNameIndex === i"
               (blur)="changeProjectName($event, i)"
               (keyup.enter)="changeProjectName($event, i)"
               (keyup.escape)="abortChangingProjectName()"
```

```
                #input_project_name_change>
      </div>
      <div class="text-center p-2 overflow-auto h-75">
        <div    class="card-text-font"    *ngIf="changingProjectDescriptionIndex    !==
i">{{project.project.description}}</div>
        <textarea type="text" class="box-area" [value]="project.project.description"
                  *ngIf="changingProjectDescriptionIndex === i"
                  (blur)="changeProjectDescription($event, i)"
                  (keyup.enter)="changeProjectDescription($event, i)"
                  (keyup.escape)="abortChangingProjectDescription()"
                  #input_project_description_change>
        </textarea>
      </div>
    </div>
  </div>
  <div class="spinner-border" role="status" *ngIf="!projectsAreLoad"></div>

  <div *ngIf="projects.length === 0 && projectsAreLoad">
    <div class="text mt-3">You don't have any projects.
      <a  class="cursor-pointer"  (click)="openCreationModal()">Do  you  want  to  create
one?</a>
    </div>
  </div>
</div>

<button class="plus-button d-flex align-items-center justify-content-center position-
fixed cursor-pointer"
        (click)="openCreationModal()"
>
  <span class="plus-sign">+</span>
</button>
<context-menu>
  <ng-template    contextMenuItem    (execute)="deleteProject($event.item)">Delete</ng-
template>
  <ng-template contextMenuItem (execute)="startChangingProjectName($event.item)">Change
name</ng-template>
  <ng-template                                                          contextMenuItem
(execute)="startChangingProjectDescription($event.item)">Change       description</ng-
template>
</context-menu>
```