НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Комп'ютерних наук
(назва кафедри)

_____ Голуб Б.Л.
(підпис) (ПІБ)

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Інформаційна система керування проєктами
Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
доктор економічних наук, професор _____ _____Руденський Р.А._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
кандидат фізико-математичних наук, доцент _____ ____Кириченко В.В.____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Чурілов І.В._____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____к.т.н., доцент Голуб Б.Л_____

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)

"____" _____20_____р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту

_____Чурілову Іллі Вячеславовичу_____
(прізвище, ім'я, по батькові)

Спеціальність 122 – «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи Інформаційна система керування проєктами
затверджена наказом ректора НУБіП України від " 01 "травня 2025 р. №644 «С»
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:
Це розробка інформаційної системи курування проектами, вивчення технологій веб-розробки та основ управління проектами, а також збір даних для функціонального аналізу, включаючи потреби користувачів та можливості управління проектами, ролями, завданнями та робочими процесами. Ці компоненти сприятимуть створенню потужної та функціональної системи, що дозволить ефективно управляти проектами та завданнями всередині них.
Перелік питань, які потрібно розробити:
1. аналіз предметної області;
2. проєктування системи для керування проектами;
3. реалізація програмного забезпечення платформи створення веб-сайту;
4. впровадження системи;

Дата видачі завдання "_____"_____2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ ___Кириченко В.В____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ ____Чурілов І.В.___
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ВСТУП

У наш час, коли цифрові технології активно входять у всі сфери діяльності, і ефективне керування проєктами стає надзвичайно важливим для успішного функціонування компаній, організацій і стартапів, через постійне зростання попиту на ринку програмних продуктів, зростає і кількість проєктів із різними рівнями складності і виникає потреба у надійних та гнучких інструментах, що дозволять здійснити планування, узгодження дій та контроль за виконанням завдань.

Об’єктом дослідження дипломної роботи є процес управління проєктами в компаніях, в умовах використання сучасних інформаційних технологій.

Предметом дослідження є методи та інформаційні засоби та інструменти веб-розробки та серверної архітектури, що надають можливість для створення інформаційних систем курування проєктами.

Метою дипломної роботи – є розробка системи з мінімалістичним, простим та зрозумілим інтерфейсом який відповідає вимогам сьогодення, що дасть змогу користувачам виконувати створення нових проєктів та виконувати їх управління. До цього відноситься можливість розподіляти ролі серед працівників та колег, легко формулювати завдання та відстежувати їх процес виконання, для реалізації задачі, поставленої мети. Буде розгорнуто план проведення детального аналізу сучасних інструментів і технологій веб-розробки, а також підходів до створення інформаційної системи у сфері управління проєктами.

Основну увагу в дипломній роботі буде приділено вибору оптимальної структури системи. Не менш важливим є вибір мови програмування, пошуку шляхів зберігання й обробки даних, а також можливості інтеграції з зовнішніми сервісами.

До основних функцій розробленої системи відноситься можливість створення проєктів, гнучке керування ролями учасників, формування та контроль завдань, відстеження їхнього виконання і прогресу. Пріоритет буде зрушено в бік зручності у користуванні, підтримці до масштабування та можливість для подальшого розширення функціоналу під потреби юзерів.

Отже, оцінка бажаної ефективності системи вираховуватиметься на основі перевірки її функціональних можливостей, аналізу продуктивності, швидкодії та порівнння з вже існуючими рішеннями в цій галузі. В результаті очікується, що після аналізу та розробки буде створено високопродуктивний, гнучкий та зручний у використанні веб додаток з великим потенціалом для розширення та масштабування.

В дипломному проекті всі отримані результати сприятимуть та покращать наявні цифрові рішення для управління проєктами та відкриють нові перспективи для організацій і команд, які хочуть збільшити ефективність роботи в розробці за допомогою сучасних IT-рішень.

1. Системний аналіз предметної області

Сьогодні веб-простір охоплює безліч сайтів, відкритих для широкого кола користувачів. Кожен з них має індивідуальну будову, що формується завдяки тематиці, технологічній основі та зовнішньому вигляду. Структура сайтів варіюється від простих інформаційних сторінок до складних, динамічних платформ з інтерактивними елементами.

Веб-сторінки створюються за допомогою різних мов програмування, наприклад серед яких є HTML, PHP, JavaScript та TypeScript, інші. Вони дозволяють управляти вмістом сайту та надавати можливість виконувати взаємодію юзера з елементами інтерфейсу, до прикладу, такими як мультимедійні файли — відео, зображення чи аудіо.

Інструменти сьогодення, такі ось як наприклад React і ASP.NET, дають можливості для розробки інтерактивних платформ. Наприклад оновлення даних без необхідності в перезавантаженні сторінки. Завдяки цій особливості користувач отримує більш плавний, гнучкий і зручний досвід взаємодії. На відміну від традиційного програмного забезпечення що встановлено на пристроях, веб-додатки найчастіше виконуються без компіляції. Код інтерпретується безпосередньо браузером або сервером. Це забезпечує процес ефективного обміну даними із залученням запита клієнта та відповіді від сервера. HTML слугує основою структури веб-сторінки. CSS виконує роль стилістичного оформлення. JavaScript (TypeScript) виконує взаємодію користувач із сторінкою. React дає змогу створювати гнучкі застосунки SPA, які швидко реагують на дії користувача, тоді як ASP.NET застосовується для побудови логіки на стороні сервера.

1.1 Аналіз вимог до програмної системи

Аналіз вимог до інформаційної системи курування проєктами є ключовим етапом у процесі її розробки. Вимоги визначають її функціональні і нефункціональні можливості. Це необхідно враховувати при дослідженні розробки, реалізації та впровадженні. Вони складається з аналізу, можливостей розробки, документування, передбачення дій майбутніх користувачів, бізнес-цілей та технічних обмежень. Ретельне аналізування дає змогу чітко уявити систему, її функціональність, забезпечити її відповідність до очікувань користувачів, а також зменшити непорозуміння у процесі користуванням.

Функціональні вимоги визначають, які конкретні задачі повинна виконувати система курування проєктами:

- Створення та управління проєктами

Система має мати можливість створювати нові проєкти за допомогою кнопки «Створити проєкт», після натиснення на яку, буде відкриватися зручна форма на якій можна ввести основну інформацію, таку як: назва, опис, дедлайн, цілей тощо. Всі створені проєкти, будуть відображатись у вигляді списку, і користувач зможе відкривати їх.

- Призначення ролей учасникам проєкту

В блоці «Учасники» адміністратор проєкту повинен мати спроможність додавати нових учасників, вводячи їх ID, після цього він зможе обрати запропоновану роль зі випадаючого списку доступних. В середині системи передбачаються такі ролі: менеджер, виконавець, переглядач, адміністратор. Визначені ролі мають надавати чіткий розподіл обов’язків між учасниками. Це має на меті обмежити доступ до конкретних функцій системи, для певних груп користувачів.

- Управління завданнями

У рамках кожного проєкту має бути створено область, в якій користувачі зможуть створювати нові задачі, натискаючи на кнопку «Додати завдання». У цій формі для створення завдання, треба буде вказати всі необхідні дані а саме: назву, опис, виконавця, дедлайн, пріоритет, статус. Всі завдання можна буде змінювати в будь який момент та видаляти. Також має бути реалізована можливість переміщати між виконавцями на канбан-дошці для візуального контролю процесу виконання завдань.

- Визначення тегів і категоризація завдань

Користувачі матимуть змогу створювати й призначати теги у процесі створення чи редагування завдань. Система повинна мати можливість реалізовувати кастомні теги із кольоровим маркуванням. Це має спростити навігацію та групування завдань. Додатково, буде створено окремий сектор для управління тегами на рівні всього проєкту.

- Фільтрація завдань

Інтерфейс має включати фільтрацію завдань. Це дасть змогу юзерам відобразити всі необхідні завдання за статусом, виконавцем, пріоритетом, тегамом або дедлайном. Обрані фільтри застосовуватимуться після натискання на кнопку. Це має оновити список завдань без потреби в перезавантаженні сторінки. Має бути передбачено виконання сортування при переході через посилання. Це значно полегшить роботу з великим обсягом задач і допоможе відображати лише ті які потрібні користувачу.

- Візуалізація завдань у вигляді таблиці або канбан-дошки

Користувачі матимуть можливість переглядати інформацію у вигляді канбан-дошки. У цій таблиці завдання виводитимуться у вигляді карток. Можна буде перетягувати їх між колонками зі статусами. Це дасть можливість простіше

сприймати інформацію що забезпечить гнучкість у роботі з задачами та підвищить зручність керування проєктами.

Нефункціональні вимоги описують загальні характеристики системи, які не стосуються конкретних функцій системи, але в той самий час мають значущість для її стабільної роботи, для продуктивності, безпеки та зручності у користуванні:

● Продуктивність.

Система має бути спроектовна таким чином щоб працювати стабільно та швидко. За умов помірного навантаження та великої кількості одночасних запитів, відповідь має бути надіслана впродовж 250–560 мс у 96% випадків. Максимальною кількістю одночасних з'єднань, яку система має витримати, становить від 10 000 користувачів.

● Безпека.

Важливо реалізувати просту і в той самий час надійну систему що буде забезпечувати збереженню конфіденційності даних користувачів. Система має включати обробку особистої інформації. Повинна підтримувати механізми шифрування паролей, використовувати різні токени, автентифікацію і авторизацію.

● Сумісність.

Весь інтерфейс системи має правильно працювати в першу чергу на ПК в відомих браузерах (Brave, Edge, Chrome, Explorer) з версіями понад 2016 року релізу, де має бути забезпечено зручний доступ для користувачів.

1.2 Моделювання предметної області

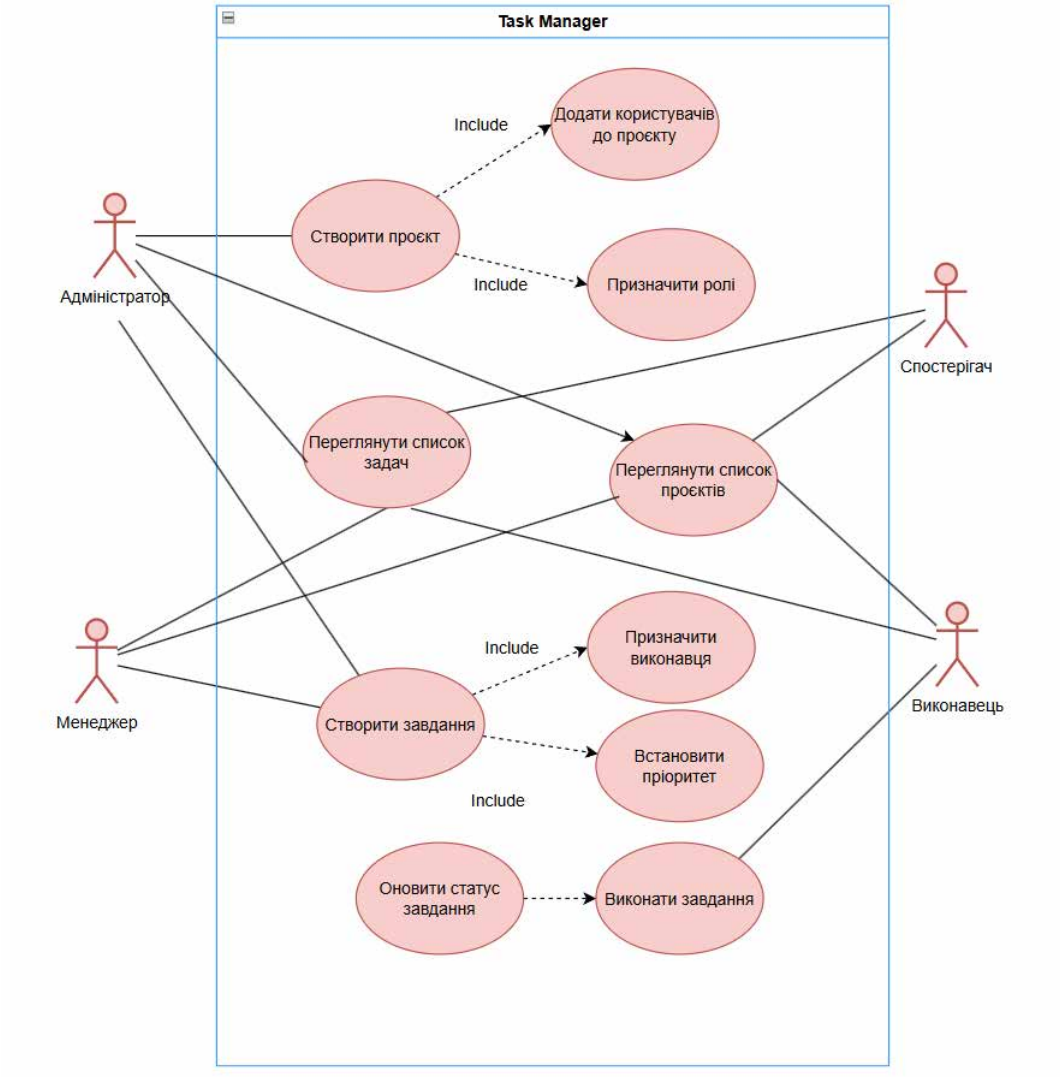
UML (Universal Modeling Language) – універсальна мова моделювання, який був розроблений компанією Rational Software з метою створення найбільш оптимального та універсальної мови для опису як предметної області, так і конкретного завдання в програмуванні. Візуальне моделювання в UML можна уявити як певний процес порівневого спуску від найбільш загальної і абстрактної концептуальної моделі системи до логічної, а потім і до фізичної моделі відповідної системи. [1]

На етапі моделювання предметної області було створено діаграми такі як:

- Діаграма прецедентів показує різні варіанти використання та різні типи користувачів системи і часто супроводжується іншими типами діаграм. Варіанти використання представлені колами або еліпсами. Актори (дійові особи) часто зображуються у вигляді паличок. [5]
- Діаграма активності — це графічне уявлення, яке демонструє чітку послідовність всіх дій і рішень з альтернативними шляхами, які реалізуються в системі. Вона застосовується при плануванні бізнес логіки або операцій, включаючи моделювання послідовних та паралельних кроків. Така діаграма дає змогу наочно продемонструвати, всі дії що виконуються для досягнення конкретного результату, та які вибори або умови які впливають на процеси.
- Діаграма послідовності — це інструмент моделювання, який ілюструє взаємодію об'єктів у системі через обмін повідомленнями у певній часовій послідовності. На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення. [6]

Діаграма прецедентів, активності та послідовності, мають значущу роль у процесі розробки моделі програмного продукту, вони дають змогу більш детальніше відобразити функціональні спроможності. Логіку виконання процесів всередині програми і взаємодію між різними її компонентам. Кожна з цих трьох діаграм відображає важливі аспекти програмної розробки з різних боків. А в купі складають цілісну картину представлення про її структуру, функціональність та поведінку.

На діаграмі 1.2.1 зображено діаграму прецедентів.



Діаграма 1.2.1 Прецедентів

В системі "Task Manager, є виконавці з різними ролями, адміністратор, менеджер, виконавець і спостерігач: діаграма показує, що може робити кожен із користувачів і які операцій вони можуть виконувати.

Адміністратор — це роль яка є найголовнішою в проекті, що має доступ до абсолютно всіх дозволів. Вона може виконувати створення нових проєктів, додавати до них працівників і призначати їм інші ролі, окрім цього, він має змогу переглядати список усіх проєктів і завдань у системі та виконувати операції над ними, здійснюючи повний контроль проєкту.

Менеджер має дещо менший перелік функцій, а ніж Адміністратор, тобто він керує лише на рівні завдань. Має можливість створювати нові завдання та призначати до них виконавців, і встановлювати потрібний пріоритет і визначати до якого часу треба виконати. Менеджер також може змінити статуси завдань — до прикладу помітити як виконане, тільки почалося або на тестування. Важливо і те що він теж має право на доступ до перегляду всіх проєктів і завдань.

Виконавець — це роль, що працює безпосередньо тільки із завданнями. Він бачить їх та всіх виконавців, він має можливість дивитись на завдання інших та позначити власні, як виконані. Також, він має дозвіл на перегляд, до яких проєктів він залучений і які завдання йому доручили, і які завдання мають інші користувачі.

Спостерігач — це роль з найменшою кількістю прав, яка взагалі не має можливості виконати зміну даних в проекті. Він просто спостерігає за ходом виконання роботи іншими учасниками.

У цій діаграмі також зображено, що деякі з дій залежать одна від одної утворюючи зв’язок. Обов'язково мають бути визначені користувачі і призначені їм ролі з дозволами. Коли створюється нове завдання, треба під час цього обрати особу яка буде його виконувати і пріоритет. Це треба, безпосередньо, щоб все

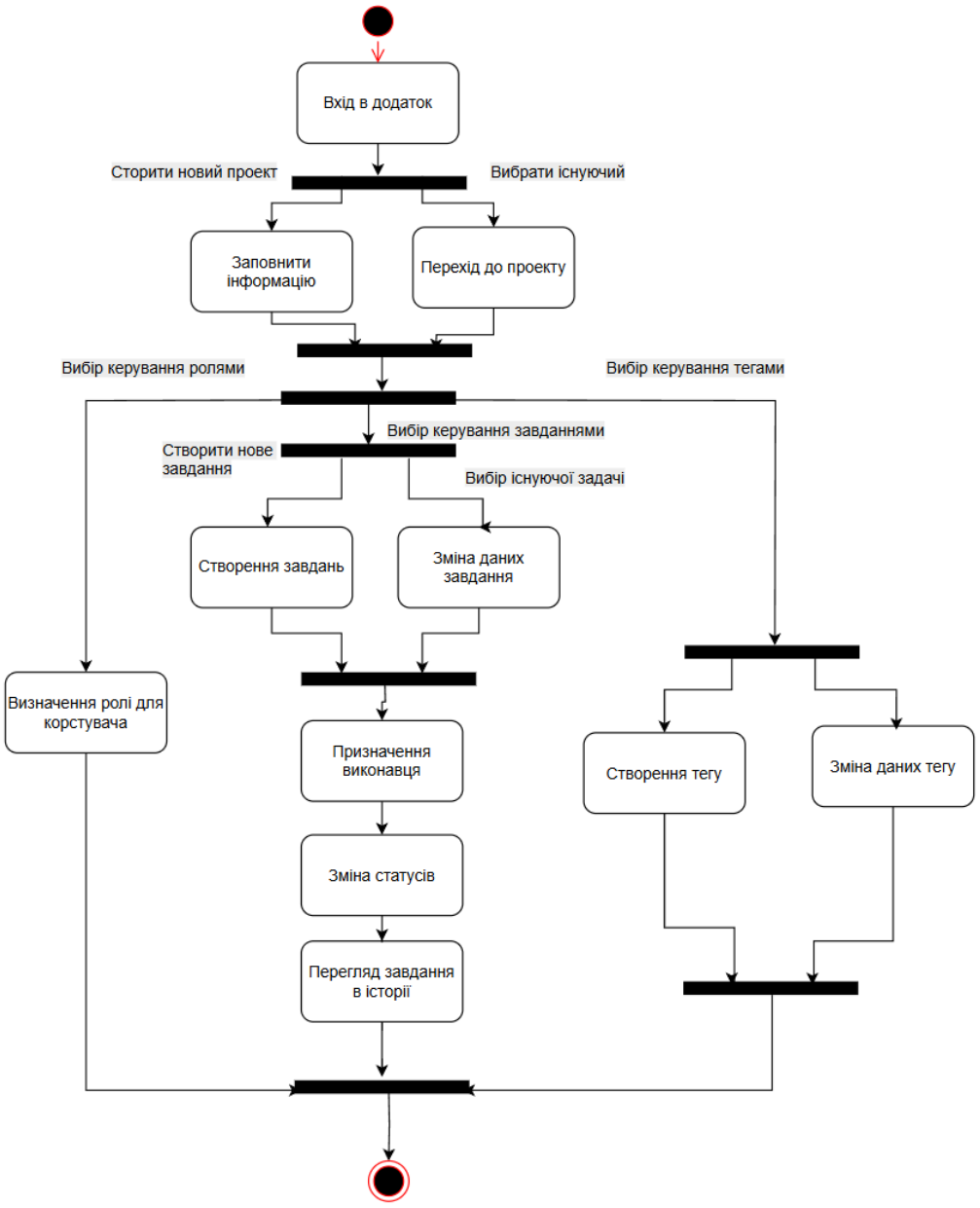
відбувалось злагоджено і коректно, щоб абсолютно жодна частинка процесу не була упущена.

На рисунку 1.2.2 показано діаграму активності. У той момент коли користувач заходить до застосунка, він має два варіанти дій, створити новий проєкт, або ж обрати той що вже існує і виконати перехід до нього. У першому випадку, потрібно заповнити всю обов’язкову інформацію у відповідні поля. У другому випадку, користувач виконує перехід до сторінки для того щоб переглянути або мати можливість для виконання редагування.

Після цього показано етап зміни користувачів. Користувач з роллю Адмін може налаштувати ролі, завдання та теги, тобто мітки для кращої організації роботи. У варіанті коли обрано керування ролями — визначається, хто і чим займається у проєкті. На противагу, коли обрано керування завданнями — можна створити нове завдання або змінити вже існуюче. Після цих всіх дій у визначеному завданні призначається сам учасник-виконавець, а також, у разі потреби, можна змінити статус завдання.

Якщо ж у користувача хоче працювати із тегами, він може виконати перехід до відповідного блоку і виконати створення нових тегів або ж відредагувати наявні в призначеному для цього вікні.

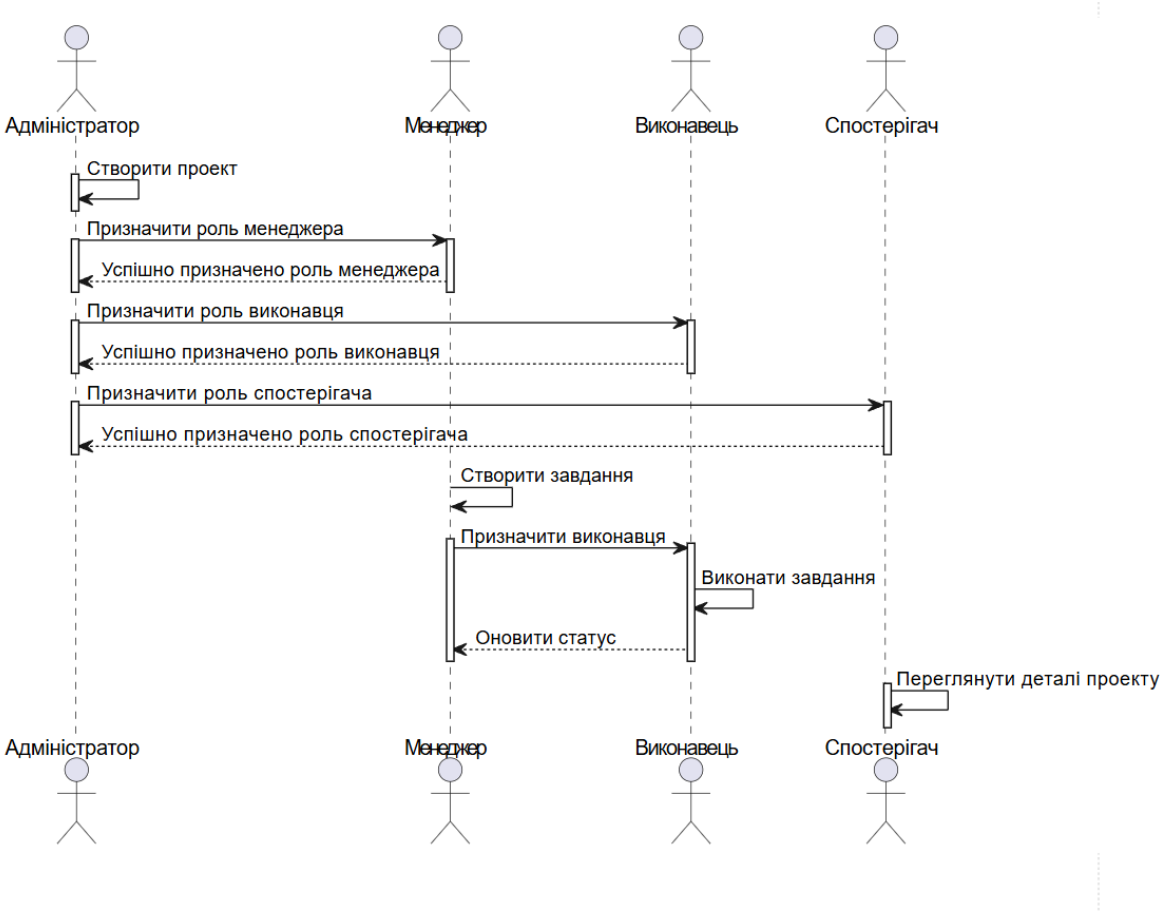
Абсолютно всі ці операції, сходяться до того що буде завершено взаємодія із системою. Всі необхідні операції мають бути виконані і система завершує свою роботу.



Діаграма 1.2.2 Активності

На рисунку 1.2.3 показано діаграму діяльності для користувача де детально представлено послідовність дій, що можна виконати під час роботи з системою, починаючи від самого моменту логіну, до початку виконання дій з проєктами, завданнями ті іншими її складовими. Діаграма показує, як виконується взаємодія із самою програмою в залежності від вибору. Створення або редагування проєкту, керування ролями учасників, управління завданнями або тегами. Вона

наочно показує логічну послідовність переходів між різними етапами, а також залежності між діями які виконує користувач.



Діаграма 1.2.3 Послідовності

1.3 Огляд існуючих рішень

На сьогоднішній день є дуже велика варіативність в рішеннях для програм управління проєктами. Вони в своєму функціоналі мають різні способи підходів, орієнтований на потреби команд що працюють використовуючи різні принципи Agile Kanban і тд. Нижче продемонстровано найпопулярніші і ті що найбагатші, функціоналом, серед них.

Trello — це один із найпопулярніших інструментів серед розробників по цілому світу для процесу організації завдань в командах. Він ґрунтується на методології Kanban, та дає змогу створювати картки та списки для завдань, інтерактивні борди, призначати до завдань виконавців, встановлювати терміни виконання та додавати медіатеги. Нажаль Trello має досить обмежену функціональність у куруванні управлінням залежностями між завданнями та ролями користувачів.

Asana — це нова платформа для розробки в команді, що дає можливість просто створювати проєкти, завдання, та до них, підзавдання, призначати відповідальних до створених завдань і підзавдань, визначати рівень пріоритетів та вести прогрес. Також, Asana має досить простий інтерфейс взаємодії і працює з інтеграціями сервісів, проте у безплатній версії додатку функціональність обмежена, а гнучкість керування ролями проста.

Jira — це складне і професійне рішення від компанії Atlassian, націлене на розробників компаній-гігантів. Воно надає хороший інструментарій для виконання контролю помилок у процесі виконання завдань, планування та реалізації спринтів, також можна виконувати керування беклогом та створювати звіти з діаграмами. Хоч і функціональність у Jira дуже велика, її налаштування та інтерфейс можуть бути занадто складними для новачків в цій сфері, а також надлишковими для невеликих компаній або команд що замаються розробкою не професійно.

ClickUp, Monday.com, Wrike та інші інструменти також надають доволі широкий діапазон для можливостей курування проєктами, такими як підтримку роботи в команді, автоматичне виконання бізнес процесів, керування етапами і тайм менеджментом, створення зручних та інформативних звітів. Проте, з них більшість, або є платними, або мають сильні обмеження у безплатних версіях.

Також вони не завжди підходять під специфічні вимоги окремих користувачів або невеликих команд.

Отже після проведеного аналізу існуючих рішень стало зрозуміло, що хоча на сучасному ринку додатків є багато готових рішень, більшість із них або ж дуже сильно насичені зайвими функціями, які не завжди потрібні для простих потреб, або ж не дають повної свободи у налаштуванні прав доступу, ролей, завдань та тегуванні у межах конкретного проєкту. У висновку можна сказати що створення своєї власної такої системи, є повністю виправданим кроком. Розроблена система буде максимально зручною, простою та ефективною щоб основний орієнтир був саме на прості.

1.4 Постановка завдання

Головною задачою для дипломної роботи це є розробка програмного забезпечення для інформаційної системи курування проєктами, тобто для створення сучасної, багатофункціональної та зручної системи, що дасть змогу ефективніше виконувати керування проєктами, налаштовувати робочі процеси для учасників та призначити їм завдання в межах кожного проєкту.

Відповідно постановка завдання вміщує в себе такі основні складові:

1. Розробка основних функціональних можливостей системи:
- Створення, редагування та видалення проєктів;
- Призначення ролей учасникам проєкту (менеджер, розробник, дизайнер тощо);
- Створення, редагування, перегляд та видалення завдань у межах проєктів;
- Встановлення статусів завдань (нове, в роботі, завершене тощо) та контроль

термінів виконання;

● Відображення прогресу виконання завдань в рамках кожного проєкту.

● Реалізація механізму управління користувачами та їхніми ролями у проєктах;

● Можливість контролю доступу до різних розділів системи залежно від ролі користувача.

1. Дизайн для користувацького інтерфейсу:

- Інтерфейс треба щоб був створений у відповідності до сучасних норм та принципів UI/UX. Із використанням темної кольорової палітри. Головними кольорами є темно сірий відтінку графіту, жовто-золотий та сірі кольори для контрасту з фоном. Використовуватиметься шрифт "Roboto", sans-serif, з конкретно визначеною ієрархічною моделлю. Ззаголовки мають бути жирними. Основний текст має бути максимально зручний для розуміння і розмір і міжрядковий інтервал мають бути такими щоб було гарно видно на великих дисплеях. Всі компоненти інтерфейсу — кнопки, картки, повідомлення, поля введення — оформлені в заокруглені кути, легка затіненість та збережену стилістичну цілісність, що надасть естетичний шарм та зручність і функціональність системи.
- Розробка зручного інтерфейсу для адмінів проєктів та виконавців завдань, що дасть змогу забезпечувати швидкий доступ до потрібної інформації;
- Інтерактивна візуалізація структури проєкту та пов’язаних із ним завдань.

2. Безпека та надійності:

● Реалізація системи автентифікації та авторизації користувачів;

● Захист персональних даних та запобігання несанкціонованому доступу до проєктної інформації;

● Резервне копіювання даних та можливість їх відновлення у разі непередбачених ситуацій..

3. Тестування та оптимізація:

● Проведення всебічного тестування функціоналу системи на відповідність вимогам;

● Виявлення та усунення помилок, а також покращення стабільності системи;

● Оптимізація продуктивності для забезпечення стабільної роботи з великими обсягами даних та багатьма користувачами.

Отже в моєму проекті постановка завдання є одинм з найважливіших кроків у процесі розробки даної інформаційної системи. Оскільки вона визначає основну мету, та функціональні вимоги з очікуваними результатами. Формулювання цих завдань відкриває двері до забезпечення покрокових та ефективних процесів в сфері розробки і досягти гарної якості кінцевого продукту.

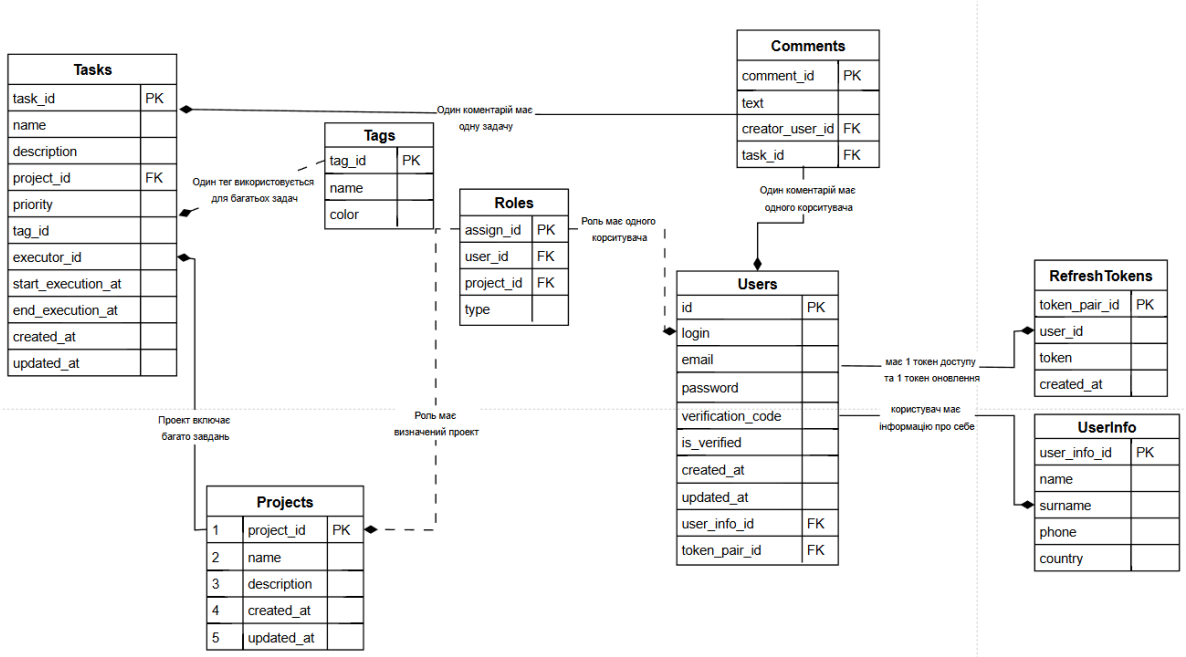
2. Проєктування інформаційного та програмного забезпечення

2.1 Логічна модель даних у вигляді ER-діаграми

ER-діаграма, або ж так звана діаграма "сутність-зв'язок" є гафічним представленням логічної моделі даних. Застосовується вона саме для моделювання сутностей, їхніх характеристик або атрибутів і взаємозв’язків між цими сутностями. Вона використовується як ефективний інструмент для аналізу та розробки структури бази даних.

В ході розробки власної платформи для розробки веб-сайтів, діаграма дає змогу чітко визначити ключові сутності, їхні властивості та логічні взаємозалежності в системі, тож саме така модель надає дуже глибоке розуміння організації структури даних. Також спрощує побудову ефективної та логічної БД.

Відповідну ER-діаграму подано на рисунку 2.1.1.



Діаграма 2.1.1 ER

Дана база була зроблена для зручного управління проєктами і завданнями. Тут враховано різні ролі користувачів, авторизацію, і ще деякі інші важливі деталі. Таблиця Projects зберігає дані про самі проєкти, а Tasks — це таблиця з завданнями, що мають відношення до конкретного проєкту (по полю ProjectId).

В кожному завданні є детальний опис, пріорітет, виконавець (ExecutorId), статус (StatusId), тег (TagId), поля для часу, типу коли створене і до коли потрібно виконати завдання. Статуси і теги — це окремі сутності-таблиці, вони реалізовані щоб краще виконувати фільтрування відповідно до потреб. Кожен тег, теж прив’язаний до проєкту через ProjectId.

Таблиця Users зберігає в собі користувачів, тобто їх логіни з паролями та пошту. А в UserRoles записано інформацію про те хто яку роль має в якому проєкті, треба це для того щоб розділяти функції для кожного з користувачів в кожному конкретному проєкті.

RefreshTokens таблиця яка створена для забезпечення безпеки в системі. Також в системі визначена таблиця UserInfos, яка зберігає детальну інформацію. Про користувача, а саме ім’я, фамілія, телефон і країна.

Всі ці таблиці зв’язані між собою через ключі, щоб нічого не загубилось і все працювало відповідно до функціональних вимог. Наприклад, Tasks прив’язано до Projects, Statuses і Tags, а UserRoles — до Users і Projects.

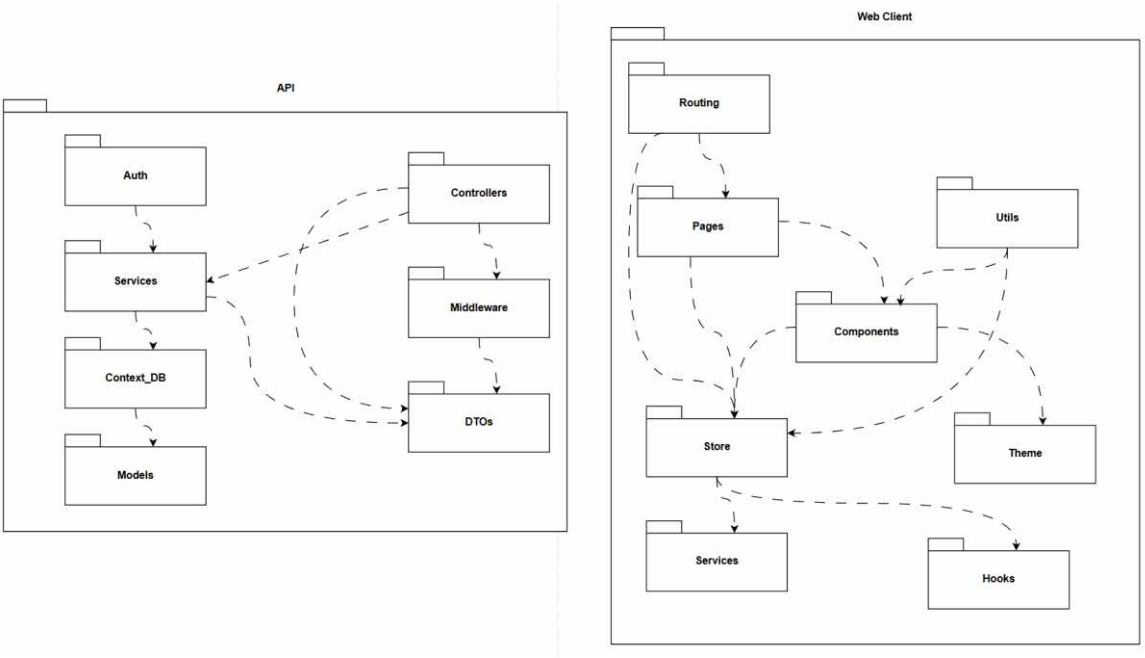
Ця модель бази дуже зручна тим, що можна легко визначати нові ролі, статуси, теги і не треба нічого кардинально змінювати в базі. В більшості таблиць є поля CreatedAt і UpdatedAt, для забезпечення можливості бачити зміни в записах що є корисним, коли щось піде не так, і тим самим швидше знайти причини помилки, опираючись на ці дані, чи просто для аудиту.

2.2 Діаграма пакетів

Діаграма пакетів застосовується для візуалізації організації сутностей та залежностей між різними модулями та пакетами. Вона сильно спрощує уявлення про організацію структуру програми. Тобто візуалізує кожен процес групування елементів разом та утворення близької взаємодії один з одним.

Основною ціллю представленої діаграми є відображення логічної організації системи та її компонентів на найвищому рівні абстракції. Головна задача це зосередитись на пакетній взаємодії, а не на конкретних шляхах реалізації внутрішніх сутностей таких як класи чи методи.

На діаграмі 2.2.1 зображено діаграму пакетів.



Діаграма 2.2.1 Пакетів

Діаграма показує повну структуру full-stack додатка з усією клієнтською та серверними частинами, API та Web Client. На сервері показано такі критично

важливі для системи модулі як для аутентифікації, шар з бізнес-логікою (Services), управління системи з СУБД (Context_DB, Models), обробки вхідних і вихідних запитів (Контролери та Middleware) та трансферу даних для парсингу до моделей для севера і клієнта. Клієнтська частина додатка включає в себе такі елементи як: маршрутизацію, сторінки, компоненти, управління станом, HTTP-запити, хуки, а також допоміжні утиліти та дизайн. Обидві частини взаємодіють через API-запити, забезпечуючи роботу додатка.

2.3 Діаграма компонентів

Діаграма компонентів також є ще одним видом відомих UML-діаграм. Ця діаграма використовується при розробці для візуалізації повноти архітектурної ідеї. Вона слугує засобом щоб візуалізувати структуру системи та взаємодію між її основними копонентами.

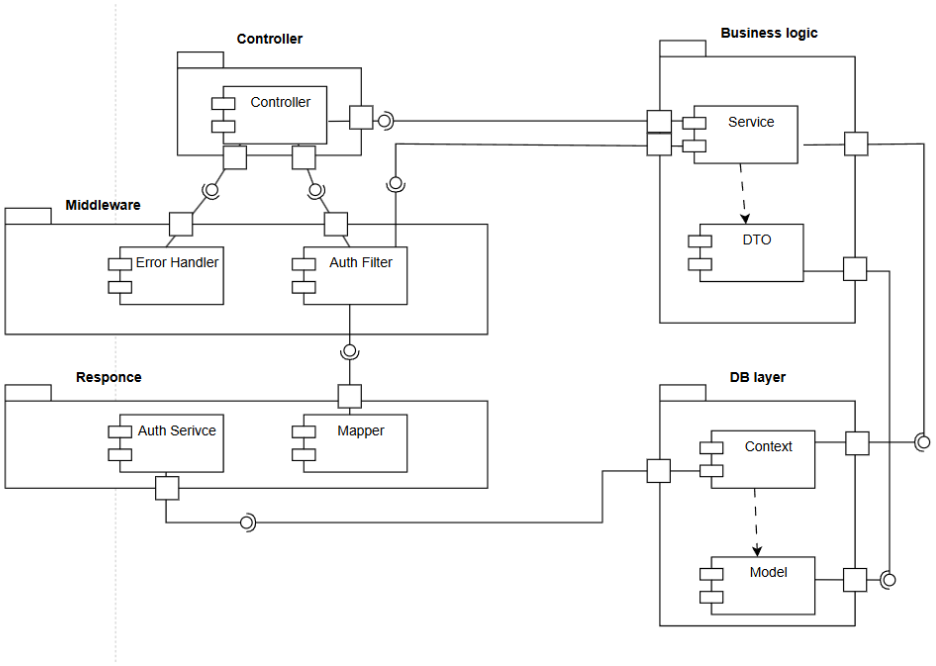
Найбільшу увагу в цій діаграмі приділено фізичному аспекту системи, таких як, наприклад компоненти, що утворюють її взаємозв’язок. Компоненти у цьому контексті — це незалежні елементи системи, що застосовуються для функціонального модуля. Вона також показує взаємодію з іншими подібними компонентами. Вони зберігаються в виді класів, модулів, бібліотек, пакетів або іншими сутностями.

Компоненти відображаються як блоки із назвами, що показують їх функціональну та фізичну роль у програмі. Взаємозв’язки у цій системі між компонентами позначають прямими стрілками або ж лініями, що показують напрямок і характер впливу.

Діаграма компонентів визначає структуру системи. Також, визначає варіанти взаємодії між її частинками та демонструє те які використані

інтерфейси. Ця діаграма є дуже корисною візуальною одиницею для виявлення компонентної взаємодії, повторно використовуваних компонентів, а також для створення картини з цілісним уявленням про архітектурні особливості.

На діаграмі 2.3.1 зображено діаграму компонентів.



Діаграма 2.3.1 Компонентів

Контролер (Controller). Контролер виконує просту роль посередника між клієнтом та бізнес-логікою. Він головний за приймання HTTP-запитів, їх обробку на вході та маршрутизацію до відповідних сервіс-систем. Після отримання готових результатів від сервісу, він виконує конфігурацію. Головна задача це перевірка коректності даних, обробка їх параметрів. Контролер дуже тісно взаємодіє з сервісним шаром, наприклад перекладаючи на нього складну логіку.

Сервісний шар містить в собі опис бізнес-логіки програми. Він виконує важливу обробку вхідних та вихідних даних. Виконання складних обрахунків та координацію роботи з різними підсистемами. Сервіси отримують дані від

контролера у вигляді DTO. Далі виконують необхідні операції з використанням моделей та структуру БД. Після чого повертають результати для подальшого форматування. Інколи, цей шар вміщує і іншу логіку таку як кешування, транзакцій і інші механізми для забезпечення цілісності даних.

Middleware виконує функцію для запитової фільтрації та попередньої їх обробки. Складається він з двох основних компонентів. Це обробник помилок та фільтра автентифікації. Перший, обробник помилок перехоплює абсолютно всі винятки, що з’являються в процесі роботи додатка. Логує їх та формує відповіді у зрозумілому для клієнта форматі. Фільтр автентифікації треба для перевірки прав доступу до запитуваних даних. Також виконує перевірку коректності токенів доступу. Це треба щоб забезпечити безпеку для API. Цей шар працює на найвищому рівні для абсолютно всіх маршрутів.

Далі, це компоненти для відповідей. Цей шар відповідає за фінальне форматування та підготовку даних для відправки для клієнта. Головна задача сервісу автентифікації - створення нових пар токенів доступу та оновлення даних авторизованої сесії. Маппер трансформує внутрішні моделеі у формати, що є зручними для клієнтської та серверної частин. Вони приховують конфіденційну інформацію та виконують процес оптимізації в загальну структуру. Всі ці компоненти забезпечують узгодженість у всіх даних та дотримуються загальноприйнятих принципів REST-API.

Бізнес-логіка включає всі компоненти які пов'язані з обробкою та зберіганням даних. Вона визначає структуру для безпечної передачі інформації між ними, забезпечуючи безпеку.

Шар бази даних інкапсулює всі операції що пов’язані із збереженням та отриманням інформації. В цьому шарі контекст відповідає за стани перебігу виконання запитів. Моделі описують самі сутності предметної області та їх

взаємозв’язок. Разом всі ці елементи створюють тісний механізм для реєстрації запитів, обробки та відправки даних.

Спершу запит проходить через шар Middleware. В Middleware виконується перевірка запиту на його коректність, та на те чи користувач має відповідні права. Далі він потрапляє до відповідного контролера. Він перекидає виконання бізнес-логіки до сервісів, сервіси використовують моделі та шар бази даних для обробки інформації. Та після чого всі результати відповіді передаються до компонентів для формування даних. Далі дані форматуються. Така архітектура є популярна та забезпечує масштабованість та легкість у підтримці.

3. Розробка інформаційного та програмного забезпечення

3.1 Система управління інформаційною базою

Вибір підходящої для системи СУБД напряму впливає на її стабільну роботу, розширюваність та гарантію безпеки системи: відповідно до потреб сучасного програмного забезпечення. Сьогодні найчастіше спеціалісти обирають ось такі рішення: PostgreSQL, MySQL та MongoDB. Кожна з цих перелічених систем БД має свої переваги та недоліки. Але на мою думку саме PostgreSQL як найкраще відповідає вимогам до системи. Найбільша частка уваги приділена структурованості даних, підтримка складних взаємозв'язків і висока надійність, що дуже важливо для мене.

PostgreSQL — це сучасна система керування даними що працює на основі об'єктно-реляційного типу, вона гарна тим що володіє широкими можливостями для моделювання даних. Вона дотримується відомого стандарту SQL. Забезпечує чистоту в логіці й передбачуваності у роботі з даними. Одним за найбільших полюсів системи PostgreSQL полягає саме у специфіці керування проєктами, а саме є її спроможність ефективно працювати з сутностями, такими як наприклад, між проєктами, виконавцями, завданнями та ролями, відповідно через ці особливості, забезпечується дуже великий рівень надійності. Також PostgreSQL має в своєму арсеналі досить таки широкий функціонал для забезпечення цілісності даних. Це наприклад, транзакції, резервне копіювання, масштабування, реплікації — ці всі аспекти є надійним інструментом для побудови сучасної систем з вимогами до стабільності та точності обробки даних.

Порівнюючи з MySQL, яка є спрощеною версію SQL та орієнтованою на простіші веб-додатки, PostgreSQL надає дуже широку функціональність і

набагато кращу підтримку складних запитів. Має більшу підтримку розширених типів даних для більш точної валідації вхідних даних. MySQL дуже добре інтегрується з системами з меншою логічною складністю. Але нажаль при дуже гострій потребі гнучкої обробки даних та дотримання суворої цілісності, PostgreSQL показує себе з кращої сторони.

MongoDB є яскравим представником типу NoSQL-баз даних і базується на зберіганні інформації у форматі простих файлів і документів, дає змогу точно працювати з динамічними або слабко структурованими табличними даними. Але, так само як і з попередніми аналогами вона має і свої мінуси: цей тип БД втрачає гнучкість в умовах коли є потреба в необхідності підтримки складних зв’язків і запитів, між сутностями. MongoDB не дає загальноприйнятої ідеї зовнішніх і внутрішніх ключів. Через це, нажаль, є неможливим автоматизоване забезпечення цілісності даних на рівні СУБД. У системах, де важлива структурованість і взаємозв’язок між об’єктами, така особливість може стати обмеженням.

Отже, мій вибір є виправданим і аргументованим і він припаждає на PostgreSQL як системи СУБД для моєї роботи. Вона задовільняє мене своїм спектром інструментарію для роботи зі структурними даними, доволі високим рівнем гнучкості, а також, відповідністю до суворих вимога комплектності й надійності даних в середині БД. На додачу, PostgreSQL також має відкритий вихідний код, що дає змогу здійснювати повний контроль над всією системою і надавати можливість для подальшої інтеграції під конкретні потреби.

3.2 Розробка інформаційної бази

Фізична модель бази даних зображена на рис. 3.2.1

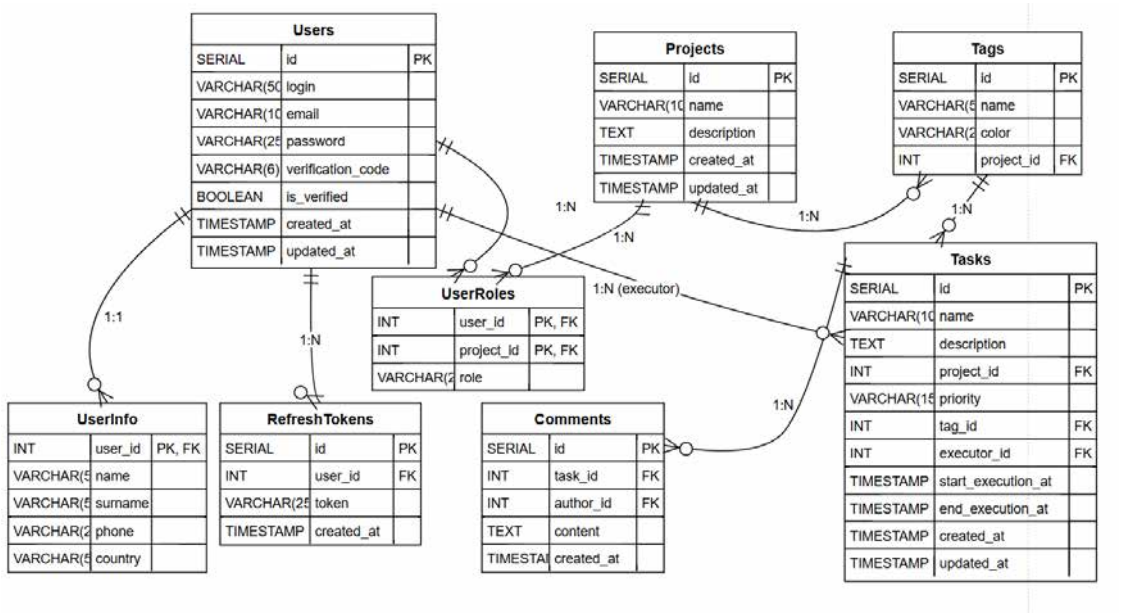


Рис. 3.2.1 Фізична модель бази даних

«Users» або Користувачі, це таблиця що вміщує в собі основні облікові дані про користувачів такі як логіни, паролі та електронні адреси, таблиця «Projects» (Проєкти), містить розгорнуту інформацію про проєкти, що створені користувачами, а саме, назва, дати та опис для організації процесів. «Tags» (Теги), використовується для того щоб маркувати завдання, зв’язана з конкретними проєктами, «UserRoles» призначена для ролей мають користувачі таких як адміністратора, менеджер, виконавець, переглядач в рамках певних проєктів. «UserInfo» – ця таблиця, зберігає особистісні дані про кожного з користувачів, такі як ім’я, прізвище, телефон та країна. Наступною табличкою в БД це табличка «RedreshTokens» – вона викорситовується створена для зберігання інформації для доступу та оновлення авторизації користуача, що дозволяють їм залишатися в системі після авторизації, та дозволяє системі автоматично продовжувати сесію. Табличка «Comments» (Коментарі) ствоврена

для коментарів до кожного із завдань, тобто містить інформацію про автора, вміст самого коментаря. Остання таблиця це таблиця «Tasks» (Завдання), містить інформацію про завдання в проєктах, такі як назва, опис, пріоритет, виконавець та терміни виконання.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

В ході розробки прикладної системи для теми мого дипломного проекту було виконано детальний і глибокий підбір. Аналіз можливого інструментарію дав змогу максимально відповідати вимогам до проєкту і дав змогу реалізувати всі функції. Основними критеріями відбору стали: надійність, стійкість, продуктивність, масштабованість, наявність достатньої кількості матеріалів для вивчення, простота інтеграції компонентів та можливість розширення.

У підсумку, було обрано сучасний технологічний стек, до складу яких входять наступні інструменти як:

Сереверна частина:

ASP.NET Core – це новий загальнодоступний та кросплатформений фреймворк для створення сучасних програм, пов'язаних із підключенням до інтернету, таких як веб-програми, програми для інтернету речей та мобільних серверів. Програмна технологія, що була запропонована відомоою корпорацією Microsoft як сучасна платформа для створення як веб-застосунків. Багато в чому є схожості з в технологію Java. Однією з ідей .NET є сумісність служб, написаних різними мовами. [9]

● Мова програмування C# – об’єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET, являється простою і в той же

час потужною мовою програмування, що дозволяє програмістам створювати багатофункціональні програми [7].

● Entity Framework являє собою спеціальну об'єктно-орієнтовану технологію на базі фреймворка .NET для роботи з даними. На відміну від засобів ADO.NET які дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища [10].

● JWT (JSON Web Token) – це технологія, яка забезпечує авторизацію клієнтів, на меті якої, є передача токена між клієнтом і сервером. Вона дає змогу створити деталізований механізм автентифікації та авторизації без збереження авторизованих сесій на сервері, що значно підвищує спроможність до розширення додатку і дає надійний рівень захисту.

Клієнт частина:

● React – це сучасна JavaScript-бібліотека для побудови інтерфейсів для web та мобільних застосунків. Вона дає змогу створити компонентно-орієнтовану структуру візуальних копонентів, що забезпечує надшвидку взаємодію між цими копонентами, дає змогу повторно використовувати елементи і зручно управляти станом. Основна мета React полягає в спрощенні створення складних користувацьких інтерфейсів, розділяючи їх на компоненти, які оновлюються автоматично при зміні даних [8].

● Redux Toolkit + RTK Query – сучасні рішення, що використовується для ефективного управління глобальним станом даних у React-додатку. RTK Query дає можливість реалізувати логіку взаємодії з API, наприклад це запити, кешування, обробка помилок, завантаження за спрощеною системою

підключенням, що значно спрощує підтримку та масштабування фронтенду системи.

● TypeScript – це мова програмування, розроблена корпорацією Microsoft у 2012 році. Вона позиціонується як інструмент для розробки вебдодатків, що розширює можливості JavaScript [14] У фронтенді використання цієї мови дає гарантію що буде підвищено стабільність додатку, що значно спростить розробку у команді, значно спростить рефакторинг і зробить краще документування коду.

● Material UI (MUI) – це дуже проста і мінімалістична стилістична React бібліотека для обгортки стандартних тегів HTML, інтеграція вже готових компонентів. Вона розроблена, за спеціалістами з Google - Material Design. Вона надає єдиний стиль, адаптивність інтерфейсу для різних пристроїв, а також дозволяє створювати естетично приємний і функціональний інтерфейс без потреби створювати компоненти з нуля.

● HTML5 – це відома гіпертекстова розмітка для сайтів, мобільних застосунків, яка використовується для структурування веб-сторінок. Він підтримує сучасні елемент такі як API сервіси та семантику, що дає змогу розробляти прості, зручні, доступні й інклюзивні інтерфейси.

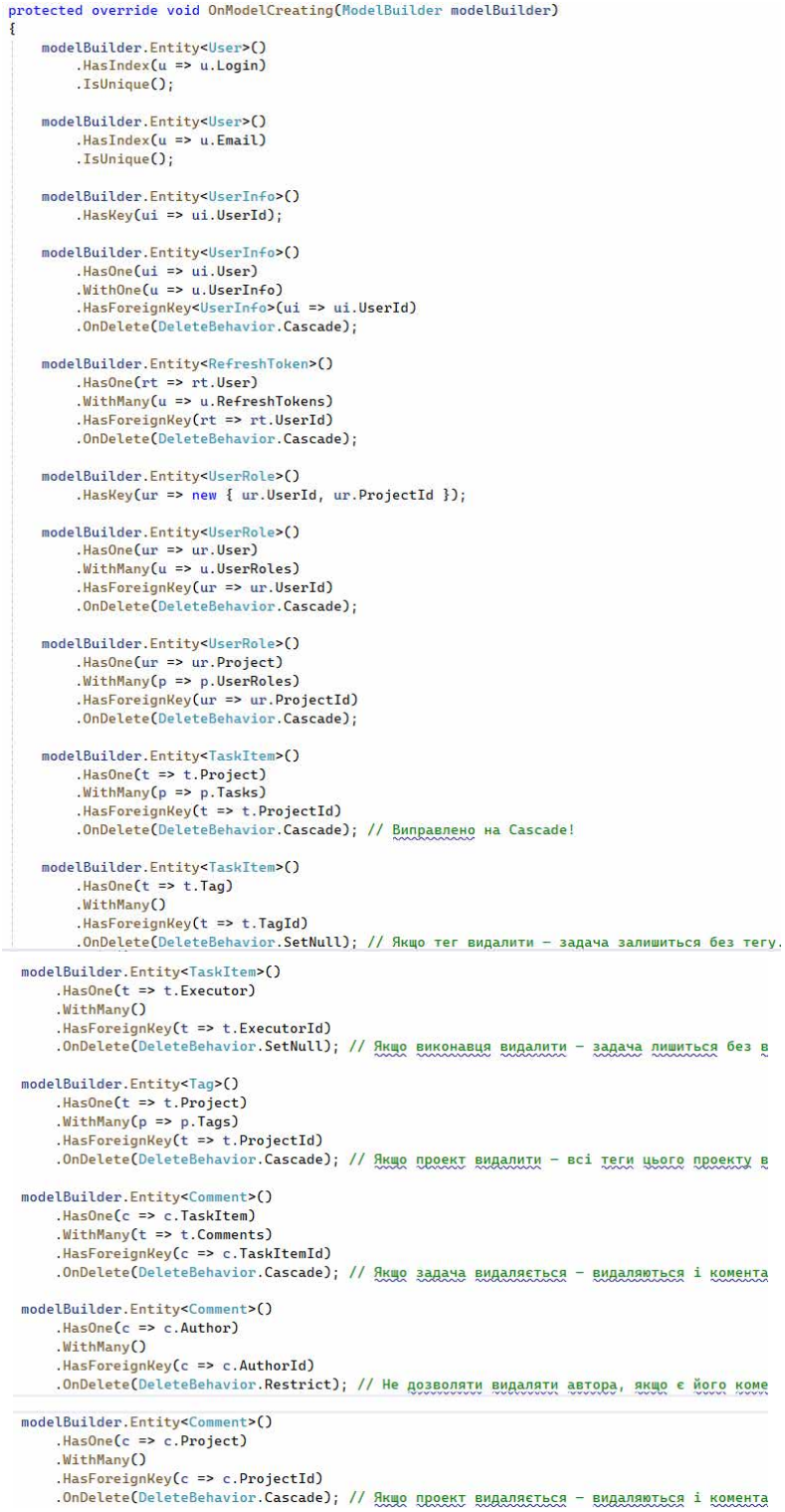
● CSS3 – це відома розмітка для створення стилізованих HTML сторінок для сайтів і не тільки, що забезпечує візуальне оформлення макету інтерфейсу: тобто за допомогою CSS3 реалізовується адаптивність, стилізація компонентів, анімація, тіні, та інші ефекти, що підвищують якість користувацького досвіду.

● JSON (JavaScript Object Notation) – це загальноприйнятий текстовий формат для передачі даних між клієнтом і сервером. Він найчастіше використовується для передачі інформації у клієнт-серверних застосунках та у розробці ігор, у структурованому форматі що дає змогу легко читати у парсованому вигляді.

Загалом, обраний стек дозволить виконати всі функціональні вимоги та досягти високого рівня узгодженості між клієнтською та серверною частинами, забезпечивши ти самим, швидку та доволі гнучку розробку, підтримку можливості масштабування. Саме такий підхід дасть змогу легко адаптувати платформу під різні потреби користувачів, доповнювати її новими модулями, забезпечувати ефективну підтримку та розвиток у майбутньому.

3.3 Розробка сереверної частини додатку

Код що зображений на Рис 3.3.1, визначає головний клас який відповідальний за роботу з БД, який успадковує контексти і представляє інтерфейс для роботи з базою даних, представляючи її управління через бібліотеку LINQ для реалізації, містить DbSet-властивості для роботи з табличками в базі даних що були описані в розділах вище. У конструкторі цього класу приймаються параметри для налаштування конфігурації та встановлення з’єднання з БД.



```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>()
        .HasIndex(u => u.Login)
        .IsUnique();

    modelBuilder.Entity<User>()
        .HasIndex(u => u.Email)
        .IsUnique();

    modelBuilder.Entity<UserInfo>()
        .HasKey(ui => ui.UserId);

    modelBuilder.Entity<UserInfo>()
        .HasOne(ui => ui.User)
        .WithOne(u => u.UserInfo)
        .HasForeignKey<UserInfo>(ui => ui.UserId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<RefreshToken>()
        .HasOne(rt => rt.User)
        .WithMany(u => u.RefreshTokens)
        .HasForeignKey(rt => rt.UserId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<UserRole>()
        .HasKey(ur => new { ur.UserId, ur.ProjectId });

    modelBuilder.Entity<UserRole>()
        .HasOne(ur => ur.User)
        .WithMany(u => u.UserRoles)
        .HasForeignKey(ur => ur.UserId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<UserRole>()
        .HasOne(ur => ur.Project)
        .WithMany(p => p.UserRoles)
        .HasForeignKey(ur => ur.ProjectId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<TaskItem>()
        .HasOne(t => t.Project)
        .WithMany(p => p.Tasks)
        .HasForeignKey(t => t.ProjectId)
        .OnDelete(DeleteBehavior.Cascade); // Виправлено на Cascade!

    modelBuilder.Entity<TaskItem>()
        .HasOne(t => t.Tag)
        .WithMany()
        .HasForeignKey(t => t.TagId)
        .OnDelete(DeleteBehavior.SetNull); // Якщо тег видалити – задача залишиться без тегу.

    modelBuilder.Entity<TaskItem>()
        .HasOne(t => t.Executor)
        .WithMany()
        .HasForeignKey(t => t.ExecutorId)
        .OnDelete(DeleteBehavior.SetNull); // Якщо виконавця видалити – задача лишиться без в

    modelBuilder.Entity<Tag>()
        .HasOne(t => t.Project)
        .WithMany(p => p.Tags)
        .HasForeignKey(t => t.ProjectId)
        .OnDelete(DeleteBehavior.Cascade); // Якщо проект видалити – всі теги цього проекту в

    modelBuilder.Entity<Comment>()
        .HasOne(c => c.TaskItem)
        .WithMany(t => t.Comments)
        .HasForeignKey(c => c.TaskItemId)
        .OnDelete(DeleteBehavior.Cascade); // Якщо задача видаляється – видаляються і комента

    modelBuilder.Entity<Comment>()
        .HasOne(c => c.Author)
        .WithMany()
        .HasForeignKey(c => c.AuthorId)
        .OnDelete(DeleteBehavior.Restrict); // Не дозволяти видаляти автора, якщо є його коме

    modelBuilder.Entity<Comment>()
        .HasOne(c => c.Project)
        .WithMany()
        .HasForeignKey(c => c.ProjectId)
        .OnDelete(DeleteBehavior.Cascade); // Якщо проект видаляється – видаляються і комента
```

Рис 3.3.1 Код контексту БД

На рисунку рис. 3.3.2 зображено код фільтру для виконання авторизації у .NET Web API. Принцип його роботи дуже простий - перевіряє права доступу

користувача до проєкту на основі даних користувача (ідентифікатор, проєкт, роль) які він отримує з так званих клеймів (інформація що зберігається в токені для ідентифікації користувача). Якщо клейми відсутні або не відповідає вимогам, повертає такі помилки як 401 або 400, далі створюється масив проєктів користувача з IProjectService та перевіряє, чи має користувач відповідну роль для того щоб отримати доступ до конкретного проєкту. Якщо ж проєкт не знайдено, повертає помилку 404, якщо роль не дозволена поверне 403, у разі інших - повертає 500.

```
public void OnAuthorization(AuthorizationFilterContext context)
{
    try
    {
        var user = context.HttpContext.User;
        var projectService = context.HttpContext.RequestServices.GetRequiredService<IProjectService>();

        var projectIdClaim = user.Claims.FirstOrDefault(c => c.Type == "ProjectId")?.Value;
        var projectRoleClaim = user.Claims.FirstOrDefault(c => c.Type == "ProjectRole")?.Value;
        var userIdClaim = user.Claims.FirstOrDefault(c => c.Type == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier")?.Value;

        // Перевірка наявності клеймів
        if (string.IsNullOrEmpty(projectIdClaim) || string.IsNullOrEmpty(projectRoleClaim) || string.IsNullOrEmpty(userIdClaim))
        {
            context.Result = new UnauthorizedObjectResult(new ApiResponse<object>(401, "Problem with role identification", null));
            return;
        }

        // Перевірка коректності клеймів
        if (!int.TryParse(projectIdClaim, out int projectId) || !int.TryParse(userIdClaim, out int userId))
        {
            context.Result = new BadRequestObjectResult(new ApiResponse<object>(400, "Invalid project or user ID", null));
            return;
        }

        // Отримання проектів користувача
        var userProjects = projectService.GetUserProjectsAsync(userId).GetAwaiter().GetResult();
        var project = userProjects.FirstOrDefault(p => p.Id == projectId);

        // Перевірка доступу до проекту та ролі
        if (project == null)
        {
            context.Result = new NotFoundObjectResult(new ApiResponse<object>(404, "Project not found", null));
            return;
        }

        if (!_allowedRoles.Contains(projectRoleClaim))
        {
            // Повертаємо 403 Forbidden з ApiResponse
            context.Result = new ObjectResult(new ApiResponse<object>(403, "You don't have the required role", null))
            {
                StatusCode = (int)HttpStatusCode.Forbidden
            };
            return;
        }
    }
    catch (Exception ex)
    {
        // Обробка несподіваних помилок
        context.Result = new ObjectResult(new ApiResponse<object>(500, "Internal Server Error", null))
        {
            StatusCode = (int)HttpStatusCode.InternalServerError
        };
    }
```

Рис. 3.3.2 Код методу фільтру (middleware) для перевірки ролі користувача

Контролер — виконує взаємодію між абстрактною моделлю і її уявленням. У задачі контролера входить відстеження визначених подій, які з’являються в

процесі дій користувача. Контролер дозволяє виконувати конструювання системи шляхом групування зв’язних операцій в спеціальний абстрактний клас.

Код (Рис. 3.3.2) реалізує два API-ендпоїнти в середині контролера по роботі з даними клієнтів для їх авторизації та реєстрації в ситемі. Перший це LoginAsync. Він приймає дані для входу, викликає метод LoginAsync. Якщо успішно відбувається процес перевірки користувача то у відповідь повертається інформацію про нього та пара токенів, refresh-токен встановлюється у cookie та повертається відповідь 200 OK до клієнта. У разі помилки звірки облікових даних повертає 401 Unauthorized, а при інших непередбачуваних помилках — 400 Bad Request. Наступний це RegisterAsync що приймає дані для реєстрації (це логін пароль, пошта), викликає RegisterAsync та якщо успішно виконана реєстрація, тобто дані збережені в БД після всіх валідацій, надсилає код підтвердження на пошту та повертає 200 OK що дасть змогу фронтенду виконати редірект до наступної сторінки. Якщо користувач вже існує або є інші помилки перевірки даних, повертає 400 Bad Request.

```
[HttpPost("login")]

public async Task<IActionResult> LoginAsync([FromBody] LoginRequest request)
{
    try
    {
        var response = await _authService.LoginAsync(request);
        SetRefreshTokenCookie(response.refresh_token);
        return Ok(new ApiResponse<UserResponce>(200, "Login successful", response));
    }
    catch (InvalidCredentialsException ex)
    {
        return Unauthorized(new ApiResponse<string>(401, ex.Message));
    }
    catch (Exception ex)
    {
        return BadRequest(new ApiResponse<string>(400, ex.Message));
    }
}

[HttpPost("register")]

public async Task<IActionResult> RegisterAsync([FromBody] RegistrationRequest request)
{
    try
    {
        var dto = await _authService.RegisterAsync(request);
        return Ok(new ApiResponse<UserDTO>(200, "Verification code is sended", dto));
    }
    catch (UserAlreadyExistsException ex)
    {
        return BadRequest(new ApiResponse<string>(400, ex.Message));
    }
    catch (Exception ex)
    {
        return BadRequest(new ApiResponse<string>(400, ex.Message));
    }
}
```

Рис 3.3.3 Код методів ендпоінтів для логіну на реєстрації

Тепер детальніше розглянемо шар сервісу, метод LoginAsync (Рис 3.3.4) є асинхронним методом для обробки логіну користувача що на меті має перевірити його облікові дані. Спершу він починає перевіряти, на факт існування цього користувача в базі даних за вказаною електронною поштою, якщо ні, повертає виняток InvalidCredentialsException який опрацьовується в ендпоінті з попереднього рисунку. Якщо ж користувач знайдений, перевіряється пароль за допомогою методу BCrypt.Verify для дешифрації паролю та їх звіряння. Якщо пароль невірний, повертається інший виняток InvalidCredentialsException. В

іншому випадку при успішній автентифікації генерується refresh token, термін дії якого становить 48 годин, і зберігається в базі даних. Наприкінці метод повертає об'єкт UserResponce, що містить access token який дійсний 6 хвилин, та refresh token який дійсний 3 години. Далі виконується маппінг даних користувача.

```
public async Task<UserResponce> LoginAsync(LoginRequest loginUser)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.Email == loginUser.Email);
    if (user == null)
        throw new InvalidCredentialsException("Email doesn't exist!");

    //if (!BCrypt.Net.BCrypt.Verify(loginUser.Password, user.Password))
    //    throw new InvalidCredentialsException("Invalid password!");

    var refreshToken = await GenerateToken(user, null, 2880);
    await SaveRefreshToken(user.Id, refreshToken);

    return new UserResponce
    {
        access_token = await GenerateToken(user, null, 10),
        refresh_token = refreshToken,
        user = MapToUserDto(user, null, null)
    };
}
```

Рис 3.3.4 Код методу логіну в сервісі

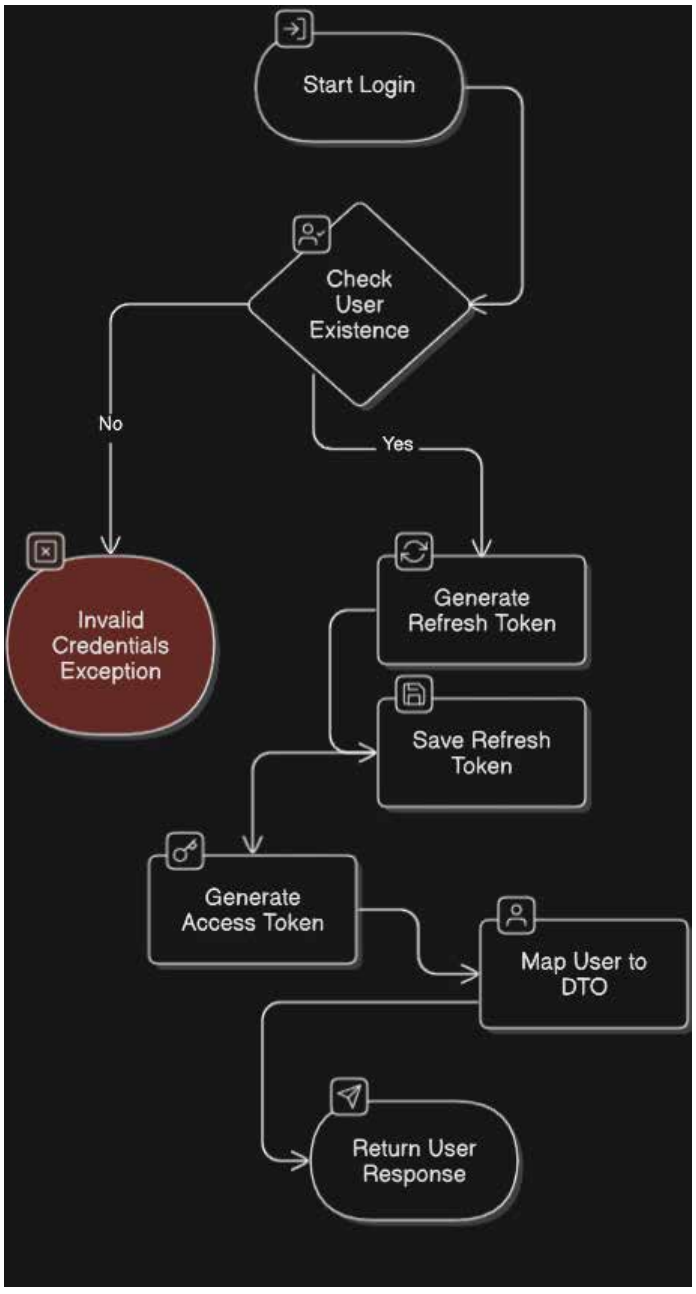


Рис 3.3.5 Блок-схема для методу логіну

Метод шару бізнес-логіки сервісу RefreshAsync (Рис 3.3.4) обробляє оновлення refresh токену та генерує нові пари токенів таких як токен доступу та refresh токен. Спершу він виконує перевірку на те чи переданий токен є дійсним і коректним, тобто виконується перевірка в якій використовуючи його ми можемо отримати доступ до ендпоінту. Далі за допомогою методу перевірки коректності токенів, якщо ж токен не є дійсний, код викидає виняток, потім

отримує ідентифікатор користувача з токена та перевіряє, чи вказано projectId та чи є він в системі. Якщо projectId не передано, метод намагається отримати його з токена шляхом розшифровки даних. Далі відбувається процес пошуку користувача по його ідентифікатору в БД.

Якщо користувач або його токен не знайдені код повертає помилку на клієнт, в протилежному випадку, метод перевіряє, чи є у користувача доступ до цього проєкту. У випадку якщо всі перевірки пройдені, генерується новий токен доступу і токен оновлення, який оновлюється в базі даних, метод повертає нові токени та дані користувача в об'єкті UserResponce.

```
public async Task<UserResponce> RefreshAsync(string token, int? projectId)
{
    var principal = ValidateToken(token);
    if (principal == null)
        throw new InvalidTokenException("Invalid refresh token");

    var userId = int.Parse(principal.FindFirstValue(ClaimTypes.NameIdentifier));

    if (projectId == null)
    {
        var projectIdClaim = principal.Claims
            .FirstOrDefault(c => c.Type == "ProjectId" || c.Type.EndsWith("ProjectId"))?.Value;

        if (projectIdClaim != null)
        {
            projectId = int.Parse(projectIdClaim);
        }
    }

    var user = await _context.Users.FindAsync(userId);
    if (user == null)
        throw new UserNotFoundException("User not found");

    var existingToken = await _context.RefreshTokens.FirstOrDefaultAsync(rt => rt.Token == token);
    if (existingToken == null)
        throw new TokenNotFoundException("Refresh token not found");

    string role = null;
    if (projectId.HasValue)
    {
        bool hasAccess = await _context.UserRoles
            .AnyAsync(ur => ur.UserId == userId && ur.ProjectId == projectId.Value);

        if (!hasAccess)
            throw new UnauthorizedAccessException("User does not have access to this project");

        // Get the role for the specified project
        role = await _context.UserRoles
            .Where(ur => ur.UserId == userId && ur.ProjectId == projectId.Value)
            .Select(ur => ur.Role)
            .FirstOrDefaultAsync();
    }

    var newAccessToken = await GenerateToken(user, projectId, 10);
    var newRefreshToken = await GenerateToken(user, projectId, 2880);

    existingToken.Token = newRefreshToken;
    existingToken.CreatedAt = DateTime.UtcNow;
    await _context.SaveChangesAsync();

    return new UserResponce...;
}
```

Рис 3.3.6 Код методу оновлення токену access на основі refresh із cookie

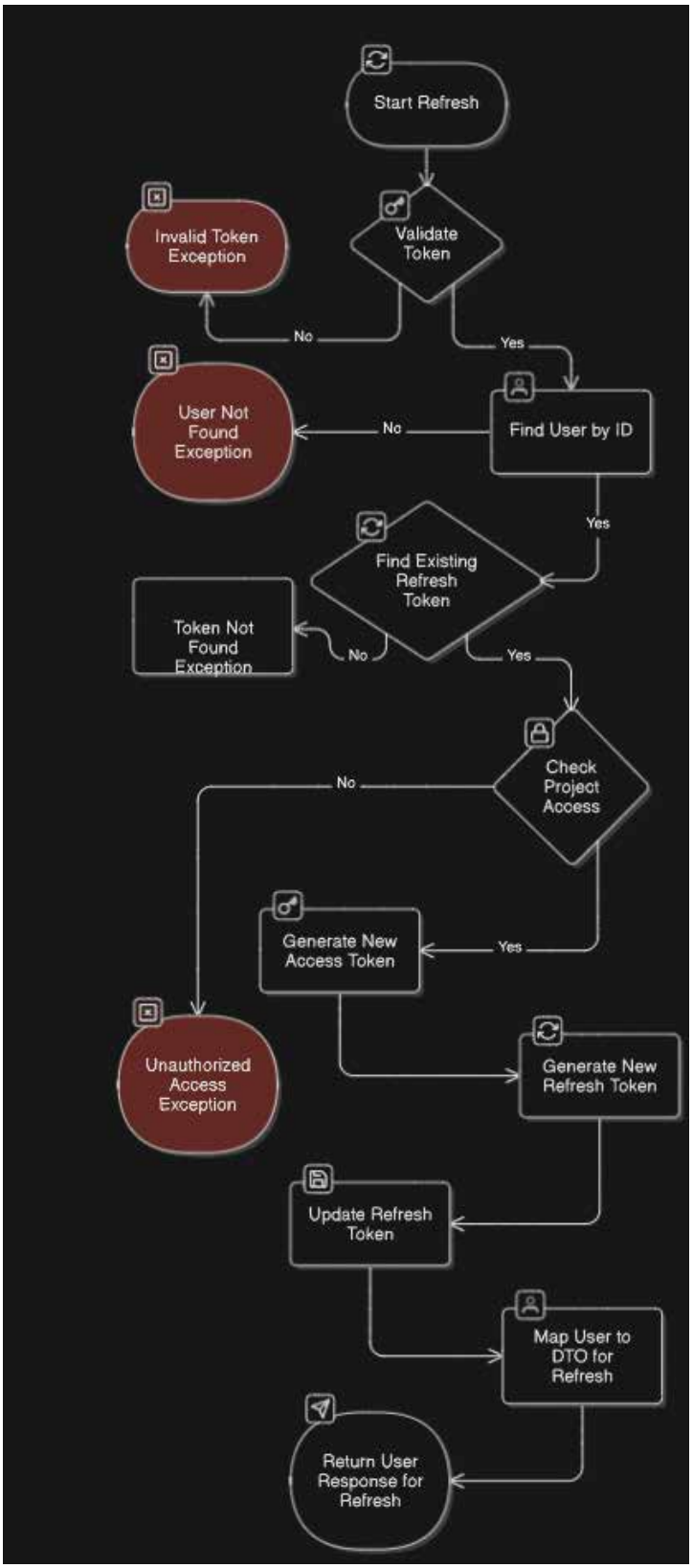


Рис 3.3.7 Код методу для оновлення токену доступу

Метод AssignRoleAsync що продемонстровано на Рис 3.3.8 що призначає роль користувачеві для конкретного проєкту. Спочатку він виконує перевірку, чи існує заданий користувач UserId та чи існує проєкт з вказаним projectId. Якщо користувача або проєкт не знайдено програма повертає помилку. Далі код перевіряє, чи є роль, яку намагаються призначити, дійсною тобто одною з допустимих ролей а сам: "ADMIN", "MANAGER", "EXECUTOR", "VIEWER". Наступно перевіркою є, чи вже існує роль для цього користувача в даному проєкті щоб уникнути дублювання і відповідно розбіжностей. Якщо роль існує, вона оновлюється, якщо ні то відповідно - створюється новий запис у таблиці UserRoles. Після цього зміни зберігаються в базі даних. Наприкінці метод повертає об'єкт, який містить інформацію про користувача, його логін, електронну пошту та призначену роль – тобто інформацію яка треба для коректного відображення на клієнтській частині.

```
public async Task<RoleResponce> AssignRoleAsync(int projectId, AssignRoleRequest request)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.Id == request.UserId);
    var projectExists = await _context.Projects.AnyAsync(p => p.Id == projectId);

    if (user == null)
        throw new NotFoundException("User not found.");
    if (!projectExists)
        throw new NotFoundException("Project not found.");

    var validRoles = new[] { "ADMIN", "MANAGER", "EXECUTOR", "VIEWER" };
    if (!validRoles.Contains(request.Role))
        throw new ValidationException("Invalid role.");

    var existingRole = await _context.UserRoles
        .FirstOrDefaultAsync(ur => ur.UserId == request.UserId && ur.ProjectId == projectId);

    if (existingRole != null)
    {
        existingRole.Role = request.Role;
    }
    else
    {
        var newRole = new UserRole
        {
            UserId = request.UserId,
            ProjectId = projectId,
            Role = request.Role
        };
        _context.UserRoles.Add(newRole);
    }

    await _context.SaveChangesAsync();

    return new RoleResponce
    {
        UserId = request.UserId,
        UserLogin = user.Login,
        UserEmail = user.Email,
        Role = request.Role
    };
}
```

Рис 3.3.8 Код методу для привласнення ролі в проєкті в сервісі

Метод UpdateTaskAsync (Рис 3.3.9) асинхронно оновлює задачу в базі даних на основі переданих даних. Спочатку виконується перевірка, чи об'єкт оновлення задачі taskDto не є порожні. Потім відбувається процес пошуку самої задачі в базі даних за taskId, в разі успішної перевірки та пошуку відбувається наступний крок а саме, перевірка правильності тегу для цього проєкту. Після цього виконується перевірка наявності виконавця в системі, якщо вказано його ID, це збереже від того що присвоєно користувача або тег з іншого проєкту. Далі оновлюють значення полів задачі (якщо нові значення надано), і зміни зберігаються в базі даних. Після цього метод повертає оновлену задачу у вигляді TaskDto.

```
public async Task<TaskDto> UpdateTaskAsync(int taskId, TaskUpdateDto taskDto)
{
    if (taskDto == null)...

    // Очікуємо завершення пошуку задачі
    var task = await _context.Tasks
        .Include(t => t.Project)
        .FirstOrDefaultAsync(t => t.Id == taskId);

    if (task == null)
    {
        throw new Exception("Task not found.");
    }

    // Очікуємо завершення пошуку тегу
    if (taskDto.TagId.HasValue)
    {
        var tag = await _context.Tags
            .FirstOrDefaultAsync(t => t.Id == taskDto.TagId && t.ProjectId == task.ProjectId);

        if (tag == null)...
    }

    // Очікуємо завершення пошуку виконавця
    if (taskDto.ExecutorId.HasValue)
    {
        var executor = await _context.UserRoles
            .FirstOrDefaultAsync(ur => ur.UserId == taskDto.ExecutorId && ur.ProjectId == task.ProjectId);

        if (executor == null)...
    }

    // Оновлення полів задачі
    task.Name = taskDto.Name ?? task.Name;
    task.Description = taskDto.Description ?? task.Description;
    task.Priority = taskDto.Priority ?? task.Priority;
    task.TagId = taskDto.TagId ?? task.TagId;
    task.Status = taskDto.Status ?? task.Status;
    task.ExecutorId = taskDto.ExecutorId;
    task.StartExecutionAt = taskDto.StartExecutionAt ?? task.StartExecutionAt;
    task.EndExecutionAt = taskDto.EndExecutionAt ?? task.EndExecutionAt;

    // Очікуємо завершення збереження змін
    await _context.SaveChangesAsync();

    return new TaskDto...;
}
```

Рис. 3.3.9 Код методу оновлення завдання в сервісі

Метод GetTasksAndUserOfProjectAsync що також належить до шару сервісів (Рис 3.3.10) має на меті надати список завдань певного проєкту з урахуванням фільтрів, згрупованих за виконавцями та датами. Спершу відбувається ініціалізація об’єктів фільтрації, якщо він не переданий, і виконується обробка дати початку та завершення дедлайну у форматі UTC. Далі код намагається отримати всіх користувачів що мають конкретно визначені ролі в проєкті, якщо вказано ExecutorIds, лишаються лише ті, чий ідентифікатор міститься у списку. Далі формується процес запиту до бази даних, який обирає всі задачі до конкретно визначеного проєкту та основі Id з токену запиту та фільтрує їх за назвою, пріоритетом, тегами, статусом та діапазоном дедлайнів, далі задачі сортуються за датою завершення, для коректного і логічного виведення даних на клієнті у прямому або зворотному порядку, отримані задачі перетоврюються на DTO-об’єкти з необхідними полями. Наступним кроком для кожного користувача обираються задачі, де він є виконавцем і відбувається групування за датами завершення, і результат формується і отримані об’єкти перетворюються в UserTasksListDto. Окремо формуються задачі, які не мають виконавця або ж які не відповідають жодному з зазначених користувачів проєкту, і додаються до результату як окрема група з Id = null, це треба для відображення окремого списку в якому містяться не розподілені задачі. Метод повертає готовий, згрупований список користувачів із коректними задачами, згрупованими по датах, та відсортованих відповідно до вимог клієнта.

```
public async Task<IEnumerable<UserTasksListDto>> GetTasksAndUserOfProjectAsync(int projectId, TaskFilterRequest? filterRequest)
{
    filterRequest ??= new TaskFilterRequest();

    DateTime? startDateUtc = filterRequest.DeadlineDateStart?.ToUniversalTime();
    DateTime? endDateUtc = filterRequest.DeadlineDateEnd?.Date.AddDays(1).AddTicks(-1).ToUniversalTime();

    var users = await _context.UserRoles
        .Where(ur => ur.ProjectId == projectId)
        .Select(ur => ur.User)
        .Distinct()
        .Where(u => !filterRequest.ExecutorIds.Any() || filterRequest.ExecutorIds.Contains(u.Id))
        .ToListAsync();

    var tasksQuery = _context.Tasks
        .Where(t => t.ProjectId == projectId);

    if (!string.IsNullOrEmpty(filterRequest.SearchQuery))
    {
        tasksQuery = tasksQuery.Where(t => t.Name.Contains(filterRequest.SearchQuery));
    }

    if (filterRequest.Priorities.Any())
    {
        tasksQuery = tasksQuery.Where(t => filterRequest.Priorities.Contains(t.Priority));
    }

    if (filterRequest.TagIds.Any())
    {
        tasksQuery = tasksQuery.Where(t => t.TagId.HasValue && filterRequest.TagIds.Contains(t.TagId.Value));
    }

    if (filterRequest.Statuses.Any())
    {
        tasksQuery = tasksQuery.Where(t => filterRequest.Statuses.Contains(t.Status));
    }

    tasksQuery = tasksQuery.Where(t => t.EndExecutionAt.HasValue &&
        (!startDateUtc.HasValue || t.EndExecutionAt.Value >= startDateUtc.Value) &&
        (!endDateUtc.HasValue || t.EndExecutionAt.Value <= endDateUtc.Value));

    bool descending = filterRequest.SortByDeadline?.EndsWith("desc") ?? true;
    tasksQuery = descending
        ? tasksQuery.OrderByDescending(t => t.EndExecutionAt)
        : tasksQuery.OrderBy(t => t.EndExecutionAt);

    var tasks = await tasksQuery
        .Select(t => new TaskDtoForList
        {
            Id = t.Id,
            Name = t.Name,
            Priority = t.Priority,
            TagId = t.TagId,
            Status = t.Status,
            ExecutorId = t.ExecutorId,
            EndExecutionAt = t.EndExecutionAt
        })
```

```
    var groupedTasks = users.Select(user => new UserTasksListDto
    {
        Id = user.Id,
        TasksByDate = tasks
            .Where(t => t.ExecutorId == user.Id)
            .GroupBy(t => t.EndExecutionAt?.Date ?? DateTime.MinValue)
            .OrderBy(g => g.Key)
            .Select(g => new TaskGroupDto
            {
                Date = g.Key.ToString("yyyy-MM-dd"),
                Tasks = g.ToList()
            })
            .ToList()
    }).ToList();

    // Додаємо окремий елемент для неприкріплених тасок
    var unassignedTasks = tasks
        .Where(t => t.ExecutorId == null || !users.Any(u => u.Id == t.ExecutorId))
        .GroupBy(t => t.EndExecutionAt?.Date ?? DateTime.MinValue)
        .OrderBy(g => g.Key)
        .Select(g => new TaskGroupDto
        {
            Date = g.Key.ToString("yyyy-MM-dd"),
            Tasks = g.ToList()
        })
        .ToList();

    if (unassignedTasks.Any())
    {
        groupedTasks.Add(new UserTasksListDto
        {
            Id = null,
            TasksByDate = unassignedTasks
        });
    }

    return groupedTasks;
}
```

Рис. 3.3.10. Метод для фільтрації завдань

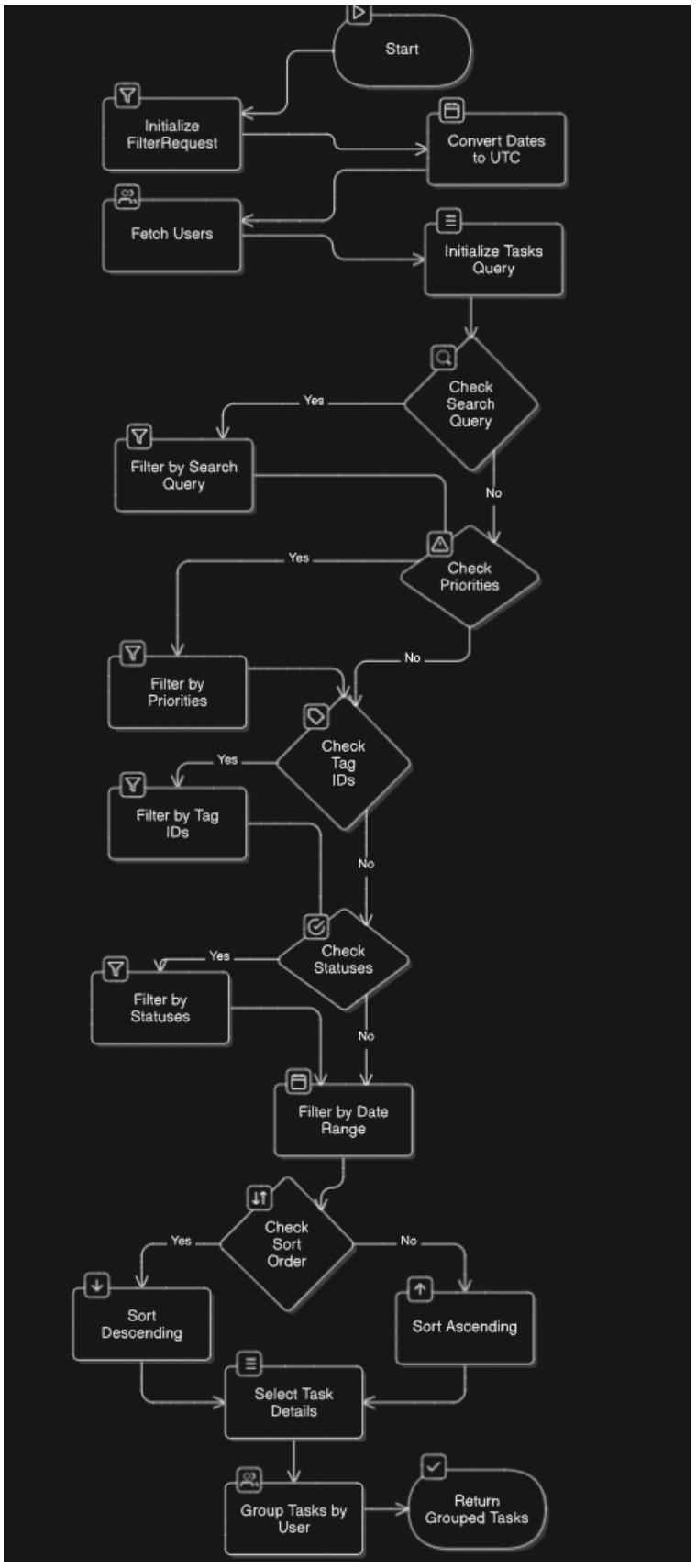


Рис. 3.3.11 Блок-схема алгоритму для фільтрації завдань

3.4 Розробка клієнтської частини додатку

Код що показано на Рис 3.4.1, створений для виконання запитів в асинхронному режимі «baseQueryWithReauth», який виконує оновлення токена в тому випадку коли він застарів, тобто час його валідності вичерпався, або в є інші помилки 401 типу. Для початку виконується звичайний запит через «baseQuery». Алгоритм намагається виконати оновлення токену, зробивши POST-запит на адресу «auth/refresh». У випадку якщо все добре вони зберігаються за допомогою функції «saveTokens» . Якщо токен оновиться то повторюється оригінальний запит з новим токеном. В іншому випадку коли оновити токен не вдалося, виводиться помилка. Далі токени очищаються, і виконується автоматичний вихід із системи, очищаючи дані в Redux через «api.dispatch» запиту «clearUser». Після всіх операцій метод повертає результат запиту, який або є оновленим результатом запиту після повторної авторизації, або оригінальним, якщо токен не потрібно оновлювати.

```
const baseQueryWithReauth: BaseQueryFn<string | FetchArgs, unknown, FetchBaseQueryError> = async (
  args,
  api,
  extraOptions
) => {
  let result = await baseQuery(args, api, extraOptions);

  if (result.error?.status === 401) {
    console.log('Отримано 401 помилку. Спробуємо оновити токен...');

    const refreshResult = await baseQuery(
      {
        url: `auth/refresh`,
        method: 'POST',
      },
      api,
      extraOptions
    ) as { data?: LoginResponse };

    // Перевіряємо, чи вдалося оновити токен
    if (refreshResult.data) {
      const { access_token, refresh_token } = refreshResult.data.data;
      saveTokens(access_token, refresh_token);
      console.log('Токени оновлено:', refreshResult);

      // Повторюємо оригінальний запит з оновленим токеном
      result = await baseQuery(args, api, extraOptions);
    } else {
      console.error('Не вдалося оновити токен. Виконуємо logout...');
      clearTokens(); // Очищаємо токени
      api.dispatch({ type: 'user/clearUser' }); // Виконуємо логаут
    }
  }

  return result;
};
```

Рис 3.4.1 Інтерцептор для оновлення некоректного токену доступу

Код що на зображено на Рис 3.4.2 створює об'єкт «createApi» від API. Для роботи з авторизацією, він виконує налаштування базового запиту через «baseQueryWithReauth». В середині реалізована обробка для виконання оновлення токенів при отриманні помилки типу «401» від серверу. Також в процесі розробки було реалізовано ендпоїнти такі як «login», який використовується для виконання входу користувача. Програма відправляє POST-запит на auth/login з інформацією для входу, така як пошта і пароль, і у результат повертає у відповідь типу «LoginResponse». «logout», у свою чергу, видалення сесії користувача з системи надсилаючи запит для видалення на auth/logout і повертаючи інформацію. «registration» потрібен для створення нового акаунту користувача, надсилаючи POST-запит на «auth/register» з даними для реєстрації та отримуючи відповідь у вигляді об’єкта користувача.

Вирифікація користувача «verify» застосовується для виконання підтвердження із застосуванням верифікаційного коду, надсилаючи запити на «auth/verify» і повертаючи у результат типу «LoginResponse». Також система може виконувати оновлення пари токенів доступу, надсилаючи запит на «auth/refresh» з параметром конкретно обраного проекту якщо він є , тобто опціонально, для оновлення токенів. «getUser», використовується для отримання даних про користувача токени якого збережені в стореджі в системі на основі токену, надсилаючи GET-запит на auth/me що повертає відповідь у вигляді об`єкту з визначеними полями інформації про користувача.

```
export const authApi = createApi({
  reducerPath: 'authApi',
  baseQuery: baseQueryWithReauth,

  endpoints: (builder) => ({
    login: builder.mutation<LoginResponse, LoginRequest>({
      query: (loginRequest) => ({
        url: 'auth/login',
        method: 'POST',
        body: loginRequest,
      }),
    }),
    logout: builder.mutation<string, void>({
      query: () => ({
        url: 'auth/logout',
        method: 'DELETE',
      }),
    }),
    registration: builder.mutation<OnlyUserDataResponse, RegisterRequest>({
      query: (regRequest) => ({
        url: 'auth/register',
        method: 'POST',
        body: regRequest,
      }),
    }),
    verify: builder.mutation<LoginResponse, VerifyRequest>({
      query: (verifyRequest) => ({
        url: '/auth/verify',
        method: 'POST',
        body: verifyRequest,
      }),
    }),
    refresh: builder.mutation<LoginResponse, number>({
      query: (projectId) => ({
        url: `auth/refresh${projectId ? `?projectId=${projectId}` : ''}`,
        method: 'POST',
      }),
    }),
    getUser: builder.query<OnlyUserDataResponse, void>({
      query: () => "/auth/me",
    }),
  }),
});
```

Рис 3.4.2 Визначення з’єднання з сервером для визначення ендпоінтів авторизації

Рис 3.4.3 область коду де виконуються різні запити і реалізовано логіку для обробки самих процесів збереження даних, вона імпортує редуктори для різних частин додатку («auth», «project», «roleAndUser», «tag», «task» та «usersReducer»),

виконую ініціалізацію логіку RTK, що дає змогу виконати збереження станів таких як: авторизації, проєктів, ролей користувачів, тегів, задач та інформації про користувачів. У залежностях утворюються всі необхідні функції для кожного з API для їх коректної роботи і взаємозв’язку і станом, що дозволяє налаштувати обробку запитів та кешування через «RTK Query». Змінні «RootState» та «AppDispatch» використовуються для типізації глобального стану та функцій «dispatch» у додатку, щоб полегшити роботу з типами у мові TypeScript. Отже сховище де зберігається інформація про всі глобальнні дані для зручного імпорту у всіх компонентах додатка, за допомогою «configureStore» з допоміжної бібліотеки «@reduxjs/toolkit», виконуючи налаштування редуктори та функцій для роботи з кількома API та збереження стану додатку.

```
const appReducer = combineReducers({
  [authApi.reducerPath]: authApi.reducer,
  [projectApi.reducerPath]: projectApi.reducer,
  [roleAndUserApi.reducerPath]: roleAndUserApi.reducer,
  [tagApi.reducerPath]: tagApi.reducer,
  [taskApi.reducerPath]: taskApi.reducer,
  user: usersReducer,
});

const rootReducer = (state: any, action: any) => {
  if (action.type === 'RESET_STATE') {
    return appReducer(undefined, action);
  }
  return appReducer(state, action);
};

export const store = configureStore({
  reducer: rootReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(
      authApi.middleware,
      projectApi.middleware,
      roleAndUserApi.middleware,
      tagApi.middleware,
      taskApi.middleware
    ),
});

export const resetStore = () => {
  store.dispatch({ type: 'RESET_STATE' });
  store.dispatch(authApi.util.resetApiState());
  store.dispatch(projectApi.util.resetApiState());
  store.dispatch(roleAndUserApi.util.resetApiState());
  store.dispatch(tagApi.util.resetApiState());
  store.dispatch(taskApi.util.resetApiState());
};

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Рис 3.4.3 Store для глобального доступу до даних

На Рис 3.4.4 показано код форми «LoginForm». Форма що реалізує форму для входу користувача. Використовується хук-функція «useState» для зберігання введених даних, електронної пошти та пароля. Надсилаючи форму ініціюється виклик «useLoginMutation», що виконує запит на виконання автентифікації на сервер.

Зберігаються токени доступу та оновлюється стан користувача в Redux, після чого користувача перенаправляє на сторінку проєктів. У випадку якщо ж сервер повертає помилку або ж нічого не повертає впродовж конкретно визначеного часу а саме 15 секунд відображається сповіщення з описом помилки за допомогою хука «useSnackbar». Цей компонент виконує коректну обробку з'єднання з системою браузеру зберігаючи всю необхідну інформацію в стейті сайту на localstorage та cookies з відображенням відповідних повідомлень про успішність виконання запиту або про помилку.

```
const LoginForm = () => {
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [login, { isLoading }] = useLoginMutation();
  const { enqueueSnackbar } = useSnackbar(); // Хук для сповіщень

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      dispatch(clearUser())
      const response = await login({ email, password }).unwrap();
      console.log(response.data.access_token, response.data.refresh_token)
      if (response.success) {

        saveTokens(response.data.access_token, response.data.refresh_token);
        dispatch(setUser(response.data.user));

        navigate('/projects');
      }
    } catch (err) {
      console.error('Login failed:', err);
      const apiError = err as ApiErrorResponse
      enqueueSnackbar(apiError.data.message || 'Login failed. Please try again.', {
        variant: 'error', // Тип сповіщення (помилка)
      });
    }
  };
```

Рис. 3.4.4 Логіка роботи логін форми

Код що зображено на (Рис 3.4.5) визначає розмітку для форми входу користувача, яка використовує компоненти «Material-UI» для створення

стилізованого інтерфейсу, де компонент «Container» створює основну зону, що знаходиться в самому центрі на сторінці і займає рівновіддалену висоту і ширину сайту. Всередині цього контейнера знаходиться «Paper» – компонент який використовує тінь для створення ефекту 3D, містить форму з двома полями введення такими як електронної пошти («TextField» для email) та поля для пароля що приховує вміст. Кнопка відправлення форми («Button»). Під час завантаження сторінки, відмальовується обертальний індикатор («CircularProgress»), також знизу після кнопки для входу є текст, який містить інформацію про перехід та посилання на сторінку реєстрації, для користувача в якого нема акаунту або бажає створити ще один. Форма має адаптивний стиль при наведенні на посилання.

```
return (
  <Container
    component="main"
    maxWidth="xs"
    sx={{
      display: 'flex',
      flexDirection: 'column',
      alignItems: 'center',
      justifyContent: 'center',
      minHeight: '100vh',
    }}
  >
    <Paper
      elevation={3}···
      }}
    >
      <Typography component="h1" variant="h5" sx={{ color: 'primary.main', mb: 3 }}>
        Login
      </Typography>
      <form onSubmit={handleSubmit} style={{ width: '100%' }}>
        <TextField
          variant="outlined"···
          }}
        />
        <TextField
          variant="outlined"···
          }}
        />
        <Button···
        </Button>
      </form>
      <Typography variant="body2" sx={{ color: 'text.primary' }}>
        Don't have an account?{' '}
        <MuiLink
          component={Link}···
          }}
        >
          Register
        </MuiLink>
      </Typography>
    </Paper>
  </Container>
);
```

Рис 3.4.5 react верстка для форми логіну

4 Рекомендації щодо впровадження та експлуатації системи

4.1 Тестування системи

Вікно для входу в систему (Рис 4.1.1) займає два поля для введення даних що треба для входу. Перше поле використовується для введення електронної пошти, зірочка вказує на обов'язковість заповнення цього поля. Друге поле використовується для введення паролю і також є обов’язковими. Обидва поля, є текстовими. Поле пароля приховує введені символи, що потрібно для безпеки. Під полями введення розташована кнопка з написом "Login", яка призначена для підтвердження введених даних та входу в систему. Нижче посилання на форму для реєстрації нового акаунту.

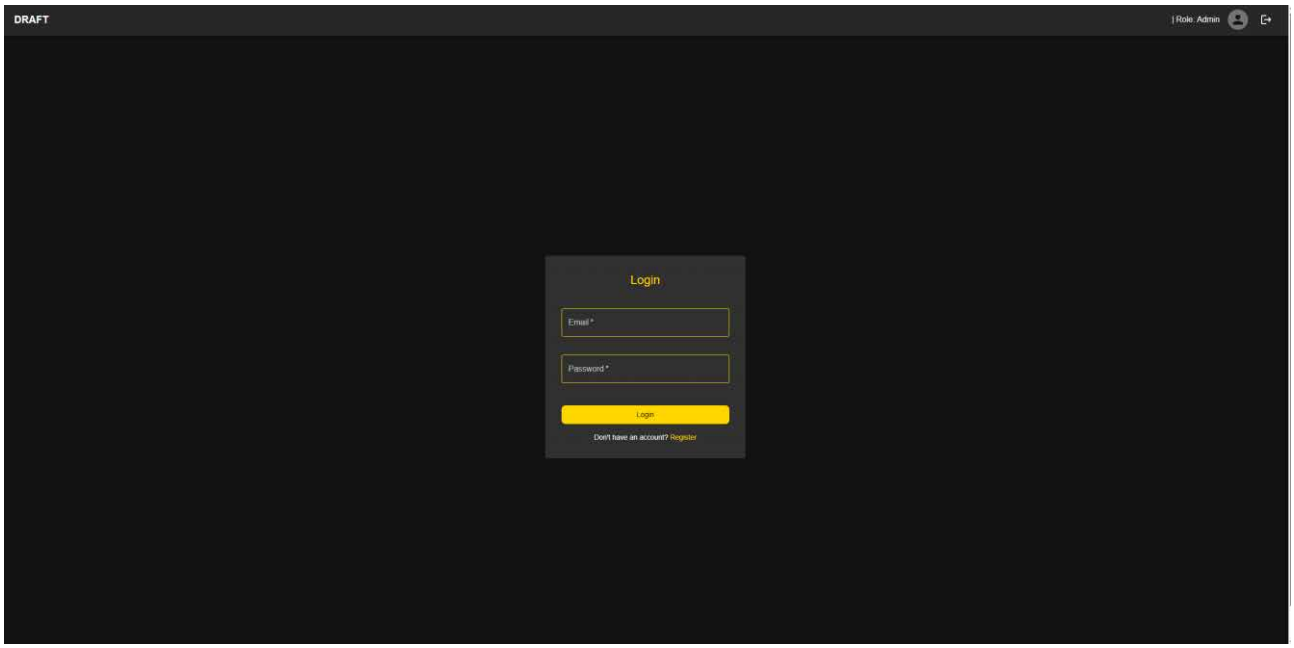


Рис 4.1.1 Вікно логіну

Наступне вікно (Рис 4.1.2) призначене для створення нового облікового запису. Для успішної реєстрації вам необхідно заповнити всі наявні обов'язкові поля які позначені зірочкою «*». У полі "Login" треба ввести унікальне ім'я користувача яке не використовувалось раніше і не збережено в БД. Ім'я може містити літери, цифри та символи наприклад, User_123. У полі "Email" треба вказувати актуальну адресу електронної пошти, на яку система відправить підтвердження реєстрації, а саме код для варифікації. У полі "Password" треба придумати надійний пароль, який рекомендується складати з 8 і більше символів, використовуючи комбінацію з великих і малих букв, чисел та спец символів.

Після того як відбулось коректне заповнення всіх полів треба натиснути на кнопку "Register" для завершення процесу реєстрації. Якщо ж користувач має обліковий запис у системі, можна скористатися посиланням "Already have an account? Login" для переходу на сторінку входу.

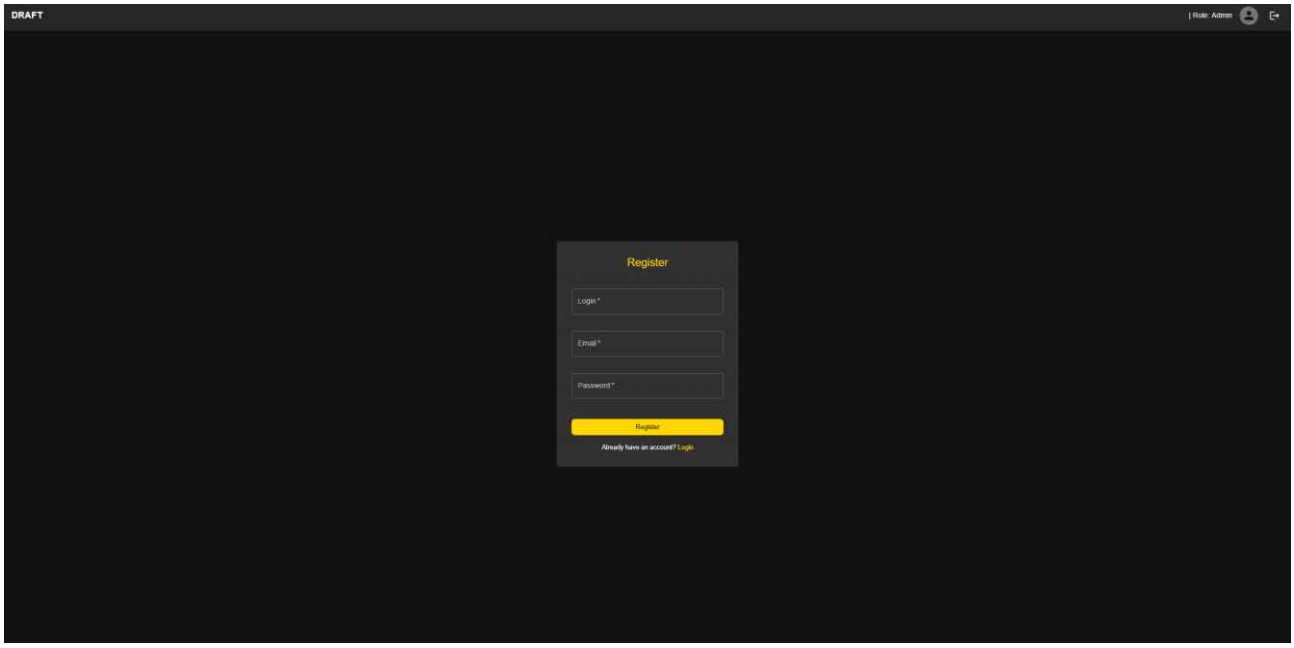


Рис 4.1.2 Вікно реєстрації

Вікно (Рис 4.1.3) призначене для завершального етапу реєстрації - підтвердження введеної електронної адреси. В цьому вікні після заповнення реєстраційної форми система відправить код підтвердження на пошту. Необхідно

ввести цей код у поле "Verification Code *", яке позначене як обов'язкове для внесення. Якщо ж відбувся якийсь збій і користувач не отримали код, то в такому випадку рекомендується спочатку перевірити папку "Спам" у поштовій скриньці. Важливо вводити код точно так, як він вказаний у листі, без додаткових пробілів чи символів, саме найкраще це просто скопіювати його. Після успішного підтвердження електронної пошти ваш обліковий запис буде повністю активовано і підготовлено до роботи. Ввівши всі дані відбудеться перехід на головну сторінку.

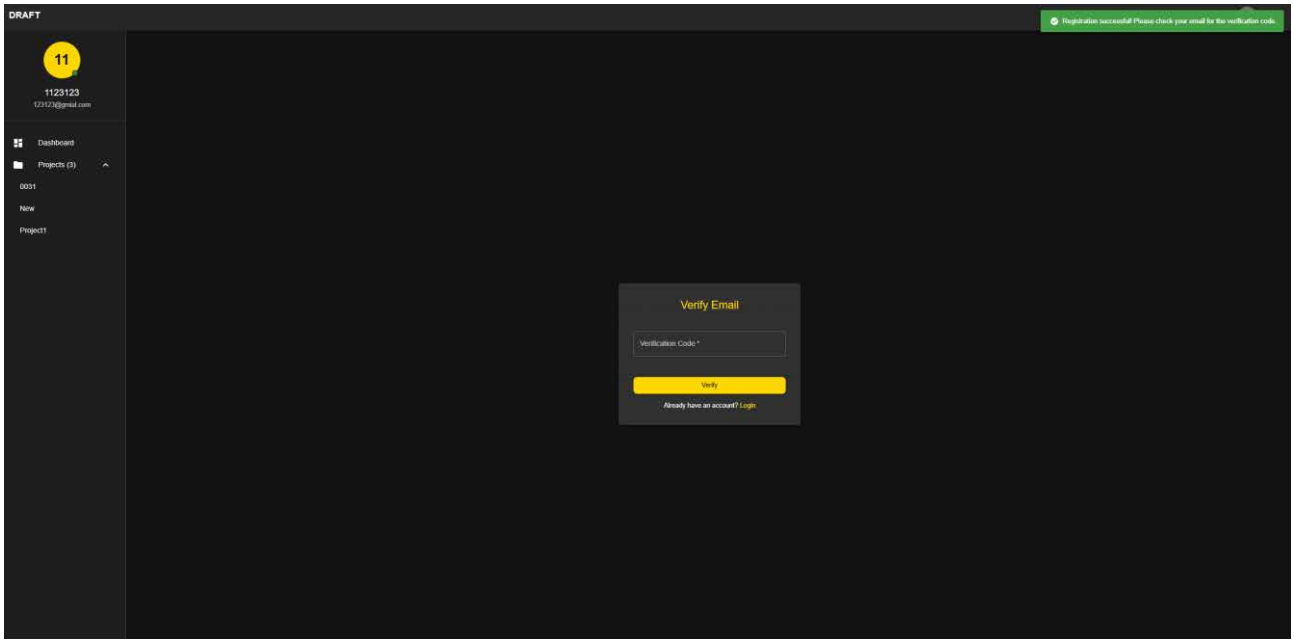


Рис 4.1.3 Вікно Верифікації

Сторінка "My Projects" (Рис 4.1.4) відображає список проєктів користувача у вигляді списку карток, кожна з цих карток має назву, опис і дату його створення. У правому верхньому куті є кнопка "Create New Project" яка дозволяє створити новий проєкт.

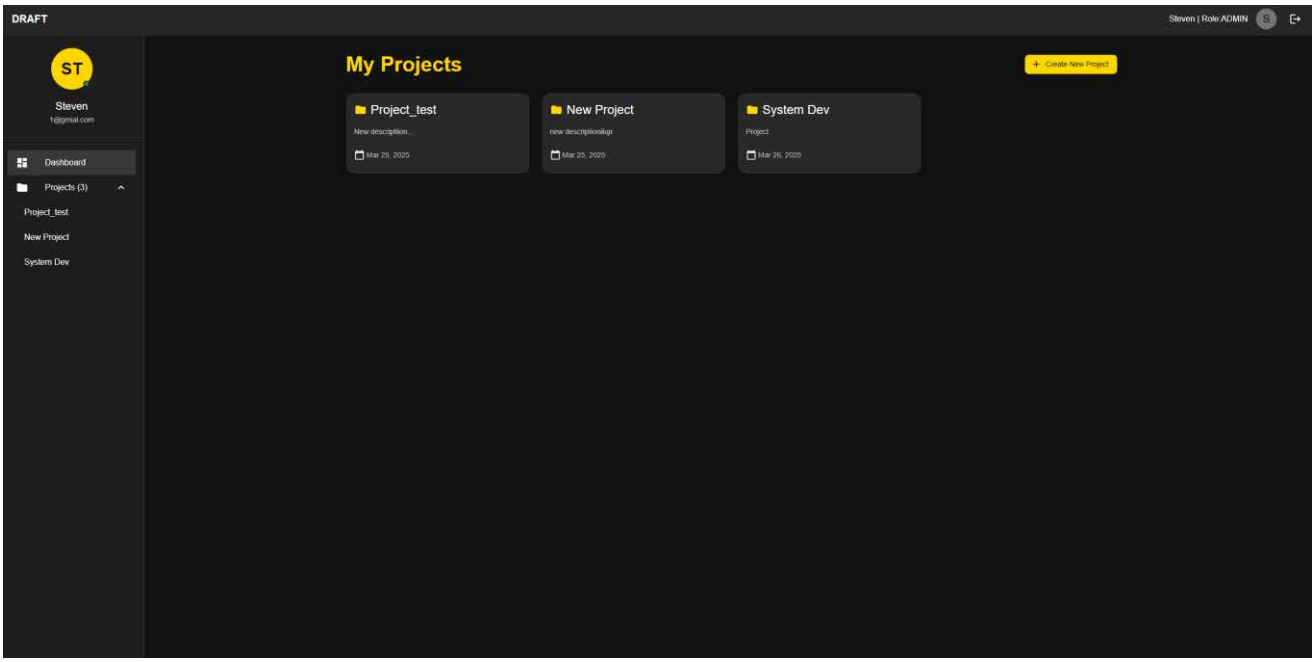


Рис 4.1.4 Сторінка із списком проєктів

Наступне вікно (Рис 4.1.5) дозволяє створити новий проєкт . У полі "Project Name" вводить назву проєкту, а в полі "Project Description" — короткий опис де можна ввести до 250 символів, включаючи пробіли нижні прочерки та інше. Заповнивши ці дані кнопка "Create Project" стане активною після заповнення цих полів.

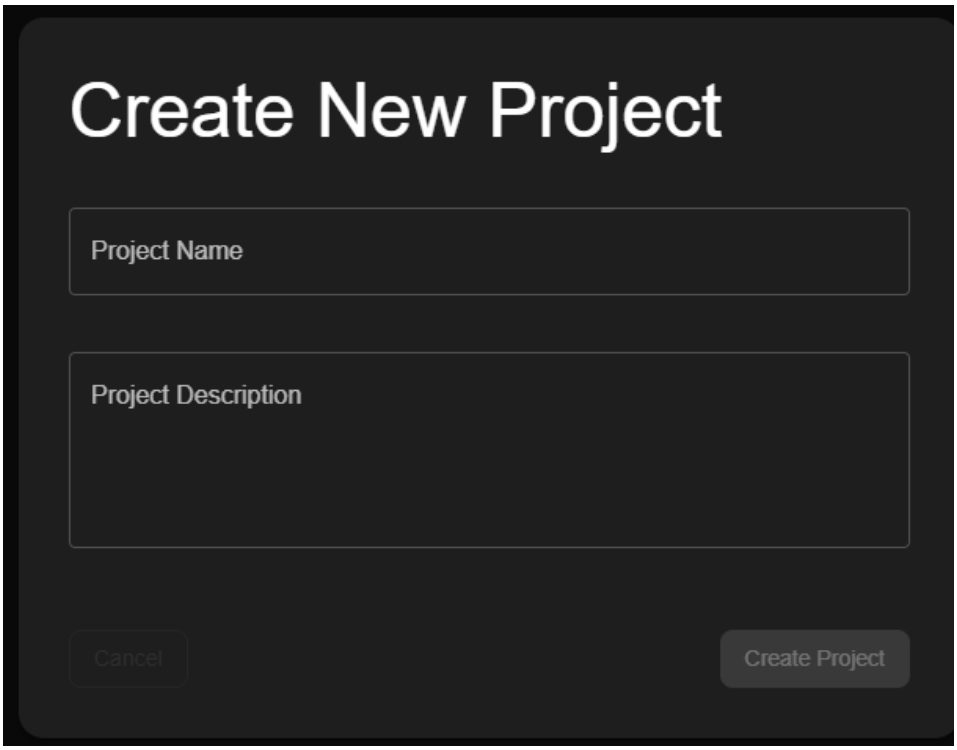


Рис 4.1.5 Створення проєкту

Наступна сторінка — це сторінка для керування самими проєктами, Сторінка яка допомагає команді ефективно розробляти організацію в робочих процесах, в середині проекту. Можна, стежити за станом перебігу виконання завдань та контролювати їхній стан та користувачів що до неї назначені, Реалізовано кілька основних блоків, де кожен із яких виконує свою важливу функцію, тут юзери можуть застосовувати фільтри за виконавцем, тегом, статусом або дедлайном. Це полегшує розуміння швидко находити всю необхідну інформацію та оцінювати їх стан для роботи над проєктом.

Учасникам проєкту можуть призначати конкретно визначені ролі, які визначають їхні права у системі і доступ до функцій. Наприклад, деякі користувачі мають можливість створювати та редагувати завдання, тоді як інші — лише переглядати їх. Такий підхід надає чіткий розподіл відповідальностей, захищає зміну даних користувачам які не мають на це права та допомагає уникнути плутанини в роботі команди.

Для зручності в організації та управління завдань для пошуку є доступна система тегів, користувачі можуть як використовувати існуючі теги, так і створювати нові відповідно до потреб проєкту. Це дає змогу дость легко згруповувати завдання за темами або пріоритетами.

Також розроблено окремий блок з відображенням непризначених завдань це дозволяє менеджерам та адміністраторам оперативно розподіляти навантаження між учасниками команди та уникати простоїв у роботі, або зберігати завдання в непризначеному статусі якщо якесь завдання тимчасово неможливо виконати.

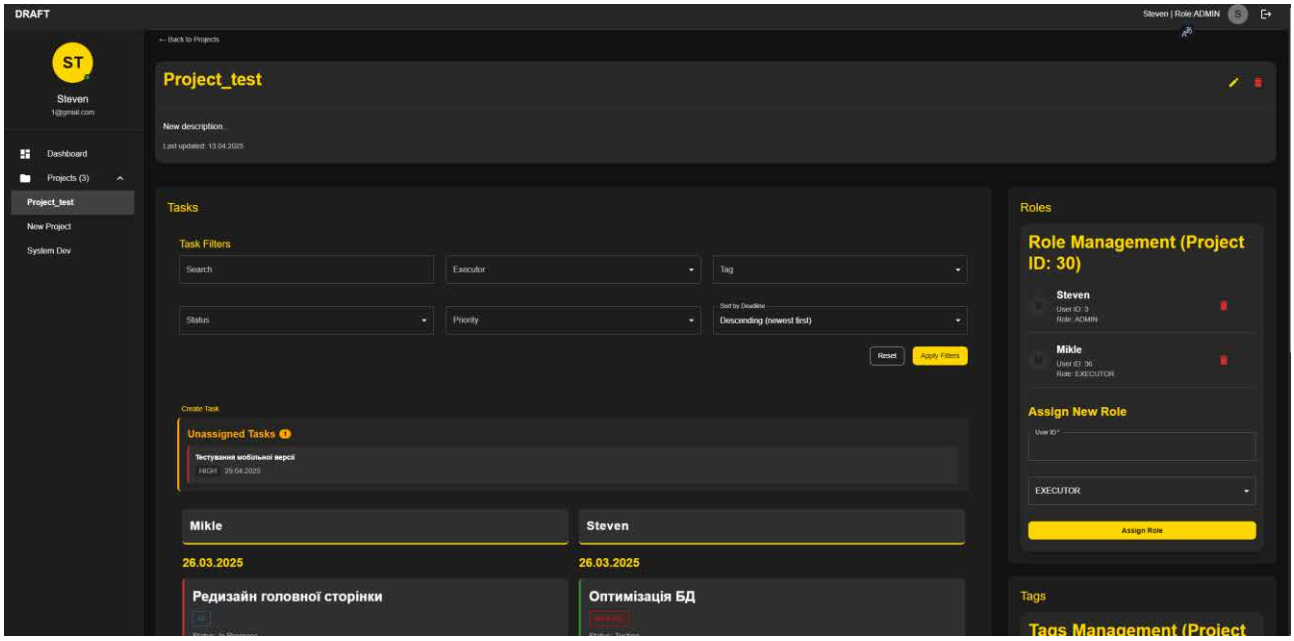


Рис 4.1.6 Стрінка проєкту (1)

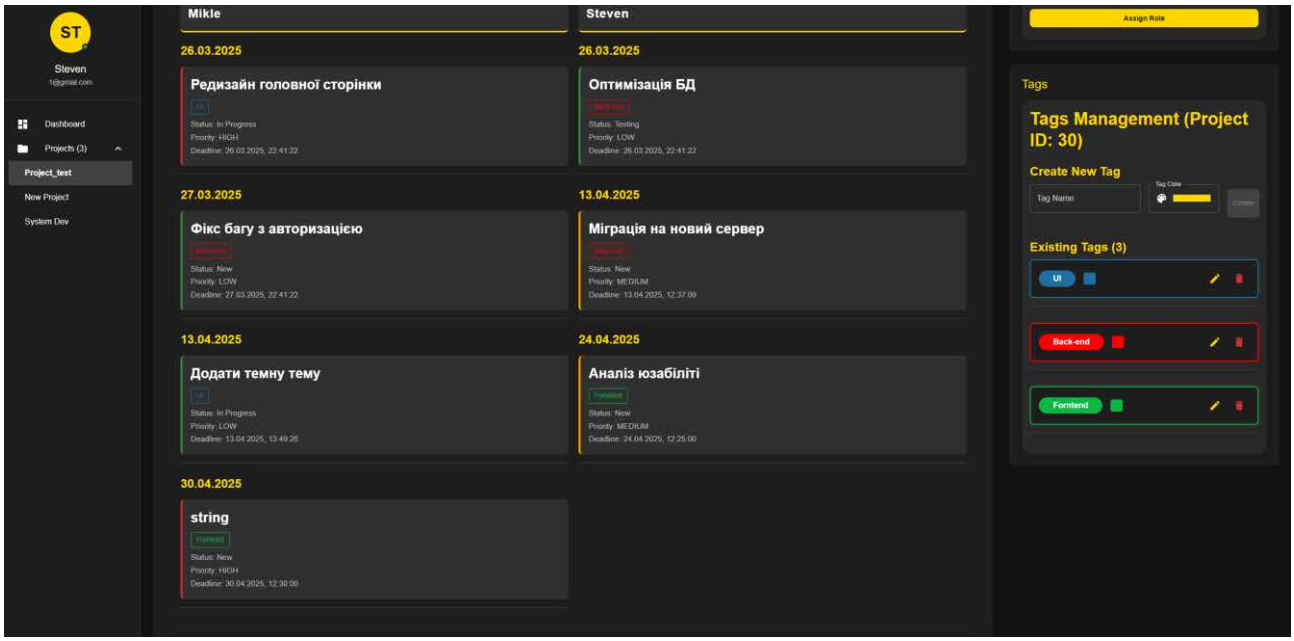


Рис 4.1.7 Стрінка проєкту (2)

Модальне вікно (Рис 4.1.8) призначене безпосередньо для створення або редагування існуючих, раніше створених завдань у системі, а також їх видалення. Воно містить форму з усіма необхідними полями для налаштування параметрів завдання. Обов'язкові поля (позначені зірочкою): Назва завдання Рівень пріоритету Статус виконання Додаткові параметри: поле для опису завдання, призначення виконавця, категорія/мітка, дати початку та завершення. Кнопки управління: видалити завдання, скасувати зміни, зберегти оновлення

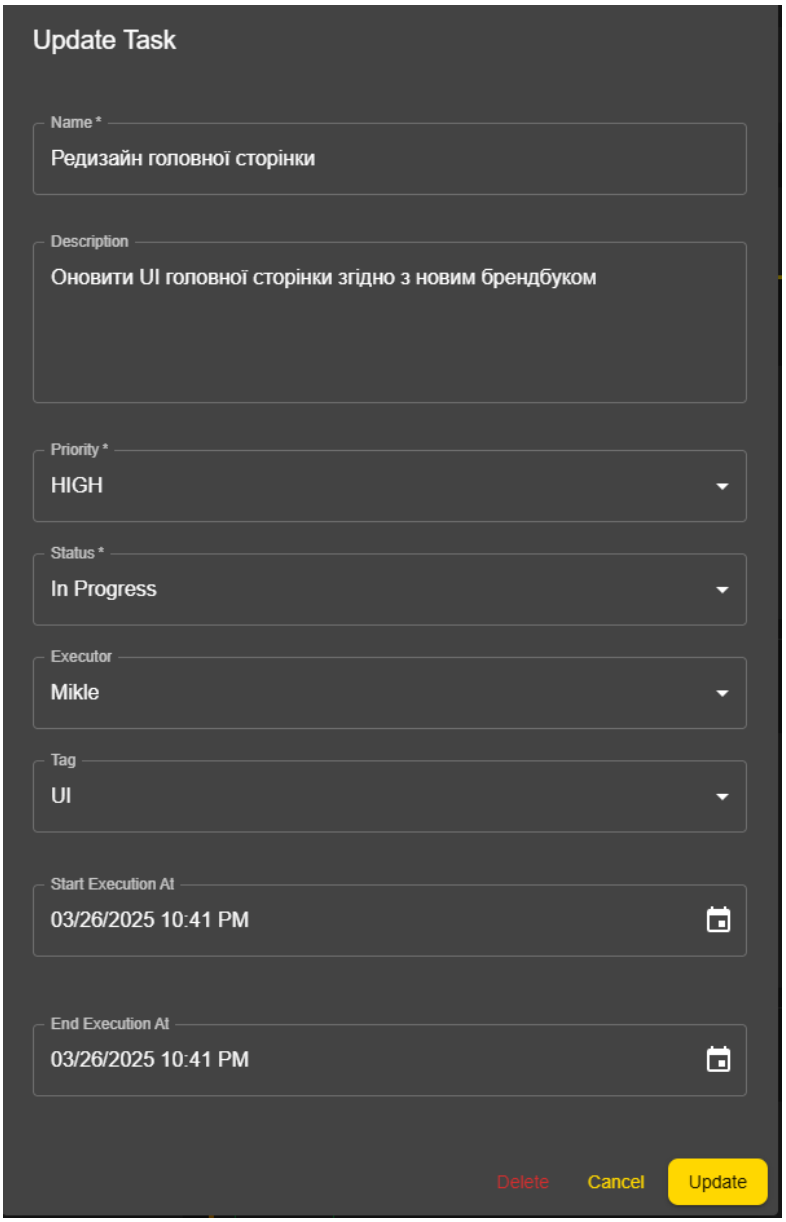


Рис Рис 4.1.8 Модальне вікно редаугвання завдань

4.2. Апаратні та програмні вимоги до роботи системи.

Апаратні та програмні вимоги до системи менеджера завдань

1. Вимоги до хостингу (серверної частини)

Апаратні вимоги вимоги до сервера

Мінімальні:

- Процесор: 2 ядра, 2.5+ GHz (Intel Xeon, AMD Ryzen)

- Оперативна пам'ять: 4 ГБ RAM
- Дисковий простір:
- 20 ГБ SSD
- Мережа: 100 Mbps

Рекомендовані (для комерційного використання):

- Процесор: 4+ ядра, 3.0+ GHz
- Оперативна пам’ять: 8+ ГБ RAM
- Дисковий простір: 50+ ГБ SSD (або NVMe)
- Мережа: 1 Gbps
- Резервне копіювання: регулярні бекапи (щонайменше раз на день)

Програмні вимоги

Операційна система: Ubuntu 22.04 LTS

Розгортання бекенду:

- .NET 8
- ASP.NET Core Web API
- Entity Framework Core

База даних: PostgreSQL 15

Система управління пакетами: Yarn (для фронтенду)

Додаткові сервіси:

- Nginx (зворотний проксі для API та фронтенду)

Вимоги до браузера клієнта

Апаратні вимоги

Мінімальні

- Процесор: Intel Core i3 / AMD Ryzen 3
- Оперативна пам’ять: 4 ГБ RAM
- Графічний процесор: інтегрований (Intel UHD 620 або еквівалент)
- Роздільна здатність екрану: 1366x768
- Мережа: 10 Mbps

Рекомендовані

- Процесор: Intel Core i5 / AMD Ryzen 5
- Оперативна пам’ять: 8 ГБ RAM

- Графічний процесор: будь-який сучасний
- Роздільна здатність екрану: 1920x1080 або вище
- Мережа: 50+ Mbps

Програмні вимоги

Операційна система:

- Windows 10 / 11
- Linux (Ubuntu 22.04 LTS, Fedora 39)

Підтримувані браузери:

- Google Chrome (остання версія)
- Mozilla Firefox (остання версія)
- Microsoft Edge (Chromium)

4.3. Інсталяційний пакет системи.

Інсталяційний пакет інформаційної системи включає в себе абсолютно всі необхідні компоненти для швидкого та зручного розгортання програмного забезпечення у середовищі, Linux чи Windows. Система реалізована двома частинами. Серверна частини на ASP.NET Core та клієнт частини, відповідно, React. Для забезпечення портативності та простоти розгортання застосовується технологія контейнеризації програм, Docker.

У складі інсталяційного пакету надані ось такі основні компоненти:

Скомпільована та готова для інтеграції серверна частина ASP.NET Core Web API розташована у директорії backend/. Вона вміщує в себе всі необхідні файли для виконання такі як *.dll, web.config, інші конфігураційні файли – це appsettings.json, appsettings.Production.json та статичні ресурси, зокрема зібраний фронтенд, інтегрований у підпапку wwwroot/.

- Фронтенд частина, розміщена в папці frontend/ яка вміщує в собі всю зібрану статику, такі файли React-додатку як ndex.html, папка static/, manifest.json тощо.
- Dockerfile для кожного сервісу, це для API та фронтенду, який визначає основні правила для збірки всіх відповідних контейнерів.
- Файл docker-compose.yml, що ініціює створення загальної інфраструктуру системи. API, клієнтський інтерфейс, та базу даних, з вказанням портів, залежностей та інших змінних інформаційного середовища.
- Файл .env, у якому задані конфіденційні налаштування програми. Зокрема такі як строки для виконання підключення з’єднання до БД, секретні ключі для створення та дешифрування токенів автентифікації, порти тощо.
- Скрипт ініціалізації бази даних — init.sql, що містить SQL-команди для визначення структури табличок та внесення даних.
- README.md — інструкція, яка реалізує детальний опис всього процесу розгортання за допомогою Docker. А також інші абсолютно типові команди для контейнерів.

Відповідно після цього буде автоматично зібрано та запущено всі ці сервіси, і далі система стане доступною за вказаною адресою в браузері. Відбувається процес інсталяції пакетів що дозволяє розгорнути інформаційну систему швидко, без додаткових проблем або складного налаштування середовища, що дає можливість для зручного впровадження та тестування системи, повний вміст файлу описано в таблиці нижче.

Таблиця 1

Файл / Папка	**Призначення**
backend/ProjectManagementSystem.dll	Скомпільоване ядро ASP.NET Web API
backend/wwwroot/	Містить зібраний фронтенд (React build)
frontend/	Окремо зібраний фронтенд
backend/appsettings.Production.json	Налаштування для продакшену (рядки підключення, токени)
backend/Dockerfile	Docker-образ для запуску API
frontend/Dockerfile	React збирається окремо, то йде окремий контейнер
database/init.sql	Створення структури бази даних
docker-compose.yml	Автоматичний запуск усієї системи
.env	Налаштування середовища (паролі, адреси БД тощо)
README.md	Інструкція для встановлення і запуску

ВИСНОВКИ

У даному дипломній роботі була розроблена і представлена Інформаційна система курування проєктами. На меті диплому є забезпечення потенціалу для використання в сучасних організаційних, виробничих та освітніх середовищах. Цей програмний продукт дає досить широкий спектр інструментів для кращого планування, в управлінні процесу реалізації та інтеграції проєктів та його задач. У свою чергу це може вирішити проблеми в процесі оптимізації командної роботи. Це дає можливість підвищити продуктивність для досягнення поставлених цілей у визначені терміни. В ході виконання дипломної роботи було досягнуто наступні цілі:

Розроблена система дозволяє користувачам виконувати створення нових проєктів, згідно до них призначати виконавців з певними ролями. Формувати завдання всередині кожного проєкту. Встановлювати в середині дедлайни та пріоритети, також можна слідкувати за поточним статусом виконання завдань. Завдяки цьому користувачі можуть чітко розподіляти обов’язки, що уникати дублювання робіт і забезпечувати прозорість у процесах розробки.

Візуальний інтерфейс платформи розроблено з дотримання всіх сучасних стандартів. Це інтуїтивно зрозумілим і дружнім для користувача. Він надає просту навігацію між модулями на сторінці. Це дозволяє швидко отримувати доступ до всієї потрібної інформації, переглядаючи прогрес завдань, взаємодіяти з іншими учасниками проєкту та максимально оперативно реагувати на зміни.

Для візуалізації даних реалізовано систему відображення активностей у дешборді, з можливістю фільтрації за тегами, статусами та іншими атрибутами.

Система має підтримку гнучкої моделі ролей. Кожному користувачу можна надати одну, визначену роль із списку доступних в межах конкретного проєкту.

Відповідно, це дозволяє чітко обмежувати доступ до функціоналу залежно від призначеної ролі та уникати несанкціонованого доступу до конфіденційною інформації кожного завдання та проекту.

Для збереження інформації були створені наступні таблиці в БД такі як: "Проєкт", "Завдання", "Користувач", "Роль" та "Тег". Вони логічно пов’язані між собою. Це дозволяє ефективно управляти інформацією в них, забезпечуючи цілісність даних що дає можливість для розширення.

Отже, в ході виконання дипломної роботи була створена інформаційна система для керування проєктами. Вона є гнучким, функціональним та масштабованим інструментом з модульною архітектурою, яка може бути використана у найрізноманітніших сферах діяльності під час розробки та для організації роботи в команді для управління ресурсами та досягнення цілей компаній.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Нильсен Я., Лоранжер Х. Web-дизайн: зручність використання Web-сайтів. Мінськ : Гревцова, 2021. С. 322–329.

2. Newman S. Building Microservices. 1st ed. Sebastopol, CA : O'Reilly Media, Inc., 2015. 280 с.

3. Fowler M. Monolith First [Електронний ресурс]. 2024. Режим доступу: https://martinfowler.com/bliki/MonolithFirst.html#footnotetypical-monolith

4. Романюк О. Н., Савчук Т. О. Організація баз даних і знань : навч. посіб. Вінниця : УНІВЕРСУМ-Вінниця, 2023. 217 с.

5. Bollig B. Message Sequence Charts. Formal Models of Communicating Systems. Berlin : Springer, 2006. 220 p.

6. Rumbaugh J., Jacobson I., Booch G. The unified modeling language reference manual. 2nd ed. Boston : Addison-Wesley, 2021. 721 с.

7. Уотсон К., Нейгел К., Педерсен Я. та ін. Visual C# 2008: базовий курс. М. : И.Д. Вільямс, 2020. 1216 с.

8. Bertoli M. React Design Patterns and Best Practices: Build easy to scale modular applications using the most powerful components and design patterns. Birmingham : Packt Publishing, 2021. 318 с.

9. Загальна інформація ASP.NET Core MVC [Електронний ресурс]. Режим доступу:https://learn.microsoft.com/enus/aspnet/core/mvc/overview?view=aspnetcore-6.0

10. Entity Framework Documentation [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/ef

11. Меєр Л. CSS Secrets: Кращі рішення для повсякденних проблем веб-дизайну. Київ : Видавництво «Фабула», 2023. 280 с.

12. Крокфорд Д. JavaScript: Хороші частини. СПб. : Символ-Плюс, 2020. 176 с.

13. JavaScript документація [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

14. Фріман А. Pro React 16. Нью-Йорк : Apress, 2021. 500 с.

15. Абрамов М., Бенкс А. Redux in Action. Нью-Йорк : Manning Publications, 2018. 375 с.

16. Фріман А. Pro ASP.NET Core MVC 2. Нью-Йорк : Apress, 2022. 700 с.

17. Лерман Д. Programming Entity Framework: Code First. Sebastopol, CA : O'Reilly Media, 2021. 280 p.

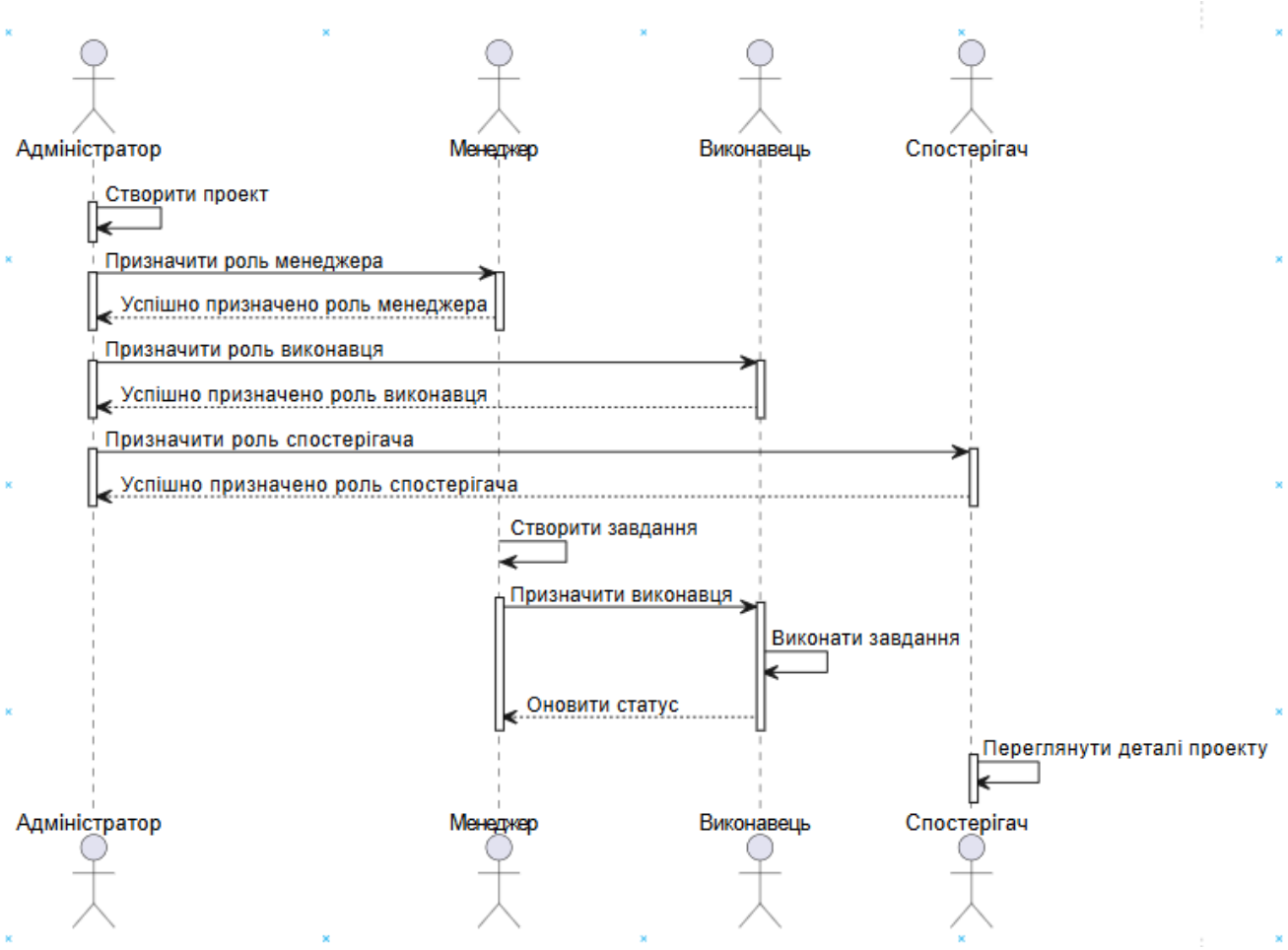
18. Розенберг Б. Programming TypeScript. Sebastopol, CA : O'Reilly Media, 2019. 250 с.

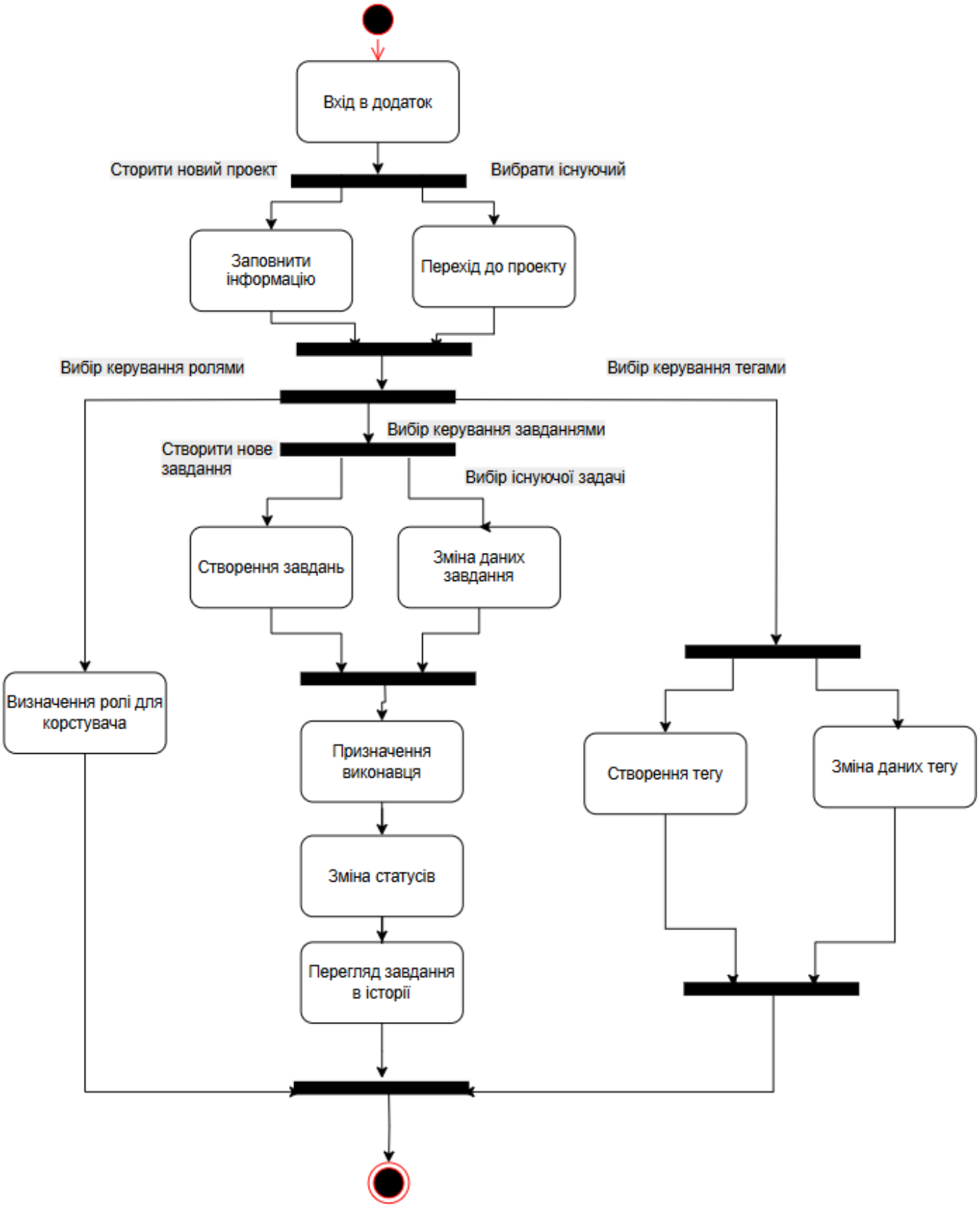
19. Крьюг С. Don't Make Me Think: A Common Sense Approach to Web Usability. 3rd ed. San Francisco : New Riders, 2023. 216 с.

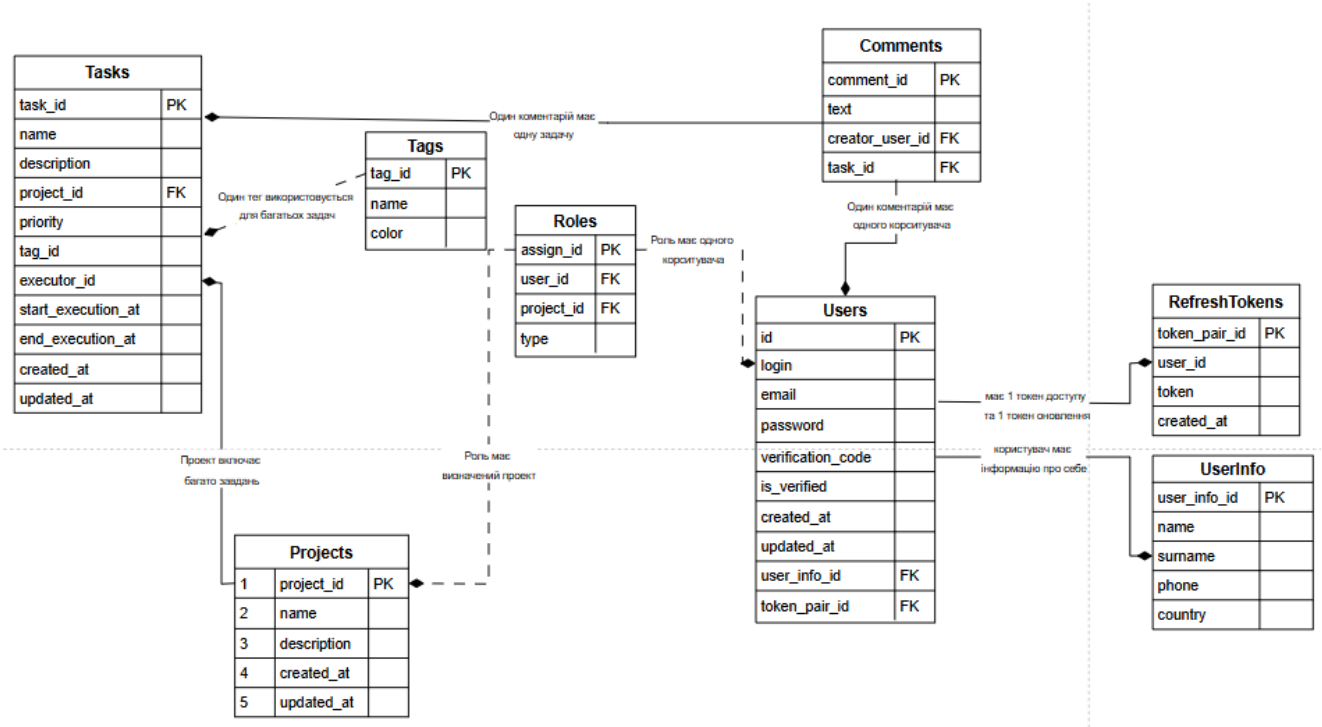
Додаток А

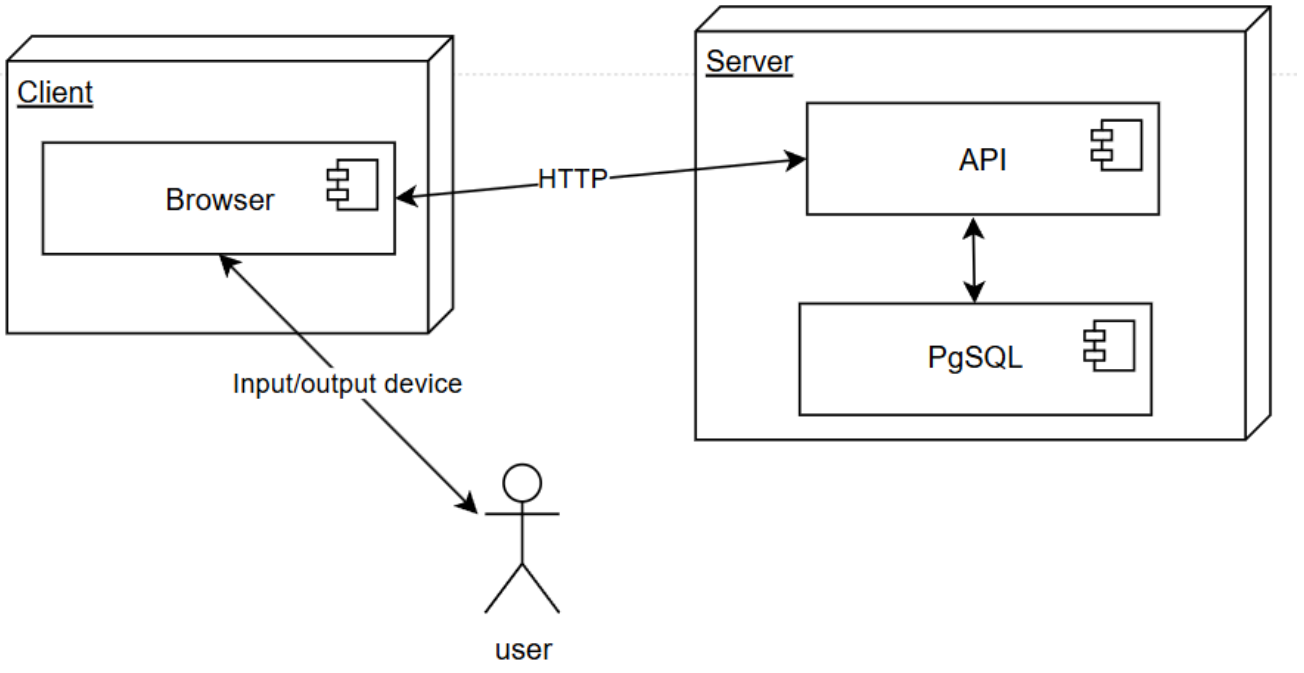
Посилання на проект на ресурсі GitHub: https://github.com/mcMackosh/Draft

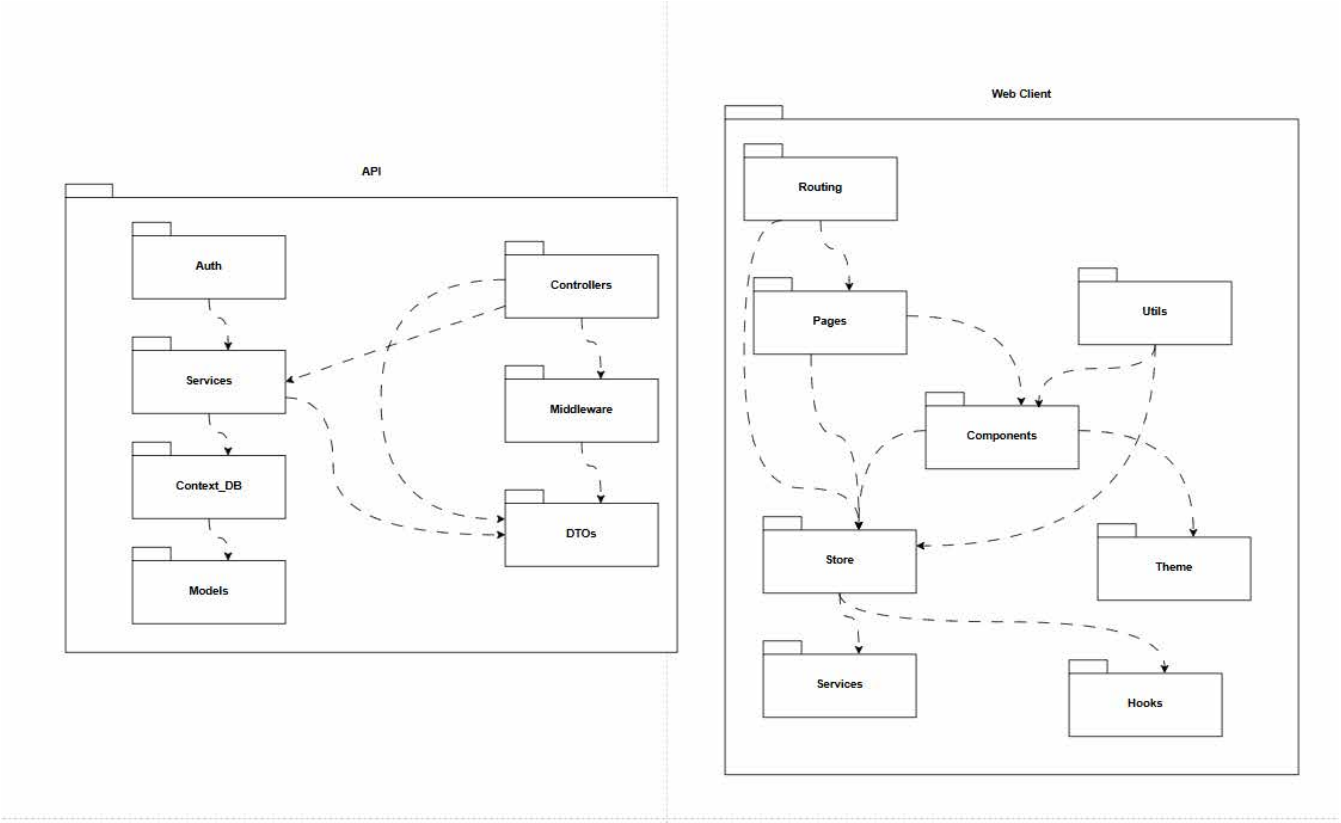
Додаток Б

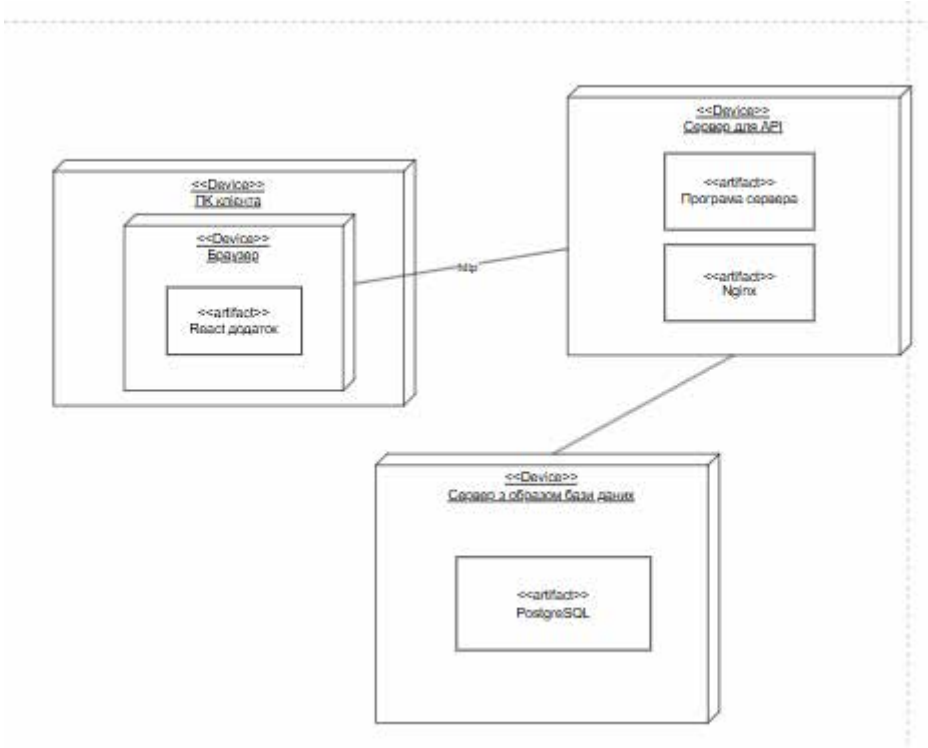

Адміністратор
Менеджер
Виконавець
Спостерігач
Створити проект
Призначити роль менеджера
Успішно призначено роль менеджера
Призначити роль виконавця
Успішно призначено роль виконавця
Призначити роль спостерігача
Успішно призначено роль спостерігача
Створити завдання
Призначити виконавця
Виконати завдання
Оновити статус
Переглянути деталі проекту
Адміністратор
Менеджер
Виконавець
Спостерігач


Вхід в додаток
Сторити новий проект
Вибрати існуючий
Заповнити інформацію
Перехід до проекту
Вибір керування ролями
Вибір керування тегами
Вибір керування завданнями
Створити нове завдання
Вибір існуючої задачі
Створення завдань
Зміна даних завдання
Визначення ролі для корстувача
Призначення виконавця
Зміна статусів
Перегляд завдання в історії
Створення тегу
Зміна даних тегу


Tasks
task_id PK
name
description
project_id FK
priority
tag_id
executor_id
start_execution_at
end_execution_at
created_at
updated_at
Comments
comment_id PK
text
creator_user_id FK
task_id FK
Один коментарій має одну задачу
Один коментарій має одного користувача
Tags
tag_id PK
name
color
Один тег використовується для багатьох задач
Roles
assign_id PK
user_id FK
project_id FK
type
Роль має одного користувача
Users
id PK
login
email
password
verification_code
is_verified
created_at
updated_at
user_info_id FK
token_pair_id FK
має 1 токен доступу та 1 токен оновлення
RefreshTokens
token_pair_id PK
user_id
token
created_at
користувач має інформацію про себе
UserInfo
user_info_id PK
name
surname
phone
country
Проект включає багато завдань
Роль має визначений проект
Projects
1 project_id PK
2 name
3 description
4 created_at
5 updated_at


Client
Browser
HTTP
Server
API
PgSQL
Input/output device
user


API
Auth
Controllers
Services
Middleware
Context_DB
DTOs
Models
Web Client
Routing
Pages
Utils
Components
Store
Theme
Services
Hooks


<<Device>>
ПК клієнта
<<Device>>
Браузер
<<artifact>>
React додаток
http
<<Device>>
Сервер для API
<<artifact>>
Програма сервера
<<artifact>>
Nginx
<<Device>>
Сервер з образом бази даних
<<artifact>>
PostgreSQL