НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення системи обліку виконання виробничих завдань створення програмних систем»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

к.т.н, доцент _____ Вайганг Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкін Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Глушко О.О.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____Голуб Б.Л.
_" "_____20 р._

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Глушку Олександру Олеговичу

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: «Програмне забезпечення системи обліку виконання виробничих завдань створення програмних систем»

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2249 «С».

Термін подання завершеної роботи на кафедру _____

Вихідні дані до бакалаврської кваліфікаційної роботи: законодавчі та нормативні документи, фінансова звітність та статутні відомості об'єкта дослідження, наукові статті, монографії тощо за темою бакалаврської дипломної роботи

Перелік питань , які потрібно розробити: Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкін Г.О

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Глушко О.О.

(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....5
ВСТУП.....6
РОЗДІЛ 1.....9
1.1. Опис предметної області.....9
1.2. Огляд інформаційних джерел та існуючих рішень.....13
1.3. Аналіз вимог до програмної системи.....18
1.3.1. Бізнес вимоги.....19
1.3.2. Функціональні вимоги.....19
1.3.3.Нефункціональні вимоги.....21
1.4. Моделювання предметної області.....22
1.5. Висновок до розділу 1.....26
РОЗДІЛ 2.....27
2.1. Логічна модель даних у вигляді ER-діаграми.....27
2.2. Діаграми класів та кооперацій.....29
2.3. Діаграма пакетів.....31
2.4. Діаграми станів.....34
2.5. Діаграма компонентів.....36
2.6. Висновок до розділу 2.....38
РОЗДІЛ 3.....40
3.1. Система управління інформаційною базою.....40
3.2. Вибір інструментарію для створення прикладного програмного забезпечення.....42
3.3. Розробка інформаційної бази.....44
3.4. Алгоритмізація та програмування програмних модулів.....49
3.5. Висновок до розділу 3.....56
РОЗДІЛ 4.....57
4.1. Тестування системи.....57
4.2. Вимоги до апаратного та програмного забезпечення.....59
4.3. Склад інсталяційного пакету.....61
4.4. Висновок до розділу 4.....64

ВИСНОВОК.....65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 68
ДОДАТОК А. Код серверної частини.....69
Додаток Б. Код тестування логіки серверної частини.....75
Додаток В. Код клієнтської частини..... 82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення – Software

БД – база даних – Database

HTTP – Hypertext Transfer Protocol

API – Application Programming Interface

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JS – JavaScript

UUID – Universally Unique Identifier

ORM – Object-Relational Mapping

JWT – JSON Web Token

JSON – JavaScript Object Notation

JWS – JSON Web Signature

ВСТУП

У сучасних умовах, коли бізнес-процеси стають дедалі більш комплексними та вимагають високого рівня організації і результативності, особливо важливо використовувати ефективні засоби для управління проектною діяльністю. Різноманітні платформи й сервіси, призначені для керування проектами, дають змогу здійснювати планування, моніторинг і контроль робочих процесів на всіх стадіях життєвого циклу проекту. Це дозволяє компаніям досягати поставлених результатів швидше, раціонально використовуючи наявні ресурси. Такі системи забезпечують можливості для призначення завдань, відстеження їх виконання, контролю часу і бюджету, а також сприяють покращенню внутрішньої комунікації в командах.

Створення власного програмного забезпечення для обліку виконання виробничих завдань набуває особливої важливості у випадку з великими організаціями, державними структурами чи приватними підприємствами, які мають специфічні вимоги до функціоналу. Існуючі рішення не завжди дозволяють повністю адаптуватися під ці індивідуальні потреби. У зв’язку з цим обрана тема — «Програмне забезпечення системи обліку виконання виробничих завдань створення програмних систем» — є як актуальною, так і практично значущою, оскільки відкриває можливості для аналізу та реалізації гнучких рішень, орієнтованих на конкретного користувача або організацію.

Об’єктом дослідження виступає програмне забезпечення системи обліку виконання виробничих завдань створення програмних систем. Дослідження об’єкта включає аналіз функціональних і нефункціональних вимог, архітектурних рішень, засобів реалізації, а також способів забезпечення точності, надійності та зручності використання такого програмного забезпечення. До об’єкту дослідження також відносяться різноманітні моделі і методи розробки, технології, бібліотеки, підходи до проектування і розробки програмного забезпечення, які застосовуються для створення таких додатків. Програмний додаток створений після дослідження має відповідати всім

необхідним вимогам, а також сучасним стандартам розробки та засобів забезпечення захисту даних.

Предмет дослідження – це облік виконання виробничих завдань створення програмних систем. Методи обліку завдань у сфері розробки програмного забезпечення, розподіл між учасниками, контроль статусу виконання та своєчасне оновлення. Облік завдань виконує ключову роль у робочому процесі, адже він дозволяє зрозуміти, що вже зроблено, що ще в роботі, а що потребує втручання. Саме це явище — як організована, впорядкована діяльність з ведення й контролю завдань — і є предметом дослідження.

Завдання дослідження - це опис проблематики, обґрунтування вибору інструментальних засобів, створення технічного завдання, аналіз предметної області та основних складових системи, проектування та архітектура програмного забезпечення, розробка та реалізація програмного продукту. Необхідно проаналізувати сучасні підходи до обліку виробничих завдань у сфері розробки програмного забезпечення, дослідити та визначати функціональні та нефункціональні вимоги до такої системи. Наступним кроком, необхідно визначити архітектуру системи та розробити інформаційну модель, що буде описувати структуру та зв’язки між елементами такої системи. Також необхідно виконати проектування користувацького інтерфейсу з урахуванням принципів зручності та реалізувати відповідне програмне забезпечення. В кінці необхідно провести тестування розробленого рішення, та виправити помилки, якщо такі присутні. В разі успішного тестування програмного забезпечення, останнім кроком є введення системи в експлуатацію.

Під час виконання кваліфікаційної роботи буде проведено підготовчу роботу, проаналізовано існуючі джерела, що можуть стосуватися до розробки такого додатку. Будуть проаналізовано та визначено вимоги та бізнес потреби, до вимог включаються функціональні, не функціональні, вимоги до інтерфейсу, до безпеки тощо. Буде розроблено програмне забезпечення. До

розробки програмного забезпечення входить проектування архітектури та інтерфейсу, розробка програмного коду. Останнім кроком буде створено висновок, який підсумує всю пророблену роботу, всі результати досліджень та розробку програми.

Під час створення програмного додатку будуть застосовані сучасні технології, які надають весь спектр необхідних можливостей для створення подібного роду систем. Для зберігання даних використовуватиметься база даних PostgreSQL, яка є потужною об’єктно-реляційною системою управління базами даних з відкритим кодом. В якості основної мови програмування для обробки логіки буде виступати C# 8 версії, та необхідні бібліотеки з платформи .Net. Що стосується створення візуальної частини – за це будуть відповідати HTML, CSS та фреймворк React, з різними бібліотеками екосистеми JS.

В якості апробації було наведено тези на навчальній конференції НУБІП.

Робота складається з 94 сторінок, 31 малюнків, 2 таблиць та 11 бібліографічних найменувань.

РОЗДІЛ 1

1.1. Опис предметної області

На сьогодні дедалі більше бізнесів як в Україні, так і за її межами переходять до проектно-орієнтованої моделі ведення діяльності. Вітчизняний ринок також демонструє цю тенденцію — близько половини компаній працюють у рамках проектного підходу, і навіть аграрна сфера активно впроваджує проектне управління у свої процеси. Така еволюція зумовлена прагненням підприємств створювати інноваційні рішення, покращувати вже існуючі продукти або досягати якісно нових результатів у знайомих напрямах. Сьогодні поняття "проект" вийшло за межі традиційного уявлення як пакету документів — це вже стратегічна діяльність, спрямована на досягнення важливих бізнес-результатів.

Успіх компанії сьогодні значною мірою залежить від здатності ефективно реалізовувати проекти. Тому облік виконання завдань в проектах є одною з ключових цілей для менеджерів усіх рівнів. У цьому контексті важливо розуміти, що таке проект і які інструменти та методи застосовуються для його успішного виконання.

Проекти є невід'ємною частиною діяльності будь-якої організації. Кожна компанія має свою стратегію розвитку, яка включає цілий ряд проектів, необхідних для досягнення стратегічних цілей.

Управління проектами забезпечує ефективне та швидке досягнення поставлених цілей і включає комплекс методів, що оптимізують розподіл ресурсів та сприяють досягненню загальних цілей компанії.

Системи обліку виконання виробничих завдань стають особливо корисними, коли проекти мають схожий характер або структуру. Вони дозволяють створювати цілісне уявлення про кожен проект, контролювати його етапи на різних рівнях, стежити за бюджетом та дотриманням дедлайнів.

Ці системи виступають як важливий інструмент у розпорядженні керівника та команди, сприяючи підвищенню прозорості усіх процесів

компанії. Завдяки їм кожен учасник проекту має доступ до інформації про часові та фінансові витрати, а також може оцінити власне та колективне навантаження.

Такі системи виконують функції спрямовані на автоматизації обліку виконання виробничих завдань, а також оптимізацію різних процесів у цій області, таких як обмін інформацією між зацікавленими особами, контроль за ходом виконання завдань та проекту в цілому, наочне представлення інформації.

Обмін інформацією між зацікавленими особами відбувається за допомогою локального серверу, або вбудованого в систему чату. Що підвищує ефективність комунікації команди і забезпечує централізовану точку для зберігання даних, що дає змогу контролювати доступ до інформації, її безпеку та конфіденційність.

Контроль за ходом виконання завдань надає поточну інформацію про стан виконання виробничих завдань, та використаних ресурсів. Завдання можуть бути позначені як "заплановано", "в процесі", "завершено", що дає змогу чітко побачити на якому етапі знаходиться кожне завдання.

Також подібного роду системи використовують засоби формування звітів для наочного відображення інформації, використовуючи різноманітні графіки, діаграми, календарні плани. Це важливий інструмент для прийняття обґрунтованих рішень і аналізу результатів. Завдяки таким звітам і візуалізації даних, керівники проектів і команди мають змогу оперативно реагувати на зміни, коригувати стратегії та забезпечувати виконання завдань у визначені терміни. Це підвищує загальну ефективність роботи й дозволяє краще планувати майбутні етапи проекту.

Професійні системи обліку виконання виробничих завдань, призначені для досвідчених менеджерів і допомагають керувати кількома складними проектами одночасно. Вони містять інструменти для планування, аналізу та контролю виконання проектів, а також забезпечують ефективну комунікацію

між членами команд і можуть інтегруватися з іншими управлінськими системами.

Не можна говорити про системи обліку виконання виробничих завдань у сфері створення ПЗ без згадування методологій управління проектами. Методологія управління проектом – це набір правил, принципів та методів роботи над проектом. Вона описує як почати проект, та як працювати над ним до самого кінця.

Існує два основних типи методологій: жорсткі та гнучкі.

Що стосується жорстких методологій, основним її представником є водоспадна модель. Це модель ЖЦПЗ, яка складається з певної кількості кроків. Перехід на наступний крок відбувається лише після того, як буде завершена робота на попередньому кроці, також варто сказати, що повернення на попередні кроки не передбачається. Вимоги до програмного забезпечення, які визначаються на етапі формування вимог, документуються та залишаються незмінними протягом усього процесу розробки. На рисунку 1.1. зображено приклад такої моделі.

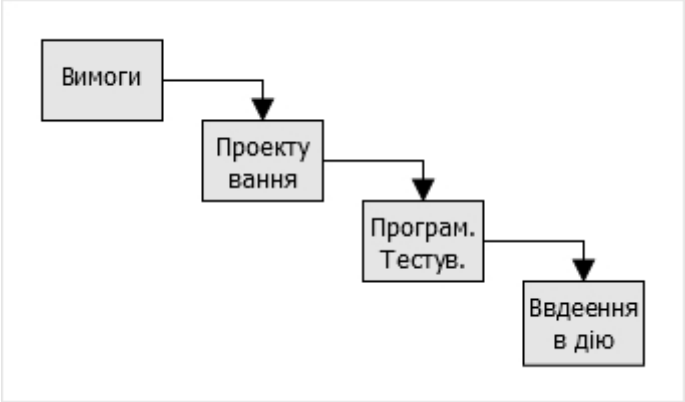

Рис. 1.1. Приклад водоспадної моделі життєвого циклу

До гнучких методологій відносяться Scrum, Lean, Agile, і тому подібні. відмінність гнучких методологій він попередньої полягає в тому, що для команди передбачена можливість дещо змінити у проєкті на проміжних стадіях. На рисунку 1.2. зображено схематичну схему методогії Agile.

Рис.1.2. Схема методології Agile

Також вартий згадування Kanban метод. Kanban — це методологія управління проектами та процесами, що ґрунтується на візуалізації роботи, обмеженні незавершених завдань та безперервному вдосконаленні. Його головною метою є покращення ефективності робочих процесів шляхом мінімізації часу виконання завдань і підвищення прозорості процесів. Kanban є частиною гнучких (Agile) методологій. Дошка задач, яка є основним елементом в цій методології, дає чітке уявлення про статус виконання завдань, що дозволяє швидко виявляти проблеми. Також всі учасники проекту мають доступ до інформації про стан завдань, що дозволяє більш ефективно координувати діяльність та приймати рішення.

Велика кількість систем для управління обліком виконання виробничих завдань у сфері розробки програмного забезпечення використовує цю методологію. Trello — один із найбільш популярних інструментів для візуалізації процесів на основі методу Kanban. Він дозволяє створювати дошки з картками для завдань, що можуть переміщатися через різні етапи виконання (наприклад, "Завдання", "В роботі", "Завершено"). Jira від Atlassian — це потужний інструмент для управління проєктами, який підтримує методології Scrum, Kanban і Agile загалом. Jira дозволяє створювати Kanban-дошки, на яких можна відслідковувати завдання, обмежувати незавершені завдання на кожному етапі і аналізувати потік робіт. Asana — ще один популярний

інструмент для управління завданнями, який дозволяє створювати Kanban - дошки для відслідковування виконання завдань. Asana дозволяє автоматизувати деякі етапи роботи, встановлювати терміни виконання і інтегрувати інші інструменти.

Підсумовуючи, можна сказати, що впровадження систем обліку виконання виробничих завдань дозволяє підвищити ефективність комунікацій команди, оптимізувати робочі процеси, та забезпечити прозорість виконання проектів. Серед методологій, що активно використовуються для управління проектами, гнучкі методи, зокрема Kanban, набули великої популярності завдяки своїй гнучкості та здатності підвищувати ефективність робочих процесів. Такі програмні системи та методології управління проектами пов’язані між собою та йдуть поруч. Методології управління проектами надають структуру і принципи, за якими ці системи функціонують, визначаючи основні правила для планування, виконання і моніторингу проектів.

1.2. Огляд інформаційних джерел та існуючих рішень

Першою на черзі є програма Trello, це веб-орієнтована система, яка по словам розробників, допомагає зібрати всіх співробітників, задачі і інструменти в одному місці. Розробником виступає компанія Atlassian – австралійська компанія, розробник програмного забезпечення для управління розробкою програмного забезпечення. На рисунку 1.3. приклад «типової дошки задач» Trello.

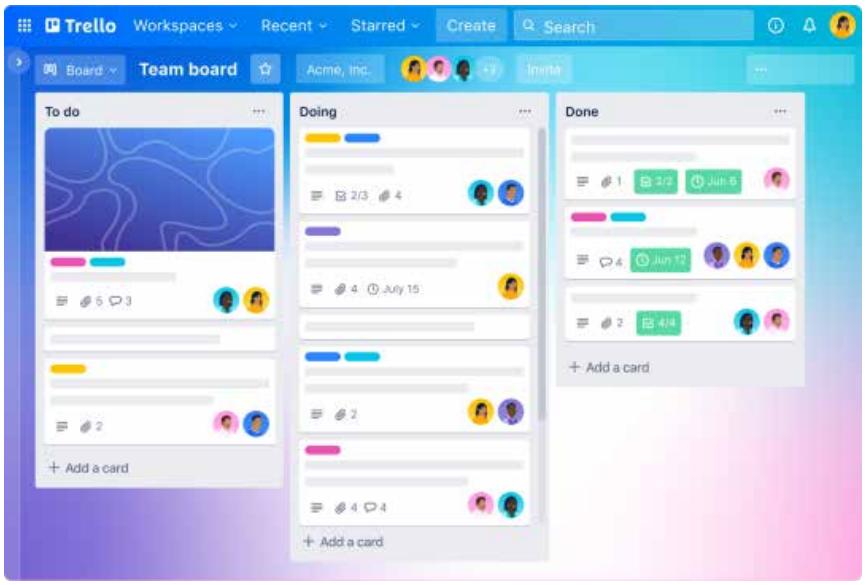

Рис. 1.3. Приклад дошки задач в Trello

Дякуючи дошкам Trello, завдання впорядковуються, і робота рухається вперед. Достатньо одного погляду, щоб побачити всі завдання, від запланованих до виконаних.

Різні етапи виконання завдання. Можна почати з простих етапів («Потрібно зробити», «В роботі», «Готово») або створити власний робочий процес, що ідеально відповідає потребам вашої команди.

Картки представляють завдання та ідеї і містять всю інформацію, необхідну для виконання роботи. Під час виконання завдань можна перетягувати картки між колонками, щоб змінити їх статус.

Також в Trello присутній календар, і можливість назначати терміни завдання.

Що стосується сильних сторін даної системи, це те, що Trello простий в використанні, його інтерфейс легкий та інтуїтивно зрозумілий, що дозволяє користувачам швидко орієнтуватися в системі. Користувачі можуть адаптувати Trello до власних потреб, створюючи різні дошки та визначаючи власні правила для організації роботи. Trello підтримує багато інтеграцій з іншими популярними сервісами та інструментами, що дозволяє розширити його можливості. Для багатьох користувачів доступна безкоштовна версія Trello з достатньою кількістю функцій для ефективного використання.

Мінуси даної системи: обмежені функції в безкоштовній версії, недостатня можливість структурування завдання, не підходить для складних проектів з великою кількістю завдань.

Наступним кроком було проаналізовано програмний додаток під назвою Asana. Розробник Asana, Inc. - американська компанія-розробник програмного забезпечення, розташована в Сан-Франциско. Її флагманський сервіс Asana є веб та мобільною платформою "управління роботою", призначеною для допомоги командам у організації, відстеженні та управлінні своєю роботою. Ця програмна система надає можливості для створення, відслідковування задач, вона також надає можливість перегляду в різних зручних для користувача формах, таких як таблиця, дошка задач, хронологія, календар, та панель моніторингу. На рисунку 1.4. наведено вікно програми Asana.

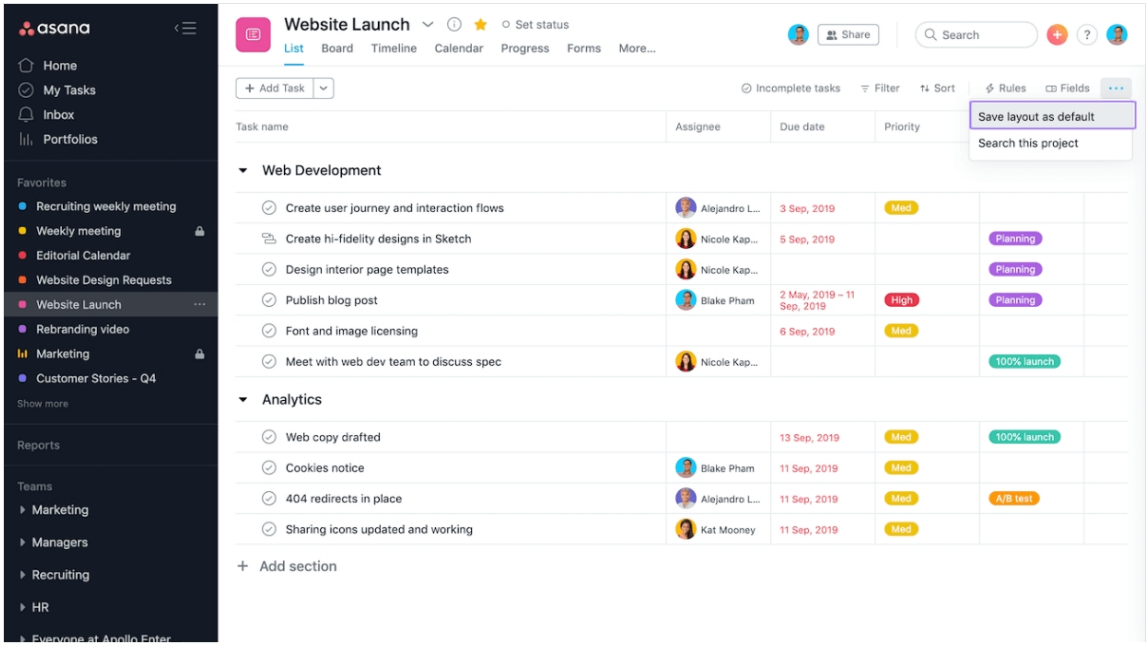

Рис.1.4. Вікно програми Asana

Переваги Asana:

- Зручне управління завданнями. Платформа дозволяє легко створювати проекти, деталізувати завдання, призначати відповідальних осіб і встановлювати дедлайни. Це забезпечує чітку структуру й контроль над робочим процесом.

- Командна взаємодія. Завдяки функціоналу спільної роботи, користувачі можуть спільно редагувати завдання, залишати коментарі, обмінюватися ідеями та координувати дії безпосередньо в системі.
- Моніторинг виконання. Asana надає інструменти для візуального відображення прогресу: можна бачити, на якому етапі знаходиться кожне завдання, що вже завершено, а що потребує уваги.
- Календар і табличний вигляд. Для зручності планування доступні різні режими перегляду – календарний та табличний, що дозволяє швидко зорієнтуватися у термінах та обсягах роботи.
- Визначення пріоритетів. Платформа підтримує можливість виставлення пріоритетів, що дає змогу фокусуватися на найбільш важливих чи термінових завданнях.

Недоліки Asana:

- Обмеження безкоштовної версії. У базовому тарифі існують обмеження як по кількості учасників команди, так і по доступному функціоналу, що може стати перешкодою для масштабних проєктів.
- Перевантаженість можливостями. Через багатий набір функцій нові користувачі можуть відчувати труднощі з адаптацією та потребують певного часу на освоєння платформи.
- Залежність від інтернет-з’єднання. Asana функціонує лише онлайн, тому робота з платформою неможлива без доступу до мережі.
- Суб’єктивна складність інтерфейсу. Хоча інтерфейс для багатьох є зручним, частина користувачів може сприймати його як не надто інтуїтивний, особливо при першому знайомстві з системою.

Третій і останній сервіс на Monday.com. Monday.com — це веб-платформа, що надає можливість налаштовувати власні інструменти для керування проєктами та робочими процесами в онлайн-середовищі. Продукт

був запущений у 2014 році, а в липні 2019 року компанія залучила 150 мільйонів доларів при оцінці у 1,9 мільярда доларів.

Сервіс дозволяє створювати проекти, розбивати їх на задачі та окремі під задачі. Сервіс дозволяє самому формувати вид таблиці та додавати окремі стовпчики, тобто користувач сам може вирішувати чи потрібно йому поле, наприклад пріоритету чи дати, також даний сервіс дозволяє прикріплювати до кожної задачі файли (зображення, документи, тощо). На рисунку 1.5. наведено приклад роботи в програмі Monday.com.

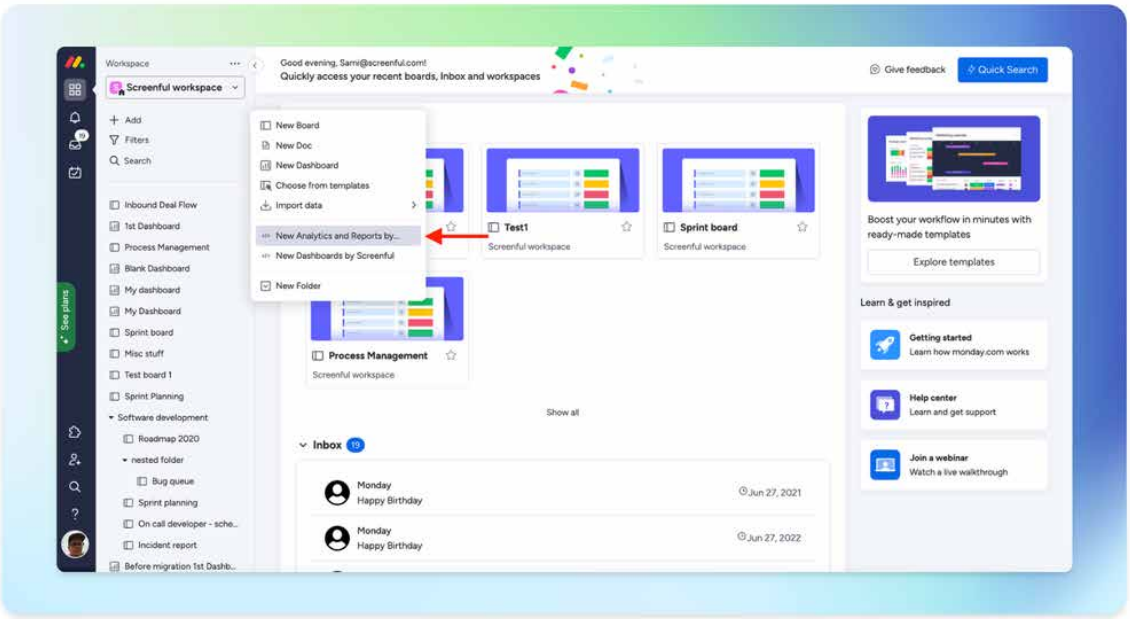

Рис. 1.5. Приклад роботи в програмі Monday.com

Однією з ключових переваг Monday.com є його зрозумілий та приємний для користувача інтерфейс, який практично не викликає труднощів навіть у тих, хто вперше стикається з системами управління проєктами. Цей сервіс створений таким чином, щоб взаємодія з ним була максимально інтуїтивною, що значно скорочує час на адаптацію. Крім того, Monday.com вирізняється високою гнучкістю — користувачі можуть адаптувати його під конкретні потреби команди або організації, налаштовуючи робочі процеси відповідно до специфіки проєктів. Це дозволяє вести контроль за виконанням завдань у зручному форматі, використовуючи графіки, діаграми, таблиці та інші інтерактивні елементи, що особливо цінується під час візуального аналізу

прогресу. Додатковим плюсом є широка інтеграція з іншими популярними сервісами, що значно спрощує спільну роботу та синхронізацію даних між різними платформами.

Разом із тим, варто враховувати і певні недоліки. Одним із найбільш очевидних є ціна — у порівнянні з багатьма альтернативами Monday.com може здатися дорожчим, особливо для невеликих команд або стартапів. Безкоштовний тариф також має суттєві обмеження, що зменшує функціональність і може стати стримувальним фактором для повноцінного використання системи. Окрім цього, хоч інтерфейс і виглядає простим, процес повноцінного впровадження платформи в робоче середовище нерідко потребує часу: користувачам необхідно ознайомитися з великою кількістю можливостей і налаштувань, що вимагає певного навчання та терпіння.

1.3. Аналіз вимог до програмної системи

Підсумовуючи всю отриману після аналізу інформацію, можна зробити висновки, що користувачі будуть очікувати такі функції:

- Створення та управління завданнями. Система повинна надавати можливість користувачам легко створювати нові завдання, присвоювати їм відповідальних, встановлювати терміни виконання та відстежувати стан завдань.
- Повідомлення та Сповіщення. Система повинна підтримувати механізм повідомлень для нагадування про терміни виконання завдань, а також для сповіщення про зміни та оновлення в системі.
- Спільна Робота та Комунікація. Система повинна забезпечувати зручні засоби спільної роботи, такі як коментарі, можливість обговорення завдань та обмін файлами між користувачами.
- Звітність та Аналітика. Система повинна надавати можливості створення звітів та аналізу продуктивності, щоб користувачі могли відстежувати свою роботу та приймати управлінські рішення.

- Історія та журналювання. Система повинна вести історію змін, зберігати журнали виконання завдань та надавати можливість перегляду попередніх станів та змін.
- Підтримка різних ролей та прав доступу. Система повинна підтримувати різні ролі користувачів (адміністратор, менеджер, виконавець) та надавати можливість налаштовувати права доступу для кожної ролі.
- Можливість імпорту та експорту даних. Система повинна забезпечувати можливість імпорту та експорту даних для зручного обміну інформацією з іншими системами та інструментами.

Відповідно, на основі аналізу будуть складені список вимог до проектованого програмного забезпечення системи обліку виконання виробничих завдань створення програмних систем.

1.3.1. Бізнес вимоги

До основних бізнес вимог такої системи відноситься автоматизація обліку та моніторингу завдань, що дозволить зменшити кількість ручних операцій і підвищити точність відслідковування прогресу роботи, покращення контролю та управління процесами, підвищення продуктивності продуктивності та ефективності завдяки автоматизованому обліку та аналізу виконання завдань, покращення якості планування процесів, а також скорочення витрат завдяки автоматизації процесів.

1.3.2. Функціональні вимоги

- Користувач повинен мати можливість реєструватися в системі.
- Користувач повинен мати можливість авторизуватися в системі на основі своїх даних.
- Користувач повинен мати можливість вийти зі свого облікового запису.
- Програма повинна мати можливість створення користувачів на основі ролей.

- Програма повинна мати можливість розмежування прав доступу до даних та операцій з даними на основі ролей.
- Користувач повинен мати можливість переглядати та редагувати інформацію свого профілю, таку як фото профілю, ім’я, прізвище, адреса.
- Користувач повинен мати можливість переглядати проекти в яких він є учасником.
- Користувач повинен мати можливість створювати нові проекти та додавати до них всю необхідну інформацію.
- Користувач з відповідними правами доступу повинен мати можливість редагувати дані проекту.
- Користувач з відповідними правами доступу повинен мати можливість архівувати проект.
- Користувач з відповідними правами доступу повинен мати можливість переглядати інформацію конкретного проекту.
- Користувач повинен мати можливість фільтрувати та шукати проекти й завдання за різними критеріями (статус, виконавець, дата тощо).
- Користувач з відповідними правами доступу повинен мати можливість створювати нові завдання проекту.
- Користувач з відповідними правами доступу повинен мати можливість редагувати дані завдань.
- Користувач з відповідними правами доступу повинен мати можливість призначати відповідального за завдання користувача.
- Користувач з відповідними правами доступу повинен мати можливість видаляти завдання.
- Користувач з відповідними правами доступу повинен мати можливість переглядати список завдань проекту.

- Користувач з відповідними правами доступу повинен мати можливість переглядати деталі конкретного завдання.
- Програма повинна мати можливість прикріплення файлів до конкретного проекту чи завдання.
- Програма повинна мати можливість видалення прикріплених файлів з конкретного проекту чи завдання.
- Користувач з відповідними правами доступу повинен мати можливість переглядати прикріплені файли.
- Користувач з відповідними правами доступу повинен мати можливість завантажувати прикріплені файли.
- Програма повинна мати можливість відправляти повідомлення користувачеві.
- Програма повинна автоматично сповіщати користувача про нові призначення завдань або зміни у проектах через системні повідомлення.
- Користувач з відповідними правами доступу повинен мати можливість змінювати права доступу інших користувачів.

1.3.3.Нефункціональні вимоги

- Система повинна забезпечувати структурне логування в окрему базу даних.
- Система повинна надавати функціонал для перегляду та управління логами.
- Система повинна забезпечувати логування результатів запитів, помилок.
- Система не повинна логувати чутливі дані користувачів, такі як паролі, чи токени доступу.
- Система повинна забезпечити безпечне з’єднання за допомогою протоколу HTTPS.

- Система повинна забезпечити зберігання усіх конфіденційних даних (паролі, токени, персональні дані) у зашифрованому вигляді з використанням сучасних алгоритмів (наприклад, AES-256 для даних, SHA-256 для хешування паролів).
- Система повинна забезпечити аутентифікацію через JWT-токени, з обмеженням часу дії access/refresh токенів.
- Система повинна забезпечити вихід з системи до анулювання поточного токена.
- Пароль користувача повинен відповідати мінімальним вимогам: не менше 8 символів, містити великі/малі літери, цифри та спеціальні символи.
- Система повинна бути захищена від найбільш критичних вразливостей, таких як: SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Broken Authentication, Security Misconfiguration.
- Система повинна виконувати резервне копіювання БД щонайменше раз на добу.
- Система повинна надавати можливість можливість відновлення системи з копії протягом 1 години після інциденту.
- Веб-застосунок повинен працювати на стаціонарних і мобільних пристроях (ПК, ноутбуки, планшети, смартфони).
- Інтерфейс повинен адаптуватися під екрани з роздільною здатністю від 360px (мобільні пристрої) до 1920px (десктопи).

1.4. Моделювання предметної області

Першим кроком під час моделювання предметної області було визначено основні абстракції системи, їх зображено на рисунку 1.6. Було виділено 5 основних абстракцій: каталог проектів, сховище файлів, система сповіщень, каталог завдань та каталог користувачів.

Важливою властивістю каталогу проектів є список всіх проектів. До обов’язків каталогу проектів входить зберігання, створення, реагування та видалення даних проектів. Схожим чином було визначено абстракцію каталогу завдань. ЇЇ важливою властивістю є список завдань, а до обов’язків входить зберігання, редагування, створення та видалення даних завдань у системі.

Важливе місце займає абстракція сховища даних, адже саме за допомогою сховища даних будуть відбуватися всі операції з файлами в системі. Властивості сховища – це файли та їх метадані. Метадані файлів це додаткова інформація, яка поступає разом з файлом, наприклад, ким і коли був доданий файл, або ким, коли і скільки разів редагований. Сховище файлів відповідає за зберігання файлів, вивантаження файлів та їх видалення.

Каталог користувачів відповідає за операції з користувачами системи, такі як зберігання, редагування та видалення даних.

Розглянемо абстракцію системи сповіщень. Система сповіщень займає важливу роль у загальній системі, адже вона надає повідомлення та сповіщення різним користувачам протягом всього часу роботи програми. Її властивості – це список налаштувань сповіщень та список користувачів підписаних на сповіщення. Список налаштувань сповіщень – це інформація про те, як користувачі хочуть отримувати повідомлення, через які канали зв’язку та на які дії. Список користувачів підписаних на сповіщення – це всі користувачі та сповіщення, які вони хочуть отримувати. До обов’язків системи сповіщень входить зберігання даних налаштувань сповіщень, зміна налаштувань сповіщень, видалення налаштувань, надсилання сповіщень згідно налаштувань та повторне надсилання сповіщень в разі невдачі.

Рис. 1.6. Абстракції системи обліку виконання виробничих завдань

Після визначення основних абстракцій системи, функціональних та нефункціональних вимог, наступним кроком є визначення ключових акторів та прецедентів системи. Основні прецеденти та актори наведені на діаграмі прецедентів на рисунку 1.7.

Адміністратор проекту може створювати, редагувати та видаляти проекти, це виражено у прецеденті «Управління проектами». Менеджер проекту може створювати, переглядати та редагувати завдання. Створення та редагування завдань розширяється за допомогою прецеденту «Прикріплення файлу», що означає, що при створенні та редагування завдання можна прикріпляти файл за бажанням. Учасники проекту можуть переглядати завдання та відповідні файли. Також усі учасники проекту мають власну сторінку профілю і можуть її редагувати, що визначається у прецеденті «Управління профілем».

Також на діаграмі прецедентів присутні окремі підсистеми. Система авторизації відповідає за авторизацію користувачів та надання прав доступу на основі ролей, сховище файлів відповідає за зберігання та обробку файлів, а також видачу файлів користувачам. Сервіс сповіщень надає функціонал обробки налаштувань політик сповіщень та доставки сповіщень.

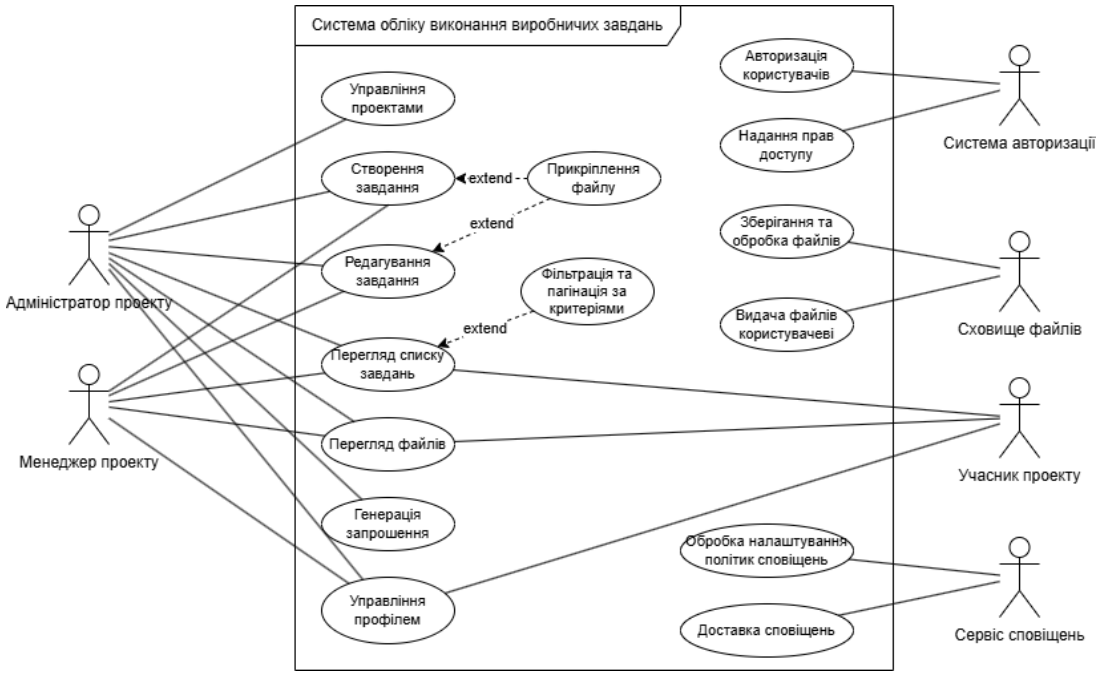

Рис. 1.6. Діаграма прецедентів системи обліку виконання виробничих завдань

Для розуміння деяких процесів, що відбувається в системі створено діаграмі послідовності, яку продемонстровано на рисунку 1.7. Першим кроком користувач надсилає запит на створення проекту разом з даними проекту, запит отримується компонентом «Web Api», далі цей запит надсилається до сервісу проектів, де відбувається необхідна бізнес логіка, пов’язана зі створенням проекту, після цього сервіс проектів надсилає запит до бази даних. В базі даних відбувається створення проекту і остання повертає результат роботи до сервісу завдань, який в свою чергу повертає цей результат до Web Api. Останній форматує результат і повертає в зручному для користувача вигляді. Таким чином завдання створено.

Далі, схожим чином, відбувається створення завдання до проекту. Спочатку користувач надсилає запит разом з даними для нового проекту, цей запит проходить свій шлях до бази даних через Web Api та сервіс роботи з завданнями, після чого відповідний результат повертається кінцевому користувачеві. Таким чином завдання створене.

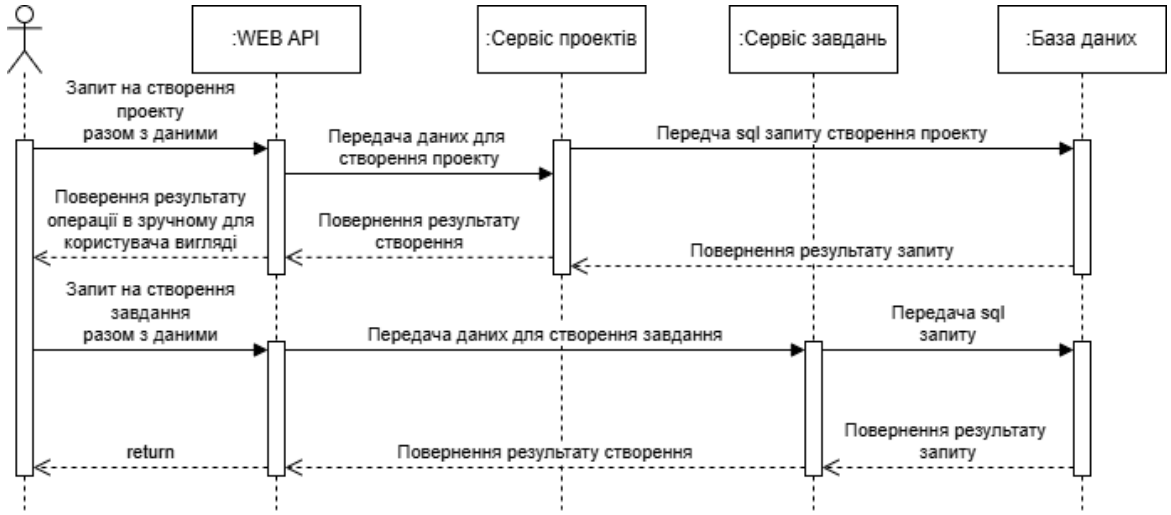

Рис. 1.7. Діаграма послідовності створення проекту та задачі

1.5. Висновки до розділу 1

Під час роботи над першим розділом відбулося дослідження та опис предметної області. Було досліджено проблематику обліку виконання виробничих завдань створення програмних систем, а також їх зв'язок з різними методологіями управління проектами.

Далі було розглянуто програми аналоги, а саме Trello, Jira та Monday.com. Наведено опис кожної програми, скріншоти роботи, а також переваги та недоліки.

Наступним кроком, після аналізу програм аналогів та потреб користувачів було визначено функціональні та нефункціональні вимоги до проектованої системи.

Останнім в цьому розділі розглянулося моделювання предметної області. Було виявлено основні абстракції системи, до яких увійшли: каталог проектів, сховище файлів, каталог завдань, каталог користувачів та система сповіщень. Для кожної абстракції було наведено важливі властивості та їх обов’язки. Також створено діаграму прецедентів, яка показує всіх основних акторів та прецедентів проектованої системи. Останнім кроком було проаналізовано послідовність створення проекту та завдання в системі і відображено на відповідній діаграмі послідовності.

РОЗДІЛ 2

2.1. Логічна модель даних у вигляді ER-діаграми

«Метою моделювання даних є забезпечення розробника ІС концептуальною схемою БД у формі однієї моделі або декількох локальних моделей, що відносно легко можуть бути відображені у будь-якій СКБД. Моделювання сутностей і зв’язків належить до моделювання логічних структур даних, яке базується на припущенні, що всі елементи ПО можна подати у вигляді ідеальних прототипів – сутностей. Такі концептуальні сутності описуються в термінах їхніх характеристик, або атрибутів (attribute). Сутності пов’язані через дії, які вони виконують відносно одна до одної. Ці дії встановлюють зв’язки (relationship) між сутностями» [1, с.150].

На даному етапі необхідно побудувати логічну модель даних, на основі якої буде відбуватися подальше проектування та розробка програмної системи. Логічна модель даних на початку дозволить побачити нариси та отримати початкове бачення системи. Для початкового моделювання буде використовуватися ER-діаграма. Entity Relations Diagram(ERD) – це найпоширеніший засіб семантичного моделювання даних. За допомогою ER-діаграм визначають головні об'єкти програмного забезпечення, їхні характеристики та зв’язки між ними.

На рисунку 2.1. зображено логічну модуль даних у вигляді ER-діаграми, що відображає концептуальну схему бази даних, не враховуючи особливості її реалізації. Є декілька основних таблиць: «Працівник», «Позиція», «Проект», «Проект», «Завдання», «Статус». Далі буде наведено детальний опис кожної з них.

Модель «Працівник» відображає працівників та користувачів системи, вона містить такі основні характеристики як ім’я, прізвище, по батькові та email. Вона зв’язана з таблицею «Позиція» неідентифікуючим зв’язком. «Позиція» – це окрема довідникова сутність, яка визначає роль працівника,

наприклад, розробник, тестувальник, аналітик, тощо. До основних її характеристик відноситься назва та короткий опис.

Сутність «Проект» відображає проекти та має такі основні характеристики: назва, дата створення, дата завершення, опис. Сутність «Проект» доречніше розглядати в зв’язці з сутністю «Завдання». Остання має такі характеристики: назва, короткий опис, дата створення, дата завершення. Модель «Завдання» відображає виробничі завдання, які можуть бути в проекті. «Проект» та «Завдання» зв’язані між собою ідентифікуючим зв’язком, адже завдання не може існувати без проекту, при чому сутність «Завдання» є дочірньою по відношенню до сутності «Проект».

Також на діаграмі присутня сутність «Статус». Це довідникова сутність, яка відображає статус виконання завдання, наприклад, в процесі, відмінено, на переробці, тощо. Її основні характеристики це назва та короткий опис. «Статус» та «Завдання» з’єднані між собою неідентифікуючим типом зв’язку.

Сутності «Працівник» та «Завдання» зв’язані між собою неідентифікуючим типом зв’язку, адже завдання не залежить від конкретного працівника, інколи воно може бути призначене не відразу.

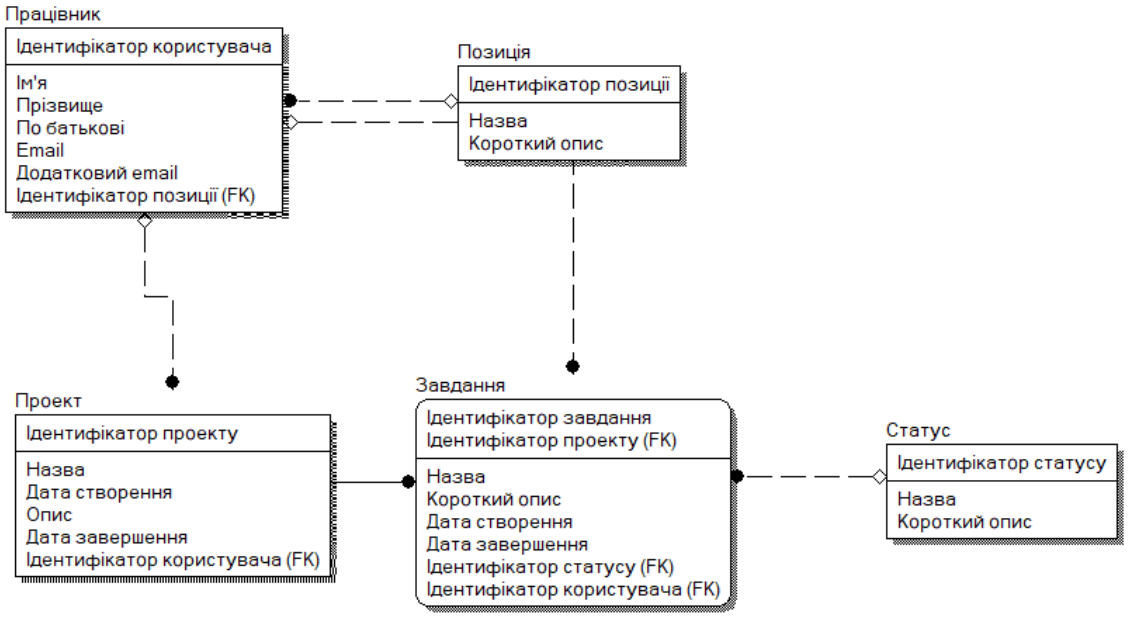

Рис. 2.1. Логічна модель даних системи обліку виконання виробничих завдань у вигляді ER-діаграми

Після створення логічної моделі даних системи обліку виконання виробничих завдань створення програмних систем у вигляді ER діаграми необхідно переконатися, що модель відповідає вимогам реляційності і, відповідно, третій нормальній формі.

«Відношення знаходиться в 3НФ тоді і тільки тоді, коли воно знаходиться у 2НФ і кожен неключовий атрибут не є транзитивно залежним від первинного ключа (це означає, що у відношенні відсутні будь-які взаємні залежності)»[2, с.136]. Іншими словами, що б модель знаходилася у третій нормальній формі, вона повинна вже знаходитися у другій нормальній формі, тобто всі атрибути функціонально залежать від усього первинного ключа, а також всі атрибути, які не є ключами, не повинні залежати від інших неключових атрибутів (відсутність транзитивних залежностей).

Наведено модель логічної схеми відповідає третій нормальній формі, адже кожна таблиця має свій первинний ключ, всі поля залежить лише від первинного ключу (тобто, не має часткових залежностей) і немає транзитивних залежностей (тобто, поля не залежать одне від одного, а лише від первинного ключа).

2.2. Діаграми класів та кооперацій

У процесі моделювання програмного забезпечення доцільно розпочинати побудову діаграм класів зі спрощеної версії, яка включає лише класи та асоціації між ними. Такий підхід є обґрунтованим як з практичної, так і з методологічної точки зору.

По-перше, це дозволяє зосередитися на розумінні логіки предметної області, визначити ключові сутності (класи), їх ролі та взаємозв’язки без ускладнення технічними деталями, такими як атрибути, методи чи модифікатори доступу. Простий рівень абстракції допомагає виявити фундаментальну структуру системи та уникнути помилок, що можуть виникати при передчасному зануренні в деталізацію.

Таким чином, відповідна діаграма класів з простими кооперaціями зображена на рисунку 2.2.

Клас «Employee» відображає всі основні атрибути та функції працівника системи, він пов’язаний асоціативним зв’язком з «Position». Останній, в свою чергу, відображає всі основні функції та атрибути позицій користувача, наприклад тестувальник чи аналітик.

Клас «Project» відображає всі основні атрибути та функції проектів, які перебувають в системі. Варто зауважити, що користувачі можуть створювати проекти, це показується в асоціативному зв’язку між «Employee» та «Project».

Клас, що відповідає за завдання проекту називається «Task». Кожен проект може мати завдання, одне, декілька чи нуль. Це визначається в асоціативному зв’язку між «Project» та «Task». Також варто зауважити, що кожне завдання може виконуватися людиною. Останнє твердження відображається в зв’язку між «Employee» і «Task».

Останньою залишається розглянути модель «Status». Це клас, який відображає всі атрибути та функції статусу завдання. Кожне завдання має певний статус, наприклад, «в роботі», «заплановано», тощо. Саме ця залежність відображається в асоціативному зв’язку між «Task» та «Status».

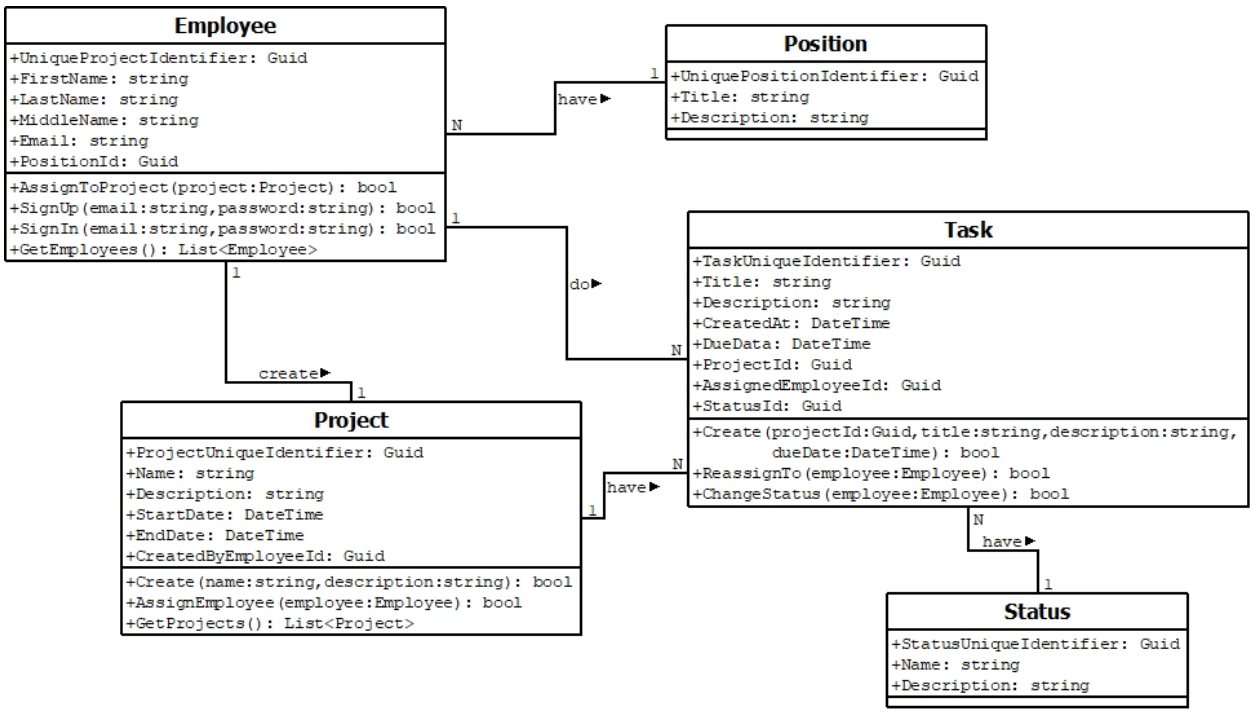

Рис. 2.2. Діаграма класів системи обліку виконання виробничих завдань з використанням простих асоціацій

На початкових етапах моделювання програмної системи, доцільним є створення діаграми класів з простими асоціаціями між основними сутностями. Такий підхід дозволяє зосередитися на базових зв'язках між об'єктами предметної області, сформувати цілісне розуміння структури системи та виключити зайву складність на ранньому етапі.

Після цього, коли структура основних сутностей уже стабілізована, доречно переходити до побудови повноцінної діаграми класів.

На рисунку 2.3. зображено діаграму класів. На ній можна помітити, що кожен клас-сервіс реалізовує певний інтерфейс. Зроблено це для того, що б досягти масштабованості та підтримуваності коду. Код буде залежати від інтерфейсу, а не реалізації, що в разі необхідності дозволить змінити реалізацію на іншу, без зміни значної частини кодової бази. Достатньо буде лише створити нову реалізацію та підмінити її в одному місці. Іншими словами, інтерфейси дозволяють уникати жорстких залежностей між компонентами системи.

Також варто зауважити, що кожен клас-сервіс має свою зону відповідальності. Це забезпечує принцип єдиної відповідальності, що також, як і попередній пункт, сприяє створенню масштабованих та підтримуваних програмних систем.

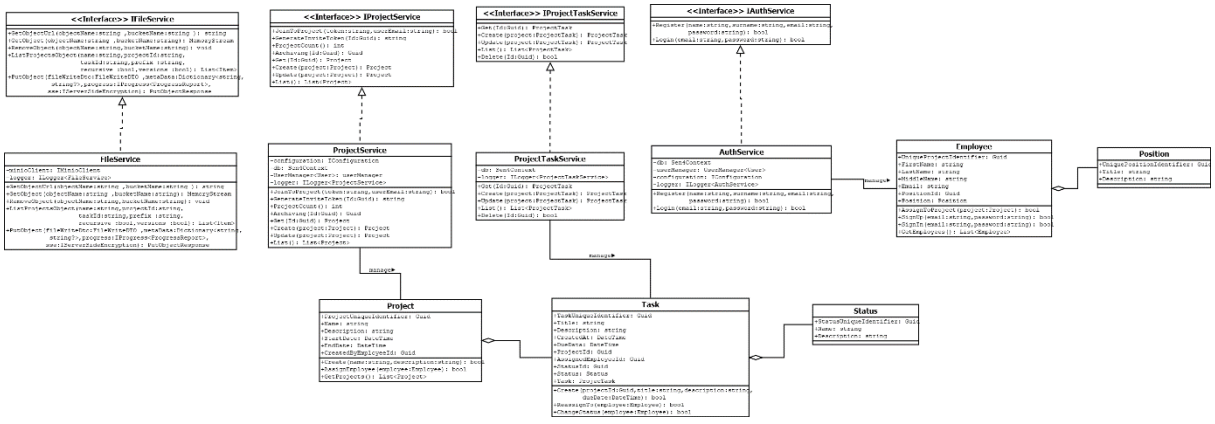

Рис. 2.3. Діаграма класів до проектованої системи

2.3. Діаграма пакетів

Діаграма пакетів є засобом моделювання, що дозволяє групувати елементи системи (наприклад, класи, модулі, компоненти) у логічні

об'єднання — пакети. Вона відображає структуру системи на високому рівні, забезпечуючи уявлення про організацію і взаємозв’язки між окремими частинами програмного забезпечення. На рисунку 2.4. продемонстровано діаграму пакетів системи обліку виконання виробничих завдань створення програмних систем.

На найнижчому рівні persistence зберігаються пакети, які так чи інакше належать до зони відповідальності роботи з базою даних. Db context – це пакет, який зберігає необхідні класи для взаємодії з базою даних, exceptions – зберігає класи та типи помилок, які можуть виникати під час роботи з таким класом, а пакет configurations зберігає всі необхідні файли конфігурації, такі як рядок підключення, політики та конфігурації моделей. На цей рівень посилається core, про нього мова піде далі.

Рівень core зберігаються моделі та контракти (інтерфейси) роботи з ними.

На рівні infrastructure зберігаються пакети сервісів, помилок, методів розширень (exceptions) та dto. Сам рівень може посилатися на різні компоненти, які знаходяться за межами системи, на діаграмі це зображено компонентом external system.

На рівні api знаходяться пакети контроллерів, фільтрів, класів відображення, методів розширень, а також різних конфігурацій, необхідних для роботи системи на цьому рівні.

На рівні presentation layer знаходяться компоненти відображення та логіка роботи з ними.

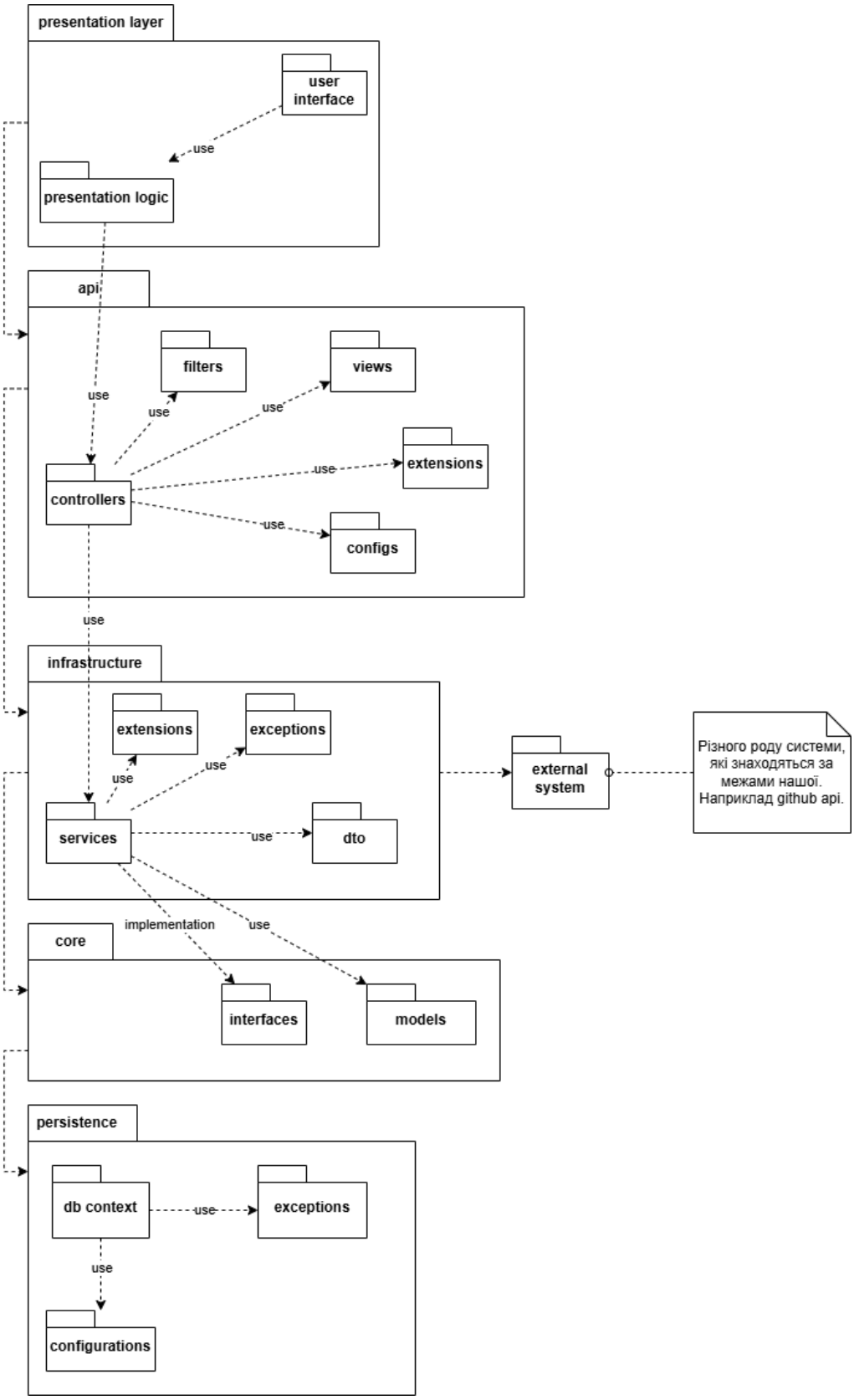

Рис. 2.4. Діаграма пакетів обліку виконання виробничих завдань створення програмних систем

2.4. Діаграми станів

«Діаграма станів (State chart diagram) визначає усі можливі стани (state), у яких може знаходитися конкретний об’єкт під час свого існування, а також процес зміни станів цього об’єкта у результаті настання деяких подій (event)»[3 с.97].

Відповідно, діаграма станів є важливим пунктом у проектуванні програмної системи, адже дозволяє зрозуміти всі можливі стани різних об’єктів на етапі аналізу, що в майбутньому сприяє точнішій реалізації логіки поведінки системи, підвищує її надійність та полегшує виявлення і виправлення помилок на ранніх стадіях розробки.

На рисунку 2.5. наведено діаграму станів до сутності «Завдання». Першим кроком завдання перебуває в стані «Створено», під час цього стану йому доступні такі операції, як зміна статусу, редагування інформації та видалення. З цього стану завдання може перейти в стан «Заплановано», під час цього стану йому доступні всі вищеописані операції. Зі стану «Заплановано» завдання може перейти в два різні стани, а саме «Виконано» та «Відмінено», в обох станах сутності доступна лише операція видалення.

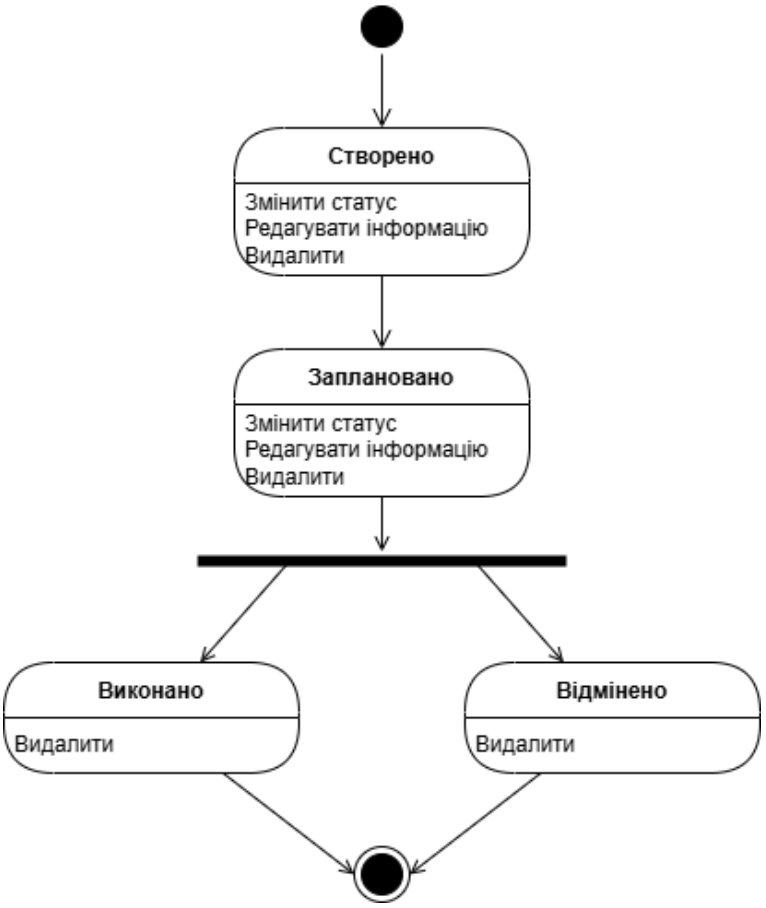

Рис. 2.5. Діаграма станів для сутності «Завдання»

На рисунку 2.6. зображено діаграму станів для сутності «Проект». Проект, після створення, перебуває в стані «Створено». В цьому стані він має доступний функціонал для видалення, редагування та запрошення нового користувача до проекту. Зі стану «Створено» проект може перейти в стан «В роботі», під час такого стану він має доступ до всіх вищеописаних операцій. Останнім станом проекту є стан «Видалено», під час нього доступна лише одна функція, а саме «Відновити».

Рис. 2.6. Діаграма станів для сутності «Проект»

2.5. Діаграма компонентів

На рисунку 2.7. зображено діаграму компонентів до проектованої системи обліку виконання виробничих завдань створення програмних систем. На ній продемонстровано фізичне представлення моделі у загальному вигляді. Першим кроком користувач взаємодіє з різними HTML сторінками, на яких зображено таблиці, кнопки, форми, тощо. HTML сторінки взаємодіють з модулем API, який об’єднує в собі логіку для відображення кожної сутності. Наприклад, UserController відповідає за користувачів, ProjectController – за проекти і, відповідно, TaskController за сутність завдань. Сутності користувача, проектів та контроллерів мають схожий опис, тому далі пояснення будуть наводитися на прикладі сутності користувачів.

UserController, компонент, залежить від інтерфейсу IUserService, тобто від абстракції роботи з користувачами. В модулі користувача наведено дві реалізації. Що стосується першої реалізації, а саме UserService, - то це вона

реалізує бізнес логіку роботи з сутністю користувача. Під бізнес логікою мається на увазі різні операції, такі як, створення, видалення, сортування, тощо. Сам модуль напряму залежить від DataBaseContext, який є класом контекстом бази даних. За допомогою цього класу ми напряму взаємодіємо з базою даних, відповідно він залежить від сховища.

Що стосується іншої реалізації, це UserServiceAdapter. Він також реалізує інтерфейс IUserService, а також залежить від UserService. Тут використовується структурний патерн адаптер, який обвертає функціональність нашого сервісу і додає функціонал роботи з кешуванням. UserServiceAdapter залежить від інтерфейсу ICacheService, який представляє контракт роботи з кешем. Реалізацією цього інтерфейсу є клас CacheService, в якому вже зосереджена логіка роботи з кешуванням. Він напряму залежить від сховища для кешу.

Далі окремо необхідно розглянути Notification module. Це модуль для роботи з сповіщеннями. В ньому є компонент NotificationService, який займається відправкою сповіщень. Він не знає куди він відправляє, та яким чином, він вміє лише відправляти. Він залежить від інтерфейсу ISender, який представляє собою контракт відправлення. Класи, що його реалізують знають куди і як відправляти. Таким чином, є можливість створювати безліч класів-відправників і розширювати функціонал сповіщень без модифікації класу NotificationService.

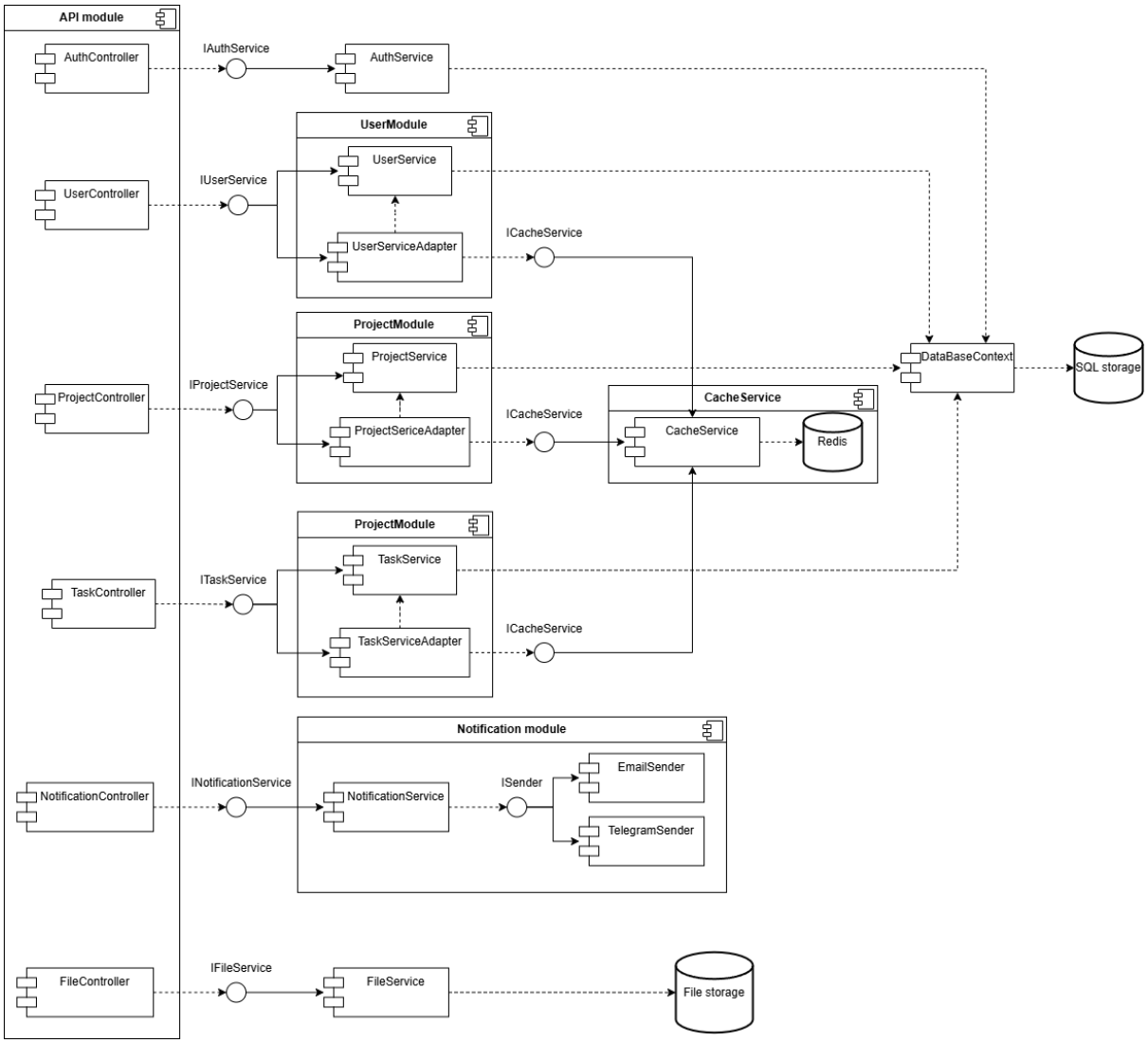

Рис. 2.7. Діаграма компонентів до проектованої системи обліку виконання виробничих завдань створення програмних систем

2.6. Висновки до розділу 2

Другий розділ був присвячений етапу проектування програмної системи, яка складається з двох основних частин – інформаційного та прикладного програмного забезпечення.

Першим кроком було створено логічну модель даних у вигляді ER-діаграми та наведено її детальний опис. Було зроблено висновок, що побудована модель відповідає вимогам реляційності і, відповідно, третій нормальній формі.

Наступним кроком відбувалося моделювання програмного забезпечення за допомогою різних діаграм, таких як діаграми класів та кооперацій, пакетів,

станів та компонентів. Побудова діаграми класів складалася з двох частин, а саме побудови діаграми з використанням простих асоціацій, а потім уже повноцінної діаграми. Діаграма пакетів, у свою чергу, групувала елементи системи у логічні об'єднання і відображала структуру системи на високому рівні, в той час, як діаграма компонентів демонструвала фізичне представлення моделі у загальному вигляді.

РОЗДІЛ 3

3.1. Система управління інформаційною базою

Для реалізації моделі даних першим кроком важливо вибрати систему управління базами даних. Бази даних, дуже часто, займають головне місце в сучасних програмних системах. Володіння знаннями та навичками створення та управління базами даних є важливою навичкою, яка необхідна для створення подібного роду систем.

Станом на сьогодні у світі існує значна кількість різноманітних систем управління базами даних, призначених для роботи з реляційними структурами. Серед них зустрічаються як платні програмні продукти, так і безкоштовні рішення з відкритим доступом.

Програмні комплекси для керування базами даних комерційного типу, зокрема Oracle чи DB2, створюються та підтримуються відомими технологічними корпораціями. Вони спеціалізуються на обробці значних масивів інформації та забезпечують одночасну роботу багатьох користувачів. Через це такі СУБД відзначаються високою ціною й зазвичай використовуються великими організаціями та підприємствами.

В якості СУБД для даного проекту було обрано PostgreSQL — потужну, вільно розповсюджувану реляційну СУБД з відкритим вихідним кодом. PostgreSQL — об’єктно-реляційна база даних, а це означає, що в ній підтримується об’єктно-орієнтовано модель, тобто класи, наслідування, тощо.

PostgreSQL підтримує резервне копіювання даних та відновлення, стандарт SQL, представлення, тригери, всі стандартні запити: SELECT/INSERT/UPDATE/DELETE/UNION/WITH, а також підзапити й різноманітні специфічні операції (наприклад, REPLACE у MySQL і MERGE у PostgreSQL. Також вона підтримує різноманітні типи даних, такі як JSON, UUID, та, навіть, користувацькі типи даних. Таким чином, вона надає весь необхідний функціонал для створення подібного роду систем. PostgreSQL

підходить для складних і високонавантажених систем, де важлива підтримка транзакцій, складні запити та бізнес-логіка.

PostgreSQL було обрана за її переваги перед іншими СУБД. Одна з основних її переваг це об’єктно-реляційна модель, що на відміну від класичних реляційних СУБД, PostgreSQL є об’єктно-реляційною, що дозволяє створювати власні типи даних, використовувати успадкування таблиць, створювати власні оператори та функції користувача.

Наступною її перевагою є повноцінна підтримка JSON та JSONB форматів, що дозволяє зберігати специфічні дані без втрати продуктивності та функціональності.

Також до її переваг відноситься розширюваність та масштабованість. PostgreSQL — надзвичайно гнучка СУБД, вона підтримує розширення типів даних для географічних даних, для моніторингу, для IP адрес, тощо.

Дану СУБД було вибрано в тому числі за її активний розвиток, адже PostgreSQL — open-source проєкт з активною міжнародною спільнотою. Цей проект розвивається досить активно, випускаються регулярні оновлення та покращення.

Ця СУБД Працює на всіх популярних ОС: Windows, Linux, macOS, BSD, підтримується Docker, Kubernetes, хмарні сервіси (AWS RDS, GCP Cloud SQL, Azure DB).

Також дуже важливим параметром при виборі СУБД для проекту є її продуктивність. Посилаючись на дослідження продуктивності PostgreSQL, можна сказати, що вона має перевагу в деяких запитах, що може бути важливим в контексті проектованої програмної системи:

«Отримані кількісні результати демонструють перевагу PostgreSQL у виконанні операцій вибірки. У первинних тестах час виконання запиту у PostgreSQL для 1 мільйона записів коливався від 0,6 мс до 0,8 мс, тоді як у MySQL — від 9 мс до 12 мс, що свідчить про приблизно 13-кратну перевагу PostgreSQL. Для вибіркових запитів з умовою (WHERE) PostgreSQL показав

час виконання 0,09–0,13 мс, тоді як MySQL — 0,9–1 мс, що робить PostgreSQL приблизно у 9 разів ефективнішим.

Операції вставки мали схожі результати: у PostgreSQL — від 0,0007 мс до 0,0014 мс, у MySQL — від 0,0010 мс до 0,0030 мс. У складних експериментах із одночасним виконанням операцій PostgreSQL зберігав стабільну продуктивність (0,7–0,9 мс для запитів вибірки під час вставки), тоді як продуктивність MySQL значно погіршувалась (7–13 мс).

Ці результати підтверджують доцільність використання PostgreSQL в середовищах, що потребують низької затримки при роботі з даними та надійної обробки паралельних запитів, що робить цю СУБД ідеальною для систем безперервної автентифікації.»[5]

3.2. Вибір інструментарію для створення прикладного програмного забезпечення

Під час вибору інструментарію для створення прикладного програмного забезпечення важливо обрати сучасний та ефективний технологічний стек, що забезпечує масштабованість, підтримку високого навантаження та зручність супроводу. Всі обрані технології мають відповідати всім сучасним стандартам безпеки та продуктивності, а також їх має бути достатньо для задоволення всіх функціональних вимог до системи.

В якості основної мови програмування було обрано C# 8 версії. Це потужна об’єктно орієнтована мова, яка має досить широку екосистему в середовищі .Net. Вона ідеально підходить для серверної логіки та взаємодії з базами даних, адже надає спеціальні високопродуктивні фреймворки для таких цілей. Її розробка та підтримка ведеться корпорацією Microsoft, що гарантує її актуальність, регулярні оновлення та наявність якісної документації.

Основною технологією для розробки логіки серверної частини виступає фреймворк Asp.Net. Це фреймворк екосистеми .Net, який також розробляє та підтримує команда Microsoft, що гарантує всі вищеописані плюси. Asp.net надає весь необхідний функціонал для створення зручних, продуктивних веб

додатків. Важливою перевагою фреймворку є багатошарова архітектура, що включає ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web API і ASP.NET Core. Ця гнучка структура надає розробникам різноманітні можливості, а вибір конкретного підходу залежить від вимог проекту. В межах даного проекту буде використовуватися тип Web API.

Для роботи з базою даних буде використовуватися інструмент екосистеми .Net - Entity Framework Core. Entity Framework Core – ORM (Object-Relational Mapping) інструмент, що спрощує взаємодію з реляційною базою даних. Дозволяє розробнику працювати з даними за допомогою об'єктів C#, автоматично генеруючи SQL-запити та підтримуючи міграції. ORM – це об’єктно реляційне відображення. EF відображає об'єкти C# на таблиці бази даних, дозволяючи працювати з об'єктами замість SQL-запитів.

До переваг Entity Framework Core відноситься LINQ. Це потужна мова запитів, дозволяє здійснювати запити до даних за допомогою більш інтуїтивного та виразного синтаксису.

В якості підходу створення бази даних буде використовувати Code First. Code First – це підхід, при якому створюється модель домену за допомогою класів C#, і EF6 автоматично створить схему бази даних на її основі.

Важливою частиною розробки проекту є валідація вхідних даних та захист від помилок. Для таких цілей буде використовуватися FluentValidation. FluentValidation – бібліотека для декларативної валідації моделей. Забезпечує централізовану, гнучку та розширювану перевірку вхідних даних, що підвищує надійність додатку та полегшує тестування.

Відповідно до вимог проекту є потреба зберігати велику кількість різних файлів, це означає необхідність в файловому сховищі. В якості файлового сховище буде виступати MinIO. Це високопродуктивне об’єктне сховище, сумісне з Amazon S3 API. Використовується для зберігання файлів користувачів, таких як документи чи зображення, з можливістю масштабування та швидкого доступу. До його переваг відноситься те, що його дуже легко налаштувати та ввести в експлуатацію, а також сумісність с

Amazon S3 API, що дозволить в майбутньому, при потребі, замінити його на більш потужне сховище.

Що стосується логування, то для цього будуть використовуватися Serilog та Seq. Це інструменти для структурованого логування. Serialog – бібліотека екосистеми .Net, яка надає весь необхідний функціонал для логування. В свою чергу Seq – це централізоване сховище журналів для логів з великою кількістю можливостей, таких як пошук та фільтрування, аналіз подій на основі діаграм, повідомлення в реальному часі та ще багато інших. Ці два інструмента повністю задовільнять потребу у структурному логування подій у системі обліку виконання виробничих завдань.

Останнім кроком необхідно визначитися з інструментами для створення інтерфейсу користувача. HTML, CSS, JavaScript, React – використано для створення клієнтської частини. HTML, CSS та JavaScript – це стандартні інструменти для створення будь якого веб-додатку. В свою чергу, React – це бібліотека JavaScript з відкритим кодом для розробки користувацьких інтерфейсів. React дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю.

Таким чином було визначено стек технологій, які будуть використовуватися при створенні програмної системи обліку виконання виробничих завдань створення програмних систем. Зазначений стек технологій дозволяє розробити продуктивну, масштабовану та підтримувану систему, яка відповідає сучасним вимогам до розробки програмного забезпечення.

3.3. Розробка інформаційної бази

Як було зазначено раніше, для роботи з базою даних використовується Entity Framework Core та підхід «Code First», що означає, що спочатку мають бути створені прості класи, які будуть відображати сутності в базі даних.

Отже, першим кроком будуть створені прості C# класи, які і будуть відображати таблиці в базі даних. При проектування таких класів необхідно враховувати деякі особливості роботи з Entity Framework Core. Спочатку

необхідно об’явити поле первинного ключа, це є однією з вимог фреймворку. Назва поля має бути «ID», або довільна, за умови, що поле позначено атрибутом «PrimaryKey». Також варто звернути увагу на поля навігації. Це властивості в класах-сутностях, які дозволяють переходити від однієї сутності до пов’язаної. Вони потрібні для встановлення та зручної роботи з відношеннями типу один-до-одного, один-до-багатьох або багато-до-багатьох.

На рисунку 3.1. наведено розроблений клас для моделі «Проект». Першим кроком об’явлено поле «Id» типу «GUID», яке представляє собою первинний ключ. Далі створено звичайні поля, які необхідні для цієї моделі: ім’я, дата створення, дата видалення, опис. Варто звернути увагу на поля 5-7. Ці поля відображають логічний зв’язок таблицею проектів та іншими.

```
public class Project
{
    13 usages  More...
    public Guid Id { get; init; }
    5 usages
    public string Name { get; init; } = null!;
    4 usages
    public DateTime CreatedDateTime { get; init; }
    1 usage
    public string? Description { get; init; }
    3 usages
    public IList<User> Users { get; init; } = new List<User>();
    public IList<Notification> Notifications { get; init; } = new List<Notification>();

    public IList<ProjectTask>? ProjectTasks { get; init; }
    4 usages
    public DateTime? DeletedBy { get; set; } = null;
}
```

Рис. 3.1. Розроблений клас для моделі «Проект»

Схожим чином створюються і інші класи для різних моделей. Наприклад, на рисунку 3.2. наведено розроблений клас для моделі «Статус».

```
public class TaskStatus
{
    6 usages
    public Guid Id { get; set; }
    9 usages
    public string Name { get; set; } = null!;
    public IList<ProjectTask> ProjectTasks = new List<ProjectTask>();
}
```

Рис. 3.2. Розроблений клас для моделі «Статус»

Після створення моделей необхідно визначити їх конфігурацію. Під конфігурацією в даному випадку розуміються різні властивості полів моделей, такі як, наприклад, максимальне чи мінімальне обмеження, максимальна чи мінімальна довжина, тощо. Для цього необхідно створити клас, який наслідує інтерфейс IEntityTypeConfiguration<T>, де T – це наша модель, яку ми хочемо конфігурувати. На рисунку 3.3. наведено приклад класу конфігурації для моделі «Завдання». Створено клас з який наслідує IEntityTypeConfiguration<Project>. Клас реалізовує єдиний метод цього інтерфейсу – Configure, в якому і відбувається конфігурація. Дана бібліотека надає велику кількість різноманітних конфігурацій для моделей, в даному випадку використовується обмеження довжини імені на 100 символів, та опису на 1000 символів, а також налаштовування вторинних ключів сутності.

```
public class ProjectTaskConfiguration: IEntityTypeConfiguration<ProjectTask>
{
    Tenshi
    public void Configure(EntityTypeBuilder<ProjectTask> builder)
    {
        builder.Property(p :ProjectTask => p.Name).HasMaxLength(100);
        builder.Property(p :ProjectTask => p.Description).HasMaxLength(1000);

        builder.HasOne(p :ProjectTask => p.UserCreated).WithMany(p :User => p.CreatedTasks);
        builder.HasOne(p :ProjectTask => p.UserExecutor).WithMany(p :User => p.ResponsibilitiesTasks);
    }
}
```

Рис. 3.3. Клас конфігурації для моделі «Завдання»

Аналогічно відбувається конфігурація всіх інших необхідних моделей. Після створення всіх моделей та їх конфігурацій, наступним кроком є створення класу контексту бази даних. Клас контексту бази даних, це клас,

який наслідує DbContext і є посередником між C# кодом та реальною базою даних. Він дозволяє працювати з таблицями як з колекціями об'єктів і виконує перетворення між ними. Цей клас повинен обов’язково мати: два конструктори, поля з типом DbSet<T>, де T – це створені моделі, які будуть виступати таблицями в базі даних, а також метод OnModelCreating, який застосовує створені конфігурації до моделей. На рисунку 3.4. зображено частину класу контексту бази даних, на якому видно два конструктори та всі поля, моделі яких будуть використовуватися в базі даних. Саме через ці поля і відбувається взаємодія з базою даних, всі операції, такі як читання, редагування, видалення, тощо.

```
public class Sen4Context: IdentityDbContext<User, IdentityRole<Guid>, Guid>
{
    new *
    public Sen4Context(DbContextOptions<Sen4Context> options) : base(options) => Database.EnsureCreated();
    new *
    public Sen4Context() => Database.EnsureCreated();

    2 usages
    public DbSet<Post> Posts { get; set; }
    2 usages
    public DbSet<Priority> Priorities { get; set; }
    18 usages
    public DbSet<Project> Projects { get; set; }
    8 usages
    public DbSet<ProjectTask> ProjectTasks { get; set; }
    public DbSet<TaskFile> TaskFiles { get; set; }
    2 usages
    public DbSet<TaskStatus> TaskStatuses { get; set; }
    17 usages
    public DbSet<User> Users { get; set; }
    4 usages
    public DbSet<Rule> Rules { get; set; }
    12 usages
    public DbSet<Operation> Operations { get; set; }
    6 usages
    public DbSet<UsersProjects> UsersProjects { get; set; }
```

Рис. 3.4. Код класу контексту бази даних з конструкторами та полями

Що стосується методу OnModelCreating, то його код наведено на рисунку 3.5. В ньому просто створюються екземпляри класів конфігурацій.

```
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    builder.ApplyConfiguration(new ChannelConfiguration());
    builder.ApplyConfiguration(new NotificationConfiguration());
    builder.ApplyConfiguration(new NotificationRuleConfiguration());
    builder.ApplyConfiguration(new PostConfiguration());
    builder.ApplyConfiguration(new PriorityConfiguration());
    builder.ApplyConfiguration(new ProjectConfiguration());
    builder.ApplyConfiguration(new ProjectTaskConfiguration());
    builder.ApplyConfiguration(new TaskStatusConfiguration());
    builder.ApplyConfiguration(new UserConfiguration());
    builder.ApplyConfiguration(new OperationConfiguration());
}
```

Рис. 3.5. Код методу OnModelCreating з конфігураціями моделей

Таким чином базу даних для проекту створено. Після першого запуску програми, вона автоматично створить базу даних на основі наших класів. На рисунку 3.6. наведено діаграму створеної БД, яку було витягнуто з IDE JetBrains DataGrip. Варто зауважити, що вона відрізняється від побудованої раніше логічної схеми, адже фреймворки створюють деяку кількість службових та допоміжних таблиць, такі як, наприклад, таблиці авторизації, токенів, ролей, тощо.

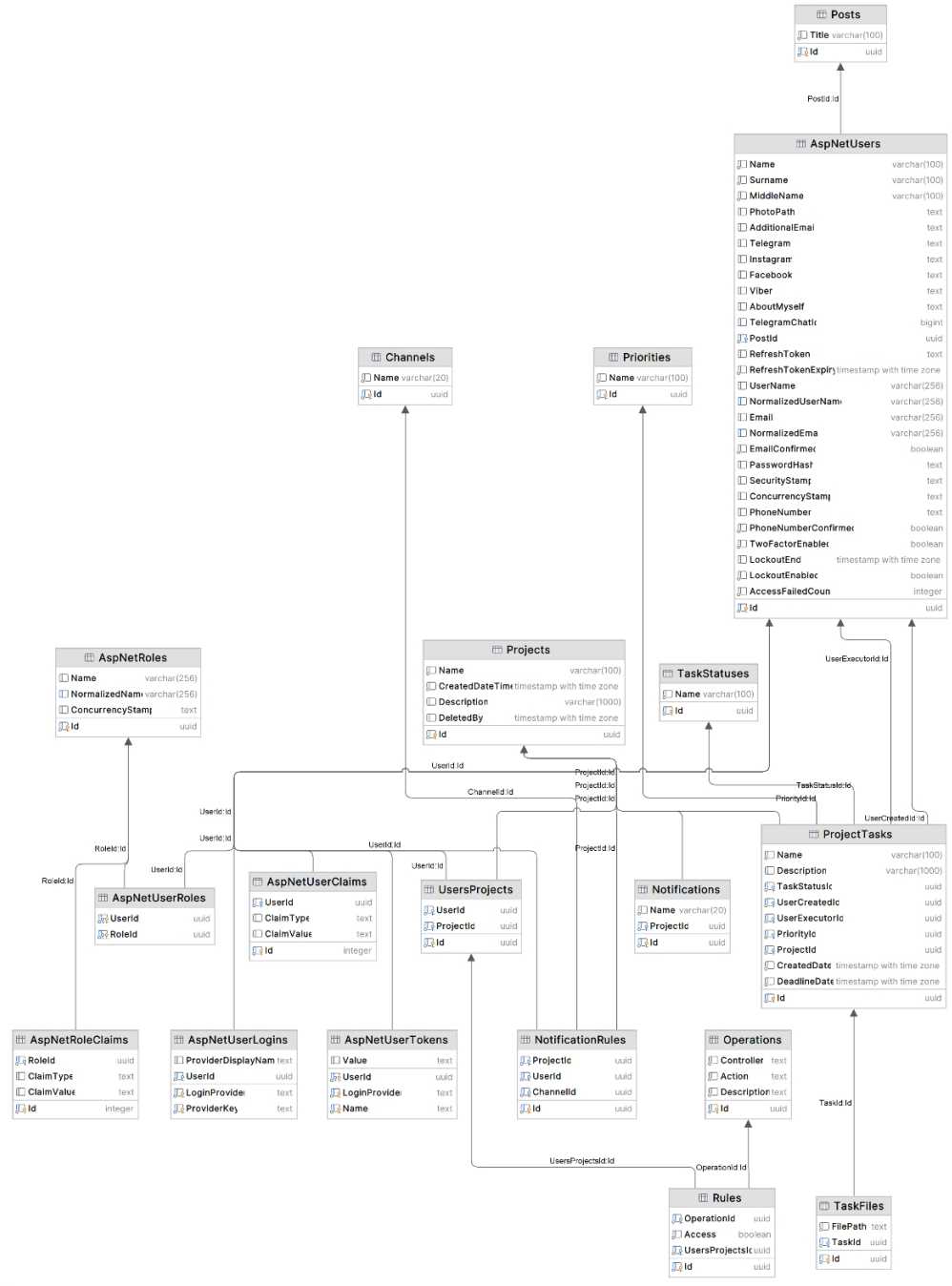

Рис. 3.6. Діаграма бази даних після її створення

3.4. Алгоритмізація та програмування програмних модулів

Програмне забезпечення системи обліку виконання виробничих завдань складається з деякої кількості підсистем, або модулів, в яких зосереджена певна логіка роботи програми.

Першим кроком доречно розглянути авторизацію та аутентифікацію в системі. Аутентифікація та авторизація є ключовими складовими безпеки інформаційної системи. У розробленій системі вони відіграють критичну роль,

оскільки забезпечують контроль доступу до різних функцій залежно від ролі користувача.

Аутентифікація — це процес перевірки особистості користувача на основі його облікових даних (електронна пошта та пароль). Система реалізує механізм аутентифікації з використанням технології JWT (JSON Web Token), що дозволяє створювати безпечні сесії без збереження стану на сервері. Користувач вводить свої облікові дані, і у випадку їхньої коректності система генерує токен доступу, який надалі використовується для взаємодії з API.

Авторизація — це процес перевірки прав користувача на виконання певних дій після проходження аутентифікації.

Ролі реалізовано через вбудований механізм ASP.NET Core Identity, що дозволяє керувати користувачами, ролями, дозволами, а також шифрувати та зберігати паролі відповідно до сучасних стандартів безпеки.

Доречним буде детальніше розглянути, що таке JWT. JWT – це компактний та безпечний спосіб представлення даних між двома сторонами. Дані передаються у JSON форматі і підписуються за допомогою JSON Web Signature (JWS). JSON Web Signature – це стандарт для підпису різних даних. JWT токени складаються з 3 частин: заголовок, корисне навантаження та підпис. Заголовок представляє собою рядок, що закодовує звичайний JavaScript об'єкт, який описує токен, а також використаний алгоритм хешування. Корисне навантаження – це дані, які передаються разом з токеном. Третя частина — підпис, що використовується для перевірки JWT.

Блок-схема алгоритму авторизації наведена на рисунку 3.7. Першим кроком користувач системи вводить логін та пароль. Система перевіряє, чи існує такий користувач, і якщо так, чи вірний пароль. Якщо ж такого користувача не існує, або пароль невірний, система генерує повідомлення про помилку та надсилає користувачеві. При такому варіанті авторизація закінчується і її потрібно починати знову. В разі вірно введених даних, система генерує JWT токен доступу та токен оновлення, останній використовується для оновлення JWT в разі, закінчення його терміну дії. Далі токени

зберігаються в базі даних і повертаються користувачеві, в свою чергу користувач використовує їх при кожному запиті до системи, як доказ того, що він авторизований.

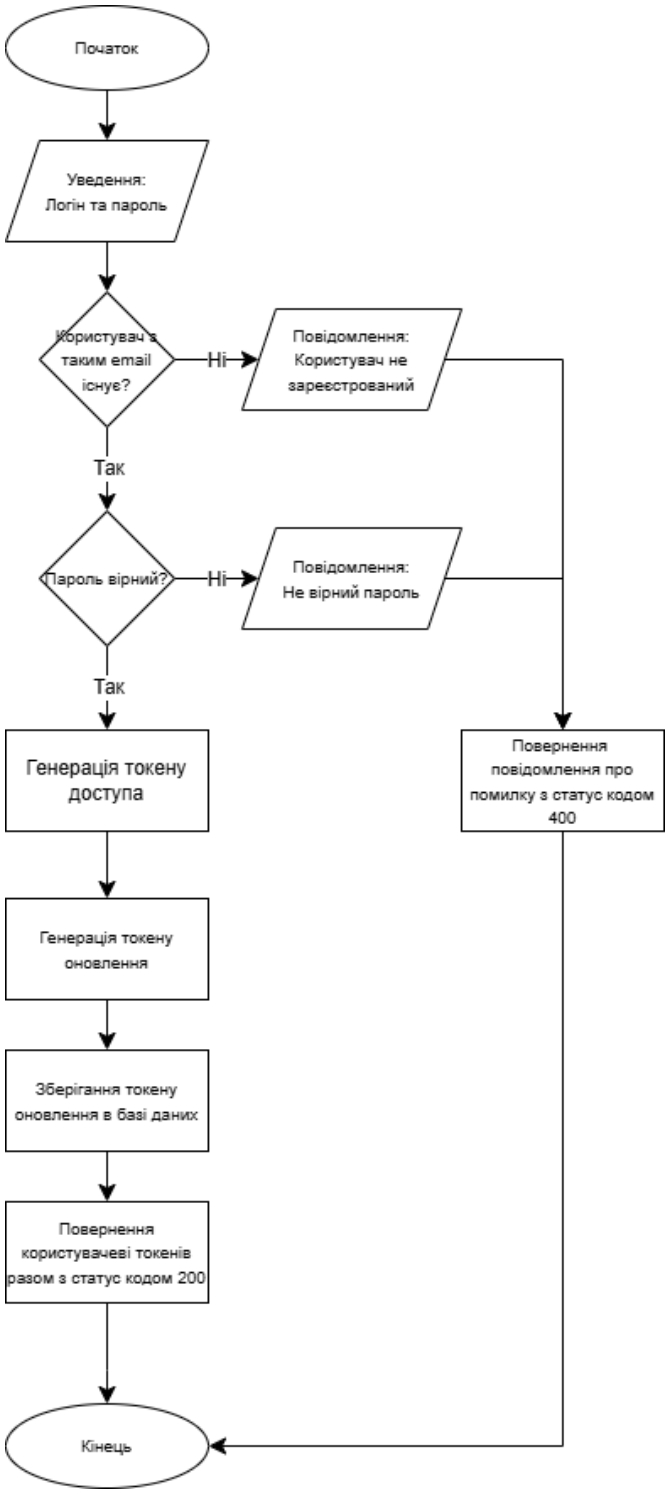

Рис. 3.7. Блок-схема алгоритму авторизації користувача в системі

Таким чином працює авторизація користувача. Він отримує токен доступу, який містить всю необхідну інформацію, на основі якої відбувається

надання прав доступу до системи. На рисунку 3.8. представлено фрагмент коду, який відповідає за створення токенів.

```
//create access&refresh tokens
var claims = new List<Claim>()
{
    new Claim( type: ClaimTypes.NameIdentifier, user.Id.ToString()),
    new Claim( type: ClaimTypes.Name, user.UserName!),
    new Claim( type: JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
};
var jwtSecret :string  = AppConfiguration.GetRequiredConfigurationValue(configuration, key: "JWT:Secret");
var issuer :string  = AppConfiguration.GetRequiredConfigurationValue(configuration,  key: "JWT:ValidIssuer");
var audience :string  = AppConfiguration.GetRequiredConfigurationValue(configuration,  key: "JWT:ValidAudience");
var expires :DateTime  = DateTime.UtcNow.AddHours(int.Parse(AppConfiguration.GetRequiredConfigurationValue(configuration,  key: "JWT:AccessValidTime")));

var accessToken :string  = tokenService.CreateJwtToken(claims, jwtSecret, issuer, audience, expires);
var refreshToken :string  = tokenService.CreateRefreshToken();
```

Рис. 3.8. Код створення токенів

На рисунку 3.9. наведено код методу авторизації, завдання якого перевірити дані користувача та вислати пару токенів в разі успіху.

```
[HttpPost( template: ":login")]
[ProducesResponseType(typeof(RefreshAccessToken), statusCode: StatusCodes.Status200OK)]
[ProducesResponseType( statusCode: StatusCodes.Status400BadRequest)]
Tenshi
public async Task<IActionResult> Login([FromBody] LoginDTO loginDto)
{
    var result :RefreshAccessToken?  = await authenticationService.Login(loginDto);
    return result is not null ? Ok(result) : Unauthorized();
}
```

Рис. 3.9. Код методу авторизації

Варто зауважити, що в системі використовується ресурсно-орієнтована авторизація. Це механізм, який дозволяє перевіряти права доступу до конкретного ресурсу, а не просто до маршруту або ролі. Це більш гнучкий і безпечний підхід у порівнянні з ролями або політиками. Простішими словами, це означає, що користувач має певну роль не в межах всієї системи, а саме в межах проекту, наприклад, один користувач може бути адміністратором свого проекту, а в іншому проекті звичайним працівником, або менеджером.

Для забезпечення ресурсно-орієнтованої авторизації необхідні дві речі: клас, який визначає правила авторизації, та перевірка авторизації для методу. Для першої речі слугує функція, код якої продемонстровано на рисунку 3.10. Вона перевіряє правила, та надає, або забороняє доступ до методу, в залежності від прав користувача в межах даного проекту (ресурсу).

```
private void CheckRules(AuthorizationHandlerContext context, OperationAuthorizationRequirement requirement, Guid projectId)
{
    (string? controller, string? action) = GetContextMetadata();

    var currentUser =  userManager.FindByEmailAsync(context.User.Identity!.Name!).Result;
    if(currentUser is null) context.Fail();

    var userProject :UsersProjects? = db.UsersProjects // DbSet<UsersProjects>
        .Include( navigationPropertyPath: p :UsersProjects => p.Rules) // IIncludableQueryable<UsersProjects,IList<...>>
        .ThenInclude(p :Rule =>p.Operation) // IIncludableQueryable<UsersProjects,Operation>
        .FirstOrDefault(p :UsersProjects => currentUser != null && p.ProjectId == projectId && p.UserId == currentUser.Id);
    if(userProject is null) context.Fail();

    var rules :IList<Rule>? = userProject?.Rules;
    if(rules is null) context.Fail();

    if (rules != null && rules.Any(p :Rule => p.Operation.Controller == controller && p.Operation.Action == action && p.Access))
        context.Succeed(requirement);
}
```

Рис. 3.10. Функція перевірки прав доступу до проекту

Що стосується другого пункту, то для його забезпечення необхідно два рядка коду, роль якого перевірити авторизацію в межах даного ресурсу і повернути результат. Цей код продемонстровано на рисунку 3.11.

```
var authorizationResult = await authorizationService.AuthorizeAsync(User, fileWriteDto.ProjectId, APIOperations.FilePost);
if (!authorizationResult.Succeeded) return Forbid();
```

Рис. 3.11. Код перевірки прав доступу до певного ресурсу

Наступним кроком доречно розглянути додавання нового користувача до проекту. Цей модуль також використовує механізм JWT токенів, але передає в них не дані авторизації, а дані запрошення до проекту. Блок-схема алгоритму приєднання нового користувача да проекту наведена на рисунку 3.11. Після того, як новий користувач отримує свій токен-запрошення, він відкриває спеціальну сторінку і далі відбувається валідація токену. В разі якщо токен валідовано успішно і ніяких помилок не виявлено новий користувач додається до проекту та отримує повідомлення про своє успішно приєднання. Якщо ж на одному із етапів відбулася помилка, генерується відповідне повідомлення та надається користувачеві. В такому разі алгоритм закінчується і потрібно починати нову спробу.

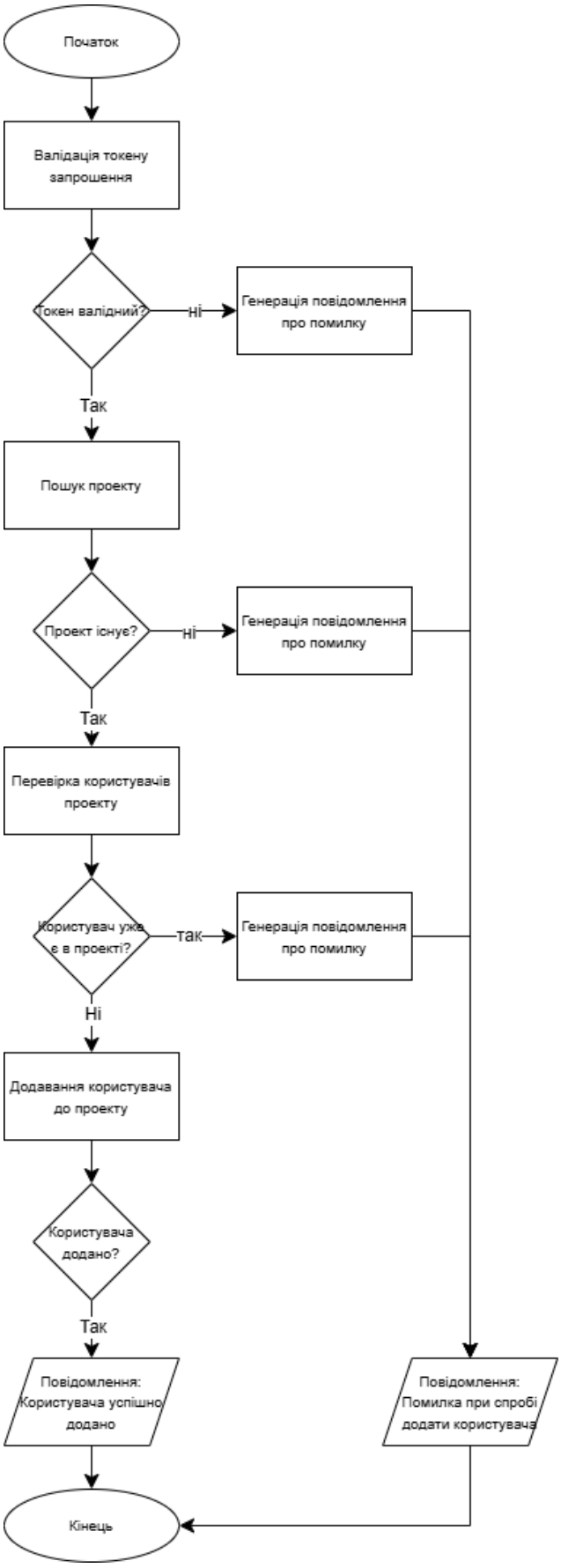

Рис. 3.11. Блок-схема алгоритму приєднання нового користувача до системи

Під час виконання одного із останніх пунктів алгоритму «Додавання користувача до проекту» відбувається транзакція. Транзакція — це послідовність операцій, які виконуються як одне неподільне ціле. У контексті баз даних транзакція гарантує, що всі дії всередині неї або успішно виконуються повністю, або не виконуються взагалі, тобто відкочуються у разі помилки. Entity Framework Core надає механізм для транзакцій за допомогою методу BeginTransactionAsync. На рисунку 3.12. зображено код такої транзакції. Створюється нова транзакція в тілі якої відбувається декілька операцій, а саме: додавання користувача до проекту, створення правил доступу для нового користувача та надання цих правил користувачеві. Якщо під час однієї операції виникне помилка вся логіка транзакції відкотиться і таким чином дані залишаться непошкодженими.

```
await using var transaction = await db.Database.BeginTransactionAsync();
try
{
    //add user to project
    var userProject = new UsersProjects()
    {
        UserId = user.Id,
        ProjectId = project.Id
    };
    db.UsersProjects.Add(userProject);

    //create rules for user
    var operations :List<Operation> = await db.Operations.ToListAsync();
    foreach (var operation in operations)
    {
        var rule = new Rule() { OperationId = operation.Id, Access = false };
        db.Rules.Add(rule);
        userProject.Rules.Add(rule);
    }

    await db.SaveChangesAsync();
    await transaction.CommitAsync();
    return true;
}
catch (Exception e)
{
    await transaction.RollbackAsync();
    throw new ProjectServiceException(e.Message);
}
```

Рис. 3.12. Код транзакції додавання нового користувача до проекту

3.5. Висновки до розділу 3

Під час роботи над третім розділом першим кроком було вибрано систему управління інформаційною базою даних. Перевагу надано СУБД PostgreSQL. Було наведено її функціонал, та переваги, за які вона була обрана, а саме: об'єктно-реляційна модель, розширюваність, масштабованість, підтримка великої кількості різних типів даних, а також продуктивність.

Наступним кроком було обґрунтовано вибір інструментарію для створення прикладного програмного забезпечення. В якості основної мови програмування було обрано C# 8 версії, а основною технологією для розробки логіки серверної частини виступив фреймворк Asp.Net Для роботи з базою даних було вибрано ORM Entity Framework Core, а для створення клієнтської частини використовувалися HTML, CSS, JS, React.

Після вибору інструментарію та СУБД було описано процес розробки інформаційної бази. Оскільки використовується технологія Entity Framework Core з підходом Code First - то спочатку було описано процес створення класів, які представлятимуть таблиці в базі даних, потім класи конфігурацій моделей і контекст бази даних.

В останній частині було наведено блок-схеми алгоритмів аутентифікації та приєднання нового користувача до проекту. Також розроблено та наведено відповідні фрагменти коду, які реалізують цю бізнес-логіку. Увагу було приділено механізму JWT токенів, який активно використовується як при аутентифікації, так і при створенні запрошень для нових користувачів.

Підсумовуючи, можна сказати, що у результаті виконання третього розділу було реалізовано повноцінну архітектуру програмної системи, починаючи з обґрунтованого вибору СУБД та інструментів розробки, і завершуючи реалізацією ключових модулів. Застосовані підходи та технології забезпечили ефективну взаємодію між клієнтською та серверною частинами, надійне управління даними, а також високий рівень безпеки завдяки використанню сучасних механізмів аутентифікації та авторизації.

РОЗДІЛ 4

4.1. Тестування системи

Важливим етапом розробки системи є її тестування. Тестування програмного забезпечення (англ. Software Testing) – техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки.

Тестування може бути:

- Модульним. Перевіряти окремі функції чи класи
- Інтеграційним. Тестувати взаємодії між модулями
- Системне. Перевіряти всю систему загалом
- Навантажувальним, стресовим, тощо.

Для тестування цього проекту використовувався інтеграційний тип тестування, при якому відбувалася перевірка взаємодії різних компонентів системи, а саме сервісів та бази даних. Для забезпечення тестування використовувався фреймворк xUnit. xUnit — це фреймворк для написання та запуску тестів. Він надає простий та потужний інтерфейс для створення тестових наборів, перевірки умов та автоматичного виявлення та запуску тестів. Також використовувалася бібліотека Shouldly, задача якої була зробити тести більш читабельними за допомогою простого синтаксису. Під час тестування важливу роль грав пакет Testcontainers.PostgreSql. Це бібліотека для запуску контейнерів Docker з PostgreSQL під час тестів. Вона дозволяє підняти тестову базу даних PostgreSQL у Docker-контейнері перед запуском тестів і знищити її після завершення, а отже при кожному тестуванні створюється окремий контейнер з окремою базою і система тестується в цілому, що дозволяє якісно протестувати взаємодію всіх об’єктів в системі.

Опис тестування буде наведено на прикладі функції Post. Функція приймає дані проекту та створює новий проект. В разу успіху вона повертає статус код 201 з створеним об’єктом, в разі помилки статус код – 400.

Першим кроком, під час тестування необхідно описати тестові випадки. Тестовий випадок – це опис того, що саме та як потрібно перевірити і який результат очікується. Тестові випадки для даної функції наведено в таблиці 4.1.

Таблиця 4.1. Тест кейси функції POST

Умови	**Очікуваний результат**
Коректне тіло запиту та ключ	Статус код 201 та створений об’єкт.
Повторний запит з однаковим тілом та ключем	Статус код 409 (конфлікт)
Повторний запит з новим тілом, але старим ключем	Статус код 201 та створений об’єкт.
Повторний запит з старим тілом, але новим ключем	Статус код 201 та створений об’єкт.
Запит з коректним тілом, але некоректним ключем	Статус код 400
Запит з некоректним тілом, але коректним ключем	Статус код 400

Після опису тестових випадків необхідно написати код тестування. Як уже було зазначено раніше, для цього використовується xUnit, Shouldly та Testcontainers. Для тестування використовується патерн arrange-act-assert – це структура написання тестів, що робить їх чіткими, логічними та зрозумілими. Arrange – це підготовка, створення об'єктів, налаштування оточення, задання вхідних даних. Act – це дія, виклик тестованого методу або функції. Assert – це перевірка результату на відповідність очікуваному. На рисунку 4.1. наведено код тестування функції з використанням даного шаблону. В функції відбувається підготовка тіла запиту, далі тіло передається в запиті, запит

виконується і повертає результат. В кінці відбувається перевірка результату запиту.

```
[Fact(DisplayName = "Success post with valid body and valid request id")]
Tenshi
public async Task SuccessPostWithValidBodyAndValidRequestId()
{
    //arrange
    var body = new ProjectWriteDTO()
    {
        Name = "Test valid body success post project",
        CreatedDateTime = DateTime.UtcNow,
        Description = null,
    };
    _httpClient.DefaultRequestHeaders.Add( name: "requestId", Guid.NewGuid().ToString());

    //act
    var response = await _httpClient.PostAsJsonAsync( requestUri: "project", body);
    var result = await response.Content.ReadFromJsonAsync<ProjectReadDTO>();

    //assert
    response.StatusCode.ShouldBe( expected: HttpStatusCode.Created);
    result.ShouldNotBeNull();
}
```

Рис. 4.1. Код тестування функції Post

Схожим чином відбувається тестування інших функцій. Підсумовуючи, можна сказати, що для тестування кожної функції є три етапи: підготовка тіла запиту, запит і порівняння результату запиту з очікуваним результатом. На рисунку 4.2. приведено результати тестування для функції Post. Як можна побачити, тестування пройшло успішно.

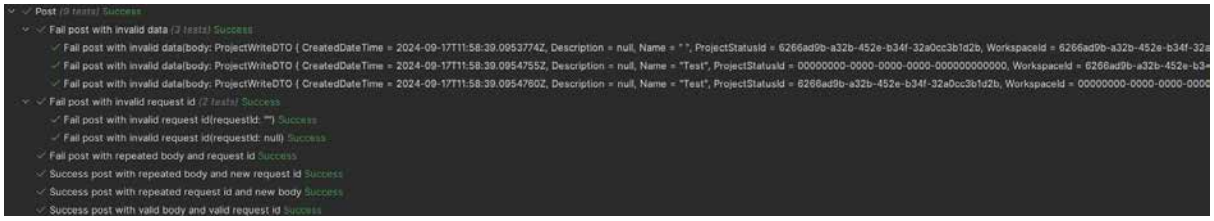

Рис. 4.2. Результати тестування функції Post

4.2. Вимоги до апаратного та програмного забезпечення

«Діаграма розміщення (deployment diagram) віддзеркалює фізичні взаємозв’язки між програмними й апаратними компонентами проектованої системи. Ця діаграма є гарним засобом для представлення маршрутів переміщення об’єктів і компонентів у розподіленій системі. Кожен вузол на

діаграмі розміщення є певним обчислювальним пристроєм (здебільшого самостійна частина апаратури). Ця апаратура може бути як простим пристроєм чи датчиком, так і мейнфреймом» [3, с.136].

На рисунку 4.3. зображено діаграму розгортання до проектованої системи. Першим компонентом виступає сервер клієнтського додатку, який являє собою хостинг для сайту. Сам клієнтський додаток спілкується з сервером API, який уже містить всю бізнес-логіку системи. Спілкування відбувається за допомогою HTTP, або HTTPS протоколу.

«HTTP (англ. HyperText Transfer Protocol) - стандартний інтернет-протокол прикладного рівня, який використовується відповідними серверами та клієнтами» [4, с.78].

«HTTPS (англ. HTTP over TLS, HTTP over SSL, чи HTTP Secure) — схема URI, що синтаксично ідентична HTTP. Така схема вказує, що протокол HTTP має використовуватися, але з іншим портом за замовчуванням (443) і додатковим шаром шифрування/автентифікації між HTTP і TCP» [4, с.78].

Простішими словами, HTTP – це протокол передачі даних в мережі, а HTTPS – його версія з шифруванням.

Сервер API спілкується з SMTP сервером за допомогою TCP/IP. SMTP сервер потрібен для функціонування модулю сповіщень за електронною адресою.

«SMTP (англ. Simple Mail Transfer Protocol, простий протокол передачі пошти) – це широко використовуваний мережевий протокол, призначений для передачі електронної пошти в мережах TCP/IP» [4, с.81].

Також сервер спілкується з сервером бази даних. Таким чином відбувається взаємодія компонентів системи на фізичному рівні.

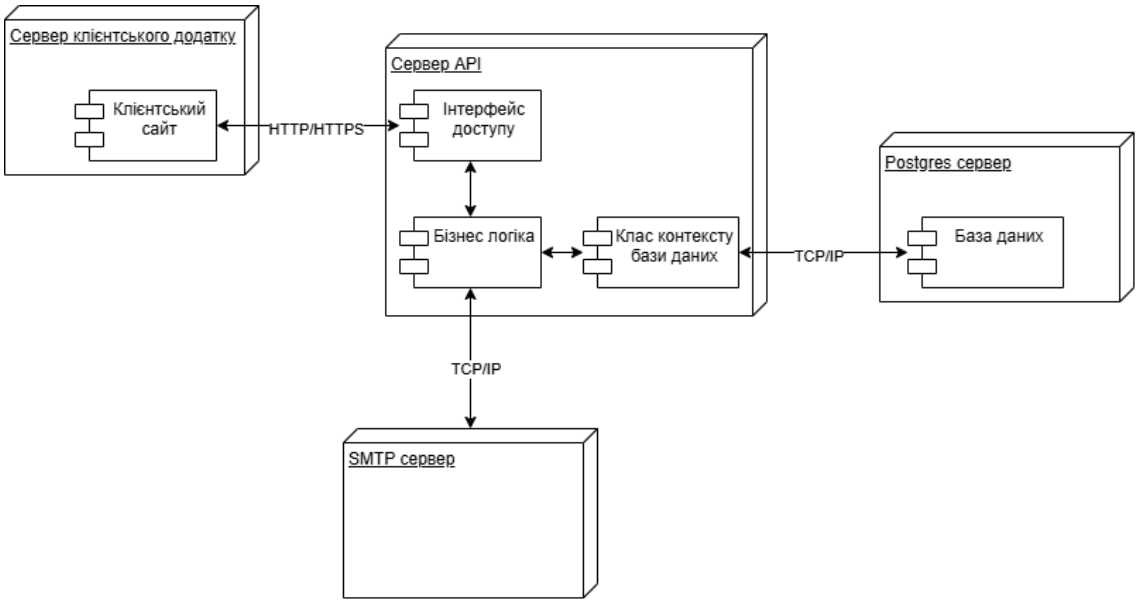

Рис. 4.3. Діаграма розгортання до проектованої системи

У відповідності до діаграми розгортання було визначено апаратні та програмні вимоги до кожного вузла системи. Вимоги наведено в таблиці 4.2.

Таблиця 4.2. Апаратні та програмні вимоги до ПЗ

Вузол	**Апаратні вимоги**	**Програмні вимоги**
Сервер клієнтського додатку	CPU 2 ядра RAM 2-4 ГБ Дисковий простір не менше 5 ГБ	ОС Ubuntu / Windows Встановлене ПЗ Node.js
Сервер API	CPU 2 ядра RAM 2-4 ГБ Дисковий простір не менше 5 ГБ	ОС Ubuntu / Windows Встановлене ПЗ .NET 8 SDK, ASP.NET Core
PostgreSQL сервер	CPU 2 ядра RAM 2-4 ГБ Дисковий простір не менше 5 ГБ	ОС Ubuntu / Windows Встановлене ПЗ PostgreSQL 15+

4.3. Склад інсталяційного пакету

Для інсталяції системи використовується інфраструктура контейнеризації Docker, зокрема — інструмент Docker Compose, який

дозволяє автоматизовано запускати всі необхідні сервіси, пов’язані між собою єдиною мережею. На рисунку 4.4. наведено код файлу docker-compose.yaml. Всі сервіси запускаються через цей файл:

- API. Власний образ ASP.NET Core Web API (збирається з Dockerfile). Експонується на порту 8080.
- DataBase. СУБД PostgreSQL. Встановлюються змінні середовища: логін/пароль адміністратора.
- Seq. Система централізованого логування Seq. Доступ через порти 8020 (інтерфейс) та 5341 (інтеграція з .NET).
- MinIO. Об’єктне сховище, сумісне з Amazon S3, використовується для зберігання файлів.

Усі сервіси підключені до єдиної віртуальної мережі Sen4Network (типу bridge), що забезпечує взаємодію між контейнерами без відкриття внутрішніх портів зовні.

Таким чином для введення програми в експлуатацію достатньо лише цього docker-compose файлу. Це значно спрощує розгортання, забезпечує повторюваність конфігурації та гарантує однакову роботу системи на будь-якому сервері-цільовому середовищі.

```yaml
networks:
  Sen4Network:
    driver: bridge

services:
  Sen4API:
    image: sen4
    ports:
      - 8080:8080
    networks:
      - Sen4Network
    depends_on:
      - Sen4Db
    build:
      context: .
      dockerfile: Sen4/Dockerfile

  Sen4Db:
    image: postgres:12.19
    ports:
      - 5433:5432
    networks:
      - Sen4Network
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: root

  Sen4Seq:
    image: datalust/seq:latest
    container_name: seq
    restart: unless-stopped
    mem_limit: 5g
    memswap_limit: 5g
    environment:
      - ACCEPT_EULA=Y
    networks:
      - Sen4Network
    ports:
      - 8020:80
      - 5341:5341

  minio:
    image: minio/minio:latest
    command: server --console-address ":9001" /data/
    networks:
      - Sen4Network
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      MINIO_ROOT_USER: ozontech
      MINIO_ROOT_PASSWORD: minio123
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://localhost:9000/minio/health/live" ]
      interval: 30s
      timeout: 20s
      retries: 3
```

Рис. 4.4. Код файлу розгортання docker-compose.yaml

4.4. Висновки до розділу 4

Під час роботи над четвертим розділом було розглянуто питання тестування системи, визначення вимог до апаратного та програмного забезпечення та визначення складу інсталяційного пакету.

Розглядаючи питання тестування було визначено, що таке тестування, які є типи тестування та який тип обрано для даного проекту. Також було визначено фреймворки та бібліотеки для тестування в межах даної системи, та наведено опис тестування на прикладі функції Post. Було створено тестові випадки, відповідний код, а також проаналізовано результати.

Наступним кроком було визначено вимоги до апаратного та програмного забезпечення. Створено діаграму розміщення, яка представляє фізичні взаємозв'язки між апаратними й програмними компонентами системи. На основі цієї діаграми було визначено апаратні та програмні вимоги до кожного вузла в системі.

Останнім кроком було визначено склад інсталяційного пакету. Використовувалася система Docker та файл docker-compose.yaml, що спрощує розгортання, забезпечує повторюваність конфігурації та гарантує однакову роботу системи на будь-якому сервері-цільовому середовищі.

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи було пройдено багато етапів. Першим кроком відбувся аналіз предметної області та програм аналогів. Було зроблено висновок, що успіх компанії сьогодні значною мірою залежить від здатності ефективно реалізовувати проекти. Тому облік виконання завдань в проектах є одною з ключових цілей для менеджерів усіх рівнів. Також було описано деякі методології управління проектами, які можуть використовуватися в подібного роду системах. На основі існуючих програм аналогів та потреб користувачів було виявлено бізнес вимоги, функціональні та нефункціональні вимоги до проектованої системи. Наступним кроком відбулося моделювання предметної області: було виділено основні абстракції такої системи та основних користувачів та прецедентів на основі вимог. Для розуміння деяких процесів, що відбувається в системі, створено діаграму послідовності, яка демонструє потік даних в певних процесах системи.

На наступному етапі роботи будувалась логічна модель даних у вигляді ER діаграми, на основі якої відбувалося подальше проектування та розробка програмної системи. На початковому етапі така діаграма дозволила побачити нариси та отримати початкове бачення системи. Наступим кроком проектування було створення діаграми класів з використанням простих асоціацій. Така діаграма включала лише класи та асоціації між ними. Це дозволило зосередитися на розумінні логіки предметної області, визначити ключові сутності (класи), їх ролі та взаємозв’язки без ускладнення технічними деталями. В подальшому була побудована повноцінна діаграма класів. Також, під час цього етапу були створені діаграми пакетів, станів та компонентів. Підсумовуючи, можна сказати, що другий другий розділ був присвячений етапу проектування програмної системи. На основі зроблених моделей та діаграм в подальшому була розроблена програмна система.

В третьому розділі було обрано систему управління інформаційною базою. Перевага надалась обрано PostgreSQL — потужній, вільно розповсюджуваній реляційній СУБД з відкритим вихідним кодом. Було наведено список її функціональних можливостей та переваг. Далі було обґрунтовано вибір інструментарію для створення прикладного програмного забезпечення. Перевага надалась продуктам компанії Microsoft, а саме - мові програмування C# 8 версії, фреймворкам Entity Framework Core та Asp.net Core, які дозволили створити код серверної бізнес логіки, який задовільнив функціональним потребам. Для побудови інтерфейсу користувача використовувався наступний стек технологій: HTML, CSS, JS, React. В цьому ж розділі було описано розробку інформаційної бази даних. Оскільки для роботи з БД використовувалася ORM та підхід Code First, це означало, що спочатку мають бути створені прості класи, які будуть відображати сутності в базі даних.

Під час алгоритмізації та розробки програмних модулів було наведено декілька блок-схем алгоритмів та код їх реалізації. Особливу увагу було надану механізму JWT за допомогою якого в системі відбувається аутентифікація та генерація посилань-запрошень.

Важливим етапом під час роботи було тестування системи. Для тестування програмного забезпечення було вибрано інтеграційний підхід. Для забезпечення тестування використовувався фреймворк xUnit екосистеми .Net та бібліотека Shouldly, задача якої була зробити тести більш читабельними за допомогою простого синтаксису. Створені тест кейси допомогли написати відповідний код, який тестував бізнес логіку роботи програми, що зробило проектовано систему більш надійнішою та стабільнішою до помилок.

Під час визначення вимог до апаратного та програмного забезпечення, першим кроком було побудовано діаграму розгортання, яка демонструвала фізичні взаємозв'язки між компонентами системи. На основі цієї діаграми було визначено апаратні та програмні вимоги до кожного вузла системи. Наступним кроком було описано склад інсталяційного пакету, підсумовуючи

можна сказати, що для введення програми в експлуатацію достатньо лише цього docker-compose файлу. Це значно спрощує розгортання, забезпечує повторюваність конфігурації та гарантує однакову роботу системи на будь-якому сервері-цільовому середовищі.

У результаті реалізації проекту було створено програмний продукт, який задовольняє вимоги до розробки прикладного програмного забезпечення. Його функціонал дозволяє ефективно автоматизувати ключові бізнес-процеси в межах діяльності підприємства. Обрана структура системи та підхід до її побудови забезпечують зручну можливість подальшого розширення. Зокрема, за необхідності можна без значних труднощів додати нові функції, такі як аналітика із використанням методів штучного інтелекту, без ризику порушення вже працюючих компонентів.

Підсумовуючи, розроблена система відповідає всім вимогам і стає практичним втіленням знань і досвіду, отриманих протягом навчання. Розроблений застосунок охоплює всі ключові вимоги предметної області, та відповідає сучасним стандартам щодо захисту даних, зручності використання й організації інтерфейсу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сидоренко В. В., Константинова Л. В., Смірнов С. А. Організація баз даних: навчальний посібник. – Кропивницький: ЦНТУ, 2018. – 274 с.

2. Погромська Г. С., Махровська Н. А. Бази даних: проектування та реалізація : навчально-методичний посібник для студентів спеціальності 122 «Комп'ютерні науки» / Г. С. Погромська, Н. А. Махровська. — Миколаїв : Видавництво, 2019. — 183 с.

3. Дудзяний І.М. Об'єктно-орієнтоване моделювання програмних систем: Навчальний посібник. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. – 108 с.

4. Коваль Ю. В. Інформаційні мережі: навчальний посібник / Ю. В. Коваль, А. Б. Ставровський. – Київ, 2021. – 84 с.

5. Salunke S. V., Ouda A. A Performance Benchmark for the PostgreSQL and MySQL Databases // Future Internet. – 2024. – Vol. 16, No. 10. – Article № 382.

6. Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення. Навчальний посібник. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284 с.

7. Життєвий цикл програмного забезпечення. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level15.lecture00

8. Коноваленко І.В. Платформа .NET та мова програмування C# 8.0: навчальний посібник / Коноваленко І.В., Марущак П.О. – Тернопіль: ФОП Паляниця В. А., 2020 – 320 с

9. Entity Framework Core 7. URL: https://abitap.com/1-1-entity-framework-core-6/

10. PostgreSQL: About. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/about/

11. Freeman, A. Pro ASP.NET Core MVC 2 / Adam Freeman. – 7th ed. – New York: Apress, 2017. – 1017 p. – ISBN 978-1-4842-2601-1.

Додаток А. Код серверної частини

```
using Infrastructure.DTO;
using Infrastructure.Interfaces;
using Microsoft.Extensions.Logging;
using Minio;
using Minio.DataModel;
using Minio.DataModel.Args;
using Minio.DataModel.Encryption;
using Minio.DataModel.Response;

namespace Infrastructure.Services;


public sealed class FileService(IMinioClient minioClient, ILogger<FileService> logger): IFileService
{

    private async Task MakeBucketIfNotExists(string name)
    {
        var bucket = new BucketExistsArgs()
            .WithBucket(name);
        if (!await minioClient.BucketExistsAsync(bucket))
        {
            var makeBucketArgs = new MakeBucketArgs()
                .WithBucket(name);
            await minioClient.MakeBucketAsync(makeBucketArgs);
        }
    }

    public async Task<string?> GetObjectUrl(string objectName, string bucketName)
    {
        var args = new PresignedGetObjectArgs()
            .WithBucket(bucketName)
            .WithObject(objectName)
            .WithExpiry(1000);

        return await minioClient.PresignedGetObjectAsync(args);
    }


    public async Task<MemoryStream?> GetObject(string objectName, string bucketName)
    {
        try
        {
            var memoryStream = new MemoryStream();

            var getArguments = new GetObjectArgs()
                .WithBucket(bucketName)
                .WithObject(objectName)
                .WithCallbackStream(async (stream, cancellationToken) =>
                {
                    await stream.CopyToAsync(memoryStream, cancellationToken);
                });

            await minioClient.GetObjectAsync(getArguments);
            memoryStream.Position = 0;

            return memoryStream;
        }
        catch (Exception e)
        {
            logger.LogError(e.Message);
            return null;
        }
    }


    public async Task RemoveObject(string objectName, string bucketName)
    {
        var removeArguments = new RemoveObjectArgs()
            .WithBucket(bucketName)
            .WithObject(objectName);

        await minioClient.RemoveObjectAsync(removeArguments);
    }


    public async Task<PutObjectResponse> PutObject(FileWriteDTO fileWriteDto, Dictionary<string, string?>
metaData, IProgress<ProgressReport>? progress = null, IServerSideEncryption? sse = null)
    {
        await MakeBucketIfNotExists(fileWriteDto.ProjectId.ToString());
        var file = fileWriteDto.File;

        //upload file
        var filestream = fileWriteDto.File.OpenReadStream();

        var putArguments = new PutObjectArgs()
            .WithBucket(fileWriteDto.ProjectId.ToString())
            .WithObject(file.FileName)
```

```
            .WithStreamData(filestream)
            .WithObjectSize(filestream.Length)
            .WithContentType("application/octet-stream")
            .WithHeaders(metaData)
            .WithProgress(progress)
            .WithServerSideEncryption(sse);

        return await minioClient.PutObjectAsync(putArguments);
    }


    public async Task<List<Item>> ListProjectsObject(string? name, string? projectId, string? taskId, string?
prefix = null, bool recursive = true, bool versions = false)
    {
        await MakeBucketIfNotExists(projectId);

        var metaData = new Dictionary<string, string?>(StringComparer.Ordinal)
        {
            { "Project", projectId },
        };
        if (taskId is not null)
            metaData.Add("Task", taskId);

        var listArgs = new ListObjectsArgs()
            .WithBucket(projectId)
            .WithPrefix(prefix)
            .WithRecursive(recursive)
            .WithIncludeUserMetadata(true)
            .WithHeaders(metaData)
            .WithVersions(versions);

        var items = new List<Item>();
        await foreach (var item in minioClient.ListObjectsEnumAsync(listArgs))
            items.Add(item);

        if (name is not null)
            items = items.Where(p => p.UserMetadata.TryGetValue("Name", out string? itemName) && itemName ==
name).ToList();
        if (projectId is not null)
            items = items.Where(p => p.UserMetadata.TryGetValue("Project", out string? project) && project ==
projectId).ToList();
        if (taskId is not null)
            items = items.Where(p => p.UserMetadata.TryGetValue("Task", out string? task) && task ==
taskId).ToList();

        return items;
    }
}

public class ProjectService(IConfiguration configuration,Sen4Context db, IMapper mapper, UserManager<User>
userManager, ITokenService tokenService): IProjectService
{
    public string GenerateInviteToken(Guid projectId)
    {
        var claims = new List<Claim>() { new Claim(type: "ProjectId", value: projectId.ToString()) };
        var jwtSecret = AppConfiguration.GetRequiredConfigurationValue(configuration, "InviteJWT:Secret");
        var issuer = AppConfiguration.GetRequiredConfigurationValue(configuration, "InviteJWT:ValidIssuer");
        var audience = AppConfiguration.GetRequiredConfigurationValue(configuration,
"InviteJWT:ValidAudience");
        var expires =
DateTime.UtcNow.AddMinutes(int.Parse(AppConfiguration.GetRequiredConfigurationValue(configuration,
"InviteJWT:ValidTime")));

        var token = tokenService.CreateJwtToken(claims, jwtSecret, issuer, audience, expires);
        return token;
    }

    private ClaimsPrincipal ValidateToken(string token)
    {
        var jwtSecret = AppConfiguration.GetRequiredConfigurationValue(configuration, "InviteJWT:Secret");
        var validationParameters = new TokenValidationParameters()
        {
            ValidateIssuer = false,
            ValidateAudience = false,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSecret)),
            ClockSkew = new TimeSpan(0, 0, 5),
            ValidateLifetime = true
        };

        return tokenService.ValidateToken(token, validationParameters, jwtSecret);
    }

    public async Task<bool> JoinToProject(string token, string userEmail)
    {
        //validate token
        ClaimsPrincipal claimsPrincipal;
        try
        {
            claimsPrincipal = ValidateToken(token);
        }
```

```
    catch (Exception e)
    {
        return false;
    }

    //get project id
    var id = claimsPrincipal.Claims
        .Where(p => p.Type == "ProjectId")
        .Select(p => p.Value)
        .SingleOrDefault();
    if (id is null) return false;

    //search project by id
    var project = await db.Projects
        .Include(p => p.Users)
        .Where(p => p.Id == new Guid(id))
        .FirstOrDefaultAsync();
    if (project is null) return false;

    //get current user
    var user = await userManager.FindByEmailAsync(userEmail);
    if (user is null) return false;

    //check if the user has already been added
    var usersProjects =
        await db.UsersProjects.FirstOrDefaultAsync(p => p.UserId == user.Id && p.ProjectId == project.Id);
    if (usersProjects is not null) return false;

    await using var transaction = await db.Database.BeginTransactionAsync();
    try
    {
        //add user to project
        var userProject = new UsersProjects()
        {
            UserId = user.Id,
            ProjectId = project.Id
        };
        db.UsersProjects.Add(userProject);

        //create rules for user
        var operations = await db.Operations.ToListAsync();
        foreach (var operation in operations)
        {
            var rule = new Rule() { OperationId = operation.Id, Access = false };
            db.Rules.Add(rule);
            userProject.Rules.Add(rule);
        }

        await db.SaveChangesAsync();
        await transaction.CommitAsync();
        return true;
    }
    catch (Exception e)
    {
        await transaction.RollbackAsync();
        throw new ProjectServiceException(e.Message);
    }
}

public int ProjectCount() => db.Projects.Count();
public async Task<Guid?> Archiving(Guid id)
{
    var project = await db.Projects.FindAsync(id);
    if (project is null) return null;
    if (project.DeletedBy is not null) return null;

    project.DeletedBy = DateTime.UtcNow;
    db.Projects.Update(project);
    await db.SaveChangesAsync();

    return project.Id;
}

public async Task<ProjectReadDTO> Get(Guid id)
{
    var project = await db.Projects
        .Where(p => p.Id == id).FirstOrDefaultAsync();
    return mapper.Map<ProjectReadDTO>(project);
}

public async Task<ProjectReadDTO> Create(ProjectWriteDTO projectWriteDto, Guid userId)
{
    await using var transaction = await db.Database.BeginTransactionAsync();
    try
    {
        //create project
        var project = mapper.Map<Project>(projectWriteDto);
        db.Projects.Add(project);

        //attach current user to project
```

```
            var userProject = new UsersProjects() { UserId = userId, ProjectId = project.Id };
            db.UsersProjects.Add(userProject);

            //create max rules for user
            var operations = await db.Operations.ToListAsync();
            foreach (var operation in operations)
            {
                var rule = new Rule() { OperationId = operation.Id, Access = true };
                db.Rules.Add(rule);
                userProject.Rules.Add(rule);
            }

            await db.SaveChangesAsync();
            await transaction.CommitAsync();

            return mapper.Map<ProjectReadDTO>(project);
        }
        catch (Exception e)
        {
            await transaction.RollbackAsync();
            throw new ProjectServiceException(e.Message);
        }
    }

    public async Task<Guid?> PatchUpdate(Guid id,JsonPatchDocument<ProjectWriteDTO> projectUpdateDto)
    {
        var project = await db.Projects.FindAsync(id);
        if (project is null) return null;

        var updateProject = mapper.Map<JsonPatchDocument<Project>>(projectUpdateDto);
        updateProject.ApplyTo(project);
        db.Projects.Update(project);
        await db.SaveChangesAsync();

        return project.Id;
    }

    public async Task<PaginatedList<ProjectReadDTO>> List(ProjectListRequest projectListRequest)
    {
        var filteredQuery = db.Projects
            .AsNoTracking()
            .Filtration(id: projectListRequest.UserId,
                name: projectListRequest.Name,
                showDeleted: projectListRequest.ShowDeleted);

        var projectsTotalCount = await filteredQuery.CountAsync();

        var projects = await filteredQuery
            .Sort(projectListRequest.SortProperty, projectListRequest.SortByDescending)
            .Pagination(projectListRequest.PageSize, projectListRequest.PageNumber)
            .ToListAsync();

        var mappedProjects = mapper.Map<List<ProjectReadDTO>>(projects);

        int? totalPage = (projectListRequest.PageNumber == null || projectListRequest.PageSize == null)
            ? null
            : (int)Math.Ceiling(projectsTotalCount / (double)projectListRequest.PageSize);

        return new PaginatedList<ProjectReadDTO>(
            list: mappedProjects,
            pageIndex: projectListRequest.PageNumber,
            totalPages: totalPage);
    }

}
public class TokenService: ITokenService
{
    public ClaimsPrincipal ValidateToken(string token,TokenValidationParameters validationParameters, string
jwtSecret)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        return tokenHandler.ValidateToken(token, validationParameters, out _);
    }

    public string CreateJwtToken(List<Claim> claims, string jwtSecret, string issuer, string audience,
DateTime? expires)
    {
        var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSecret));

        var token = new JwtSecurityToken(
            issuer: issuer,
            audience: audience,
            expires: expires,
            claims: claims,
            signingCredentials: new SigningCredentials(symmetricSecurityKey, SecurityAlgorithms.HmacSha256)
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
```

```
    public ClaimsPrincipal? GetClaimsPrincipalFromExpiredToken(string token, string secret,
TokenValidationParameters validationParameters)
    {
        try
        {
            var result = new JwtSecurityTokenHandler().ValidateToken(token, validationParameters, out _);
            return result;
        }
        catch (Exception e)
        {
            return null;
        }
    }

    public string CreateRefreshToken()
    {
        var token = new byte[64];
        using var generator = RandomNumberGenerator.Create();

        generator.GetBytes(token);
        return Convert.ToBase64String(token);
    }
}
namespace Sen4.Controllers;

[Authorize]
[ApiController]
[Route("[controller]")]
public class FileController(IAuthorizationService authorizationService, IFileService fileService):
ControllerBase
{

    [HttpGet()]
    [Description("This method return file.")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    public async Task<IActionResult> Get(string objectName, string bucketName)
    {
        var authorizationResult = await authorizationService.AuthorizeAsync(User, new Guid(bucketName),
APIOperations.FileGet);
        if (!authorizationResult.Succeeded) return Forbid();

        var result = await fileService.GetObject(objectName, bucketName);
        return result is null ? BadRequest() : File(result, "application/octet-stream", objectName);
    }


    [HttpDelete("{objectName}/{bucketName}")]
    [Description("This method delete file.")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    public async Task<IActionResult> Delete(string objectName, string bucketName)
    {
        var authorizationResult = await authorizationService.AuthorizeAsync(User, new Guid(bucketName),
APIOperations.FileDelete);
        if (!authorizationResult.Succeeded) return Forbid();

        await fileService.RemoveObject(objectName, bucketName);
        return Ok();
    }

    /// <summary>
    /// Upload file in storage
    /// </summary>
    /// <param name="fileWriteDto"></param>
    /// <response code="200">Success</response>
    /// <response code="400">Bad request</response>
    /// <response code="401">Unauthorized</response>
    /// <response code="403">Forbidden</response>
    [HttpPost]
    [Description("This method upload file.")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    public async Task<IActionResult> Post([FromForm] FileWriteDTO fileWriteDto)
    {
        var authorizationResult = await authorizationService.AuthorizeAsync(User, fileWriteDto.ProjectId,
APIOperations.FilePost);
        if (!authorizationResult.Succeeded) return Forbid();

        var metaData = new Dictionary<string, string?>(StringComparer.Ordinal)
        {
            { "Project", fileWriteDto.ProjectId.ToString() },
        };
        if (fileWriteDto.TaskId is not null)
```

```
            metaData.Add("Task", fileWriteDto.TaskId.ToString());

        var result = await fileService.PutObject(fileWriteDto,metaData);
        return StatusCode((int)result.ResponseStatusCode);
    }

    /// <summary>
    /// Get file list from storage
    /// </summary>
    /// <param name="projectId">Project id</param>
    /// <param name="taskId">Task id</param>
    /// <response code="200">Success</response>
    /// <response code="400">Bad request</response>
    /// <response code="401">Unauthorized</response>
    /// <response code="403">Forbidden</response>
    [HttpGet("List")]
    [Description("This method return file list.")]
    [ProducesResponseType(typeof(List<Item>),StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    public async Task<IActionResult> List(string? name, string projectId, string? taskId)
    {
        var authorizationResult = await authorizationService.AuthorizeAsync(User, new Guid(projectId),
APIOperations.FileList);
        if (!authorizationResult.Succeeded) return Forbid();

        var result = await fileService.ListProjectsObject(
            name: name,
            projectId: projectId,
            taskId: taskId);
        return Ok(result);
    }

    /// <summary>
    /// Method not implemented.
    /// </summary>
    [HttpPut]
    [ProducesResponseType(StatusCodes.Status405MethodNotAllowed)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    public IActionResult Put()
    {
        return StatusCode(StatusCodes.Status405MethodNotAllowed);
    }

    /// <summary>
    /// Method not implemented.
    /// </summary>
    [HttpPatch]
    [ProducesResponseType(StatusCodes.Status405MethodNotAllowed)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    public IActionResult Patch()
    {
        return StatusCode(StatusCodes.Status405MethodNotAllowed);
    }
}
```

Додаток Б. Код тестування логіки серверної частини

```
public class Delete: TestBase
{
    public Delete(IntegrationTestWebAppFactory integrationTestWebAppFactory):
base(integrationTestWebAppFactory)
    {
        //register main user
       _authorizationHelper.RegisterUserAsync("admin@gmail.com","Alex","Hlushko","Olegovich",new
Guid("6266ad9b-a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
        //register guest user
        _authorizationHelper.RegisterUserAsync("guest@gmail.com","John","Doe","Ragnarson",new Guid("6266ad9b-
a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
    }


    [Fact(DisplayName = "Success delete with auth user and access")]
    public async Task SuccessDeleteWithAuthUserAndAccess()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("SuccessDeleteWithAuthUserAndAccess");

        //act
        var response = await _httpClient.DeleteAsync($"Project/{project.Id}");
        var result = await response.Content.ReadFromJsonAsync<Guid>();

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
        result.ShouldBeEquivalentTo(project.Id);
    }

    [Fact(DisplayName = "Fail delete with non auth user")]
    public async Task FailDeleteWithNonAuthUser()
    {
        //act
        var response = await _httpClient.DeleteAsync($"Project/{Guid.NewGuid().ToString()}");

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Unauthorized);
    }

    [Fact(DisplayName = "Fail delete with auth user and non access")]
    public async Task FailDeleteWithAuthAndNonAccess()
    {
        //assert
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("FailDeleteWithAuthAndNonAccess");

        //act
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var response = await _httpClient.DeleteAsync($"Project/{project.Id}");

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Forbidden);
    }

    [Fact(DisplayName = "Success delete with added user and added rule")]
    public async Task SuccessDeleteWithAddedUSerAndAddedRule()
    {
        //create project
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("SuccessDeleteWithAddedUSerAndAddedRule");

        //create invite token
        var postResponse = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var token = await postResponse.Content.ReadAsStringAsync();

        //login as new user and join to project
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        await _httpClient.PostAsync($"Project/:join/{token}", null);

        //login as admin and add "delete" rule for new user
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");

        var operation = await _sen4Context.Operations.FirstOrDefaultAsync(p => p.Controller == "Project" &&
p.Action == "Delete");
        var userId = _userManager.FindByEmailAsync("guest@gmail.com").Result.Id;
        await _projectHelper.SetRuleAsync(userId, project.Id, operation.Id);

        //login as new user and check delete rule
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var response = await _httpClient.DeleteAsync($"Project/{project.Id}");

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
    }
```

```
    [Fact(DisplayName = "Fail delete duplicate with auth user and access")]
    public async Task FailDeleteDuplicateWithAuthAndAccess()
    {
        //assert
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("FailDeleteDuplicateWithAuthAndAccess");
        await _httpClient.DeleteAsync($"Project/{project.Id}");

        //act
        var response = await _httpClient.DeleteAsync($"Project/{project.Id}");

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Conflict);
    }
}
public class Get: TestBase
{
    public Get(IntegrationTestWebAppFactory integrationTestWebAppFactory): base(integrationTestWebAppFactory)
    {
        //register main user
        _authorizationHelper.RegisterUserAsync("admin@gmail.com","Alex","Hlushko","Olegovich",new
Guid("6266ad9b-a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
        //register guest user
        _authorizationHelper.RegisterUserAsync("guest@gmail.com","John","Doe","Ragnarson",new Guid("6266ad9b-
a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
    }

    [Fact(DisplayName = "Success get by id. User with access.")]
    public async Task SuccessGetByIdUSerWithAccess()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("SuccessGetByIdUSerWithAccess");

        //act
        var response = await _httpClient.GetAsync($"Project/{project.Id}");
        var result = await response.Content.ReadFromJsonAsync<ProjectReadDTO>();

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
        result.ShouldNotBeNull();
    }

    [Fact(DisplayName = "Success get with added user and added rule")]
    public async Task SuccessGetAddedUSerAndAddedRule()
    {
        //create project
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("SuccessGetAddedUSerAndAddedRule");

        //create invite token
        var postResponse = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var token = await postResponse.Content.ReadAsStringAsync();

        //login as new user and join to project
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        await _httpClient.PostAsync($"Project/:join/{token}", null);

        //login as admin and add "delete" rule for new user
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");

        var operation = await _sen4Context.Operations.FirstOrDefaultAsync(p => p.Controller == "Project" &&
p.Action == "Get");
        var userId = _userManager.FindByEmailAsync("guest@gmail.com").Result.Id;
        await _projectHelper.SetRuleAsync(userId, project.Id, operation.Id);

        //login as new user and check get rule
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var response = await _httpClient.GetAsync($"Project/{project.Id}");
        var result = await response.Content.ReadFromJsonAsync<ProjectReadDTO>();

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
        result.ShouldNotBeNull();
    }

    [Fact(DisplayName = "Fail get by id without auth")]
    public async Task FailGetByIdWithoutAuth()
    {
        //act
        var response = await _httpClient.GetAsync($"Project/{Guid.NewGuid().ToString()}");

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Unauthorized);
    }

    [Fact(DisplayName = "Fail get by id with auth user but not access")]
    public async Task FailGetByIdWithAuthUserButNotAccess()
    {
```

```
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("FailGetByIdWithAuthUserButNotAccess");
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");

        //act
        var response = await _httpClient.GetAsync($"Project/{project.Id}");

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Forbidden);
    }
}
public class Invite: TestBase
{
    public Invite(IntegrationTestWebAppFactory integrationTestWebAppFactory):
base(integrationTestWebAppFactory)
    {
        //register main user
        _authorizationHelper.RegisterUserAsync("admin@gmail.com","Alex","Hlushko","Olegovich",new
Guid("6266ad9b-a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
        //register guest user
        _authorizationHelper.RegisterUserAsync("guest@gmail.com","John","Doe","Ragnarson",new Guid("6266ad9b-
a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
    }

    [Fact(DisplayName = "Success generate invite token with auth user and access")]
    public async Task SuccessGenerateInviteTokenWithAuthUserAndAccess()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await
_projectHelper.CreateProjectAsync("SuccessGenerateInviteTokenWithAuthUserAndAccess");

        //act
        var response = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var result = await response.Content.ReadAsStringAsync();

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
        result.ShouldNotBeNull();
    }

    [Fact(DisplayName = "Success invite with added user and added rule")]
    public async Task SuccessInviteWithAddedUserAndAddedRule()
    {
        //create project
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("SuccessInviteWithAddedUserAndAddedRule");

        //create invite token
        var postResponse = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var token = await postResponse.Content.ReadAsStringAsync();

        //login as new user and join to project
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var debug1 = await _httpClient.PostAsync($"Project/:join/{token}", null);

        //login as admin and add "delete" rule for new user
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&"); await
_authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");

        var operation = await _sen4Context.Operations.FirstOrDefaultAsync(p => p.Controller == "Project" &&
p.Action == "GenerateInviteToken");
        var userId = _userManager.FindByEmailAsync("guest@gmail.com").Result.Id;
        await _projectHelper.SetRuleAsync(userId, project.Id, operation.Id);

        //login as new user and check delete rule
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var response = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var result = await response.Content.ReadAsStringAsync();

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
        result.ShouldNotBeNull();
    }

    [Fact(DisplayName = "Fail generate invite token with auth user which is not in project")]
    public async Task FailGenerateInviteTokenWithAuthUserWithIsNotInProject()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await
_projectHelper.CreateProjectAsync("FailGenerateInviteTokenWithAuthUserWithIsNotInProject");
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");

        //act
        var response = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
```

```
null);

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Forbidden);
    }

    [Fact(DisplayName = "Fail generate invite token with auth user in project but without rule")]
    public async Task FailGenerateInviteTokenWithAuthUserInProjectButWithoutRule()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await
_projectHelper.CreateProjectAsync("FailGenerateInviteTokenWithAuthUserInProjectButWithoutRule");
        var response = await _httpClient.PostAsJsonAsync("Project/:generateInviteToken",
project.Id.ToString());
        var token = await response.Content.ReadAsStringAsync();
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        await _httpClient.PostAsJsonAsync("Project/:join", token);

        //act
        var result = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);

        //assert
        result.StatusCode.ShouldBe(HttpStatusCode.Forbidden);
    }

    [Fact(DisplayName = "Fail generate invite token with non auth user")]
    public async Task FailGenerateInviteTokenWithNonAuthUser()
    {
        //act
        var response = await _httpClient.PostAsync($"Project/:generateInviteToken/{Guid.NewGuid().ToString()}",
null);

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.Unauthorized);
    }
}
public class Join: TestBase
{
    public Join(IntegrationTestWebAppFactory integrationTestWebAppFactory): base(integrationTestWebAppFactory)
    {
        //register main user
        _authorizationHelper.RegisterUserAsync("admin@gmail.com","Alex","Hlushko","Olegovich",new
Guid("6266ad9b-a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
        //register guest user
        _authorizationHelper.RegisterUserAsync("guest@gmail.com","John","Doe","Ragnarson",new Guid("6266ad9b-
a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
    }


    [Fact(DisplayName = "Success join to project with valid token and auth user")]
    public async Task SuccessJoinToProjectWithValidTokenAndAuthUser()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var project = await _projectHelper.CreateProjectAsync("SuccessJoinToProjectWithValidTokenAndAuthUser");
        var postResponse = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var token = await postResponse.Content.ReadAsStringAsync();

        //act
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var response = await _httpClient.PostAsync($"Project/:join/{token}", null);

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
    }

    [Fact(DisplayName = "Fail join with invalid token")]
    public async Task FailJoinWithInvalidProjectIdAndInvalidToken()
    {
        //arrange
        var token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyf
Q.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c";

        //act
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var response = await _httpClient.PostAsync($"Project/:join/{token}", null);

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.BadRequest);
    }

    [Fact(DisplayName = "Fail join to project with already exists user")]
    public async Task FailJoinToProjectWithAlreadyExistsUser()
    {
        //assert
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
```

```
        var project = await _projectHelper.CreateProjectAsync("FailJoinToProjectWithAlreadyExistsUser");
        var response = await _httpClient.PostAsync($"Project/:generateInviteToken/{project.Id.ToString()}",
null);
        var token = await response.Content.ReadAsStringAsync();

        //act
        var firstUser = await _httpClient.PostAsync($"Project/:join/{token}", null);
        await _authorizationHelper.LoginUserAsync("guest@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        await _httpClient.PostAsync($"Project/:join/{token}", null);
        var secondUser = await _httpClient.PostAsync($"Project/:join/{token}", null);

        //assert
        firstUser.StatusCode.ShouldBe(HttpStatusCode.BadRequest);
        secondUser.StatusCode.ShouldBe(HttpStatusCode.BadRequest);
    }
}
public class List: TestBase
{
    public List(IntegrationTestWebAppFactory integrationTestWebAppFactory):base(integrationTestWebAppFactory)
    {
        //register main user
        _authorizationHelper.RegisterUserAsync("admin@gmail.com","Alex","Hlushko","Olegovich",new
Guid("6266ad9b-a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
        //register guest user
        _authorizationHelper.RegisterUserAsync("guest@gmail.com","John","Doe","Ragnarson",new Guid("6266ad9b-
a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
    }

    [Fact(DisplayName = "Success get list with empty project list request")]
    public async Task EmptyProjectListRequest_SuccessGetList()
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var projectListRequest = new ProjectListRequest();

        var dictionary = new Dictionary<string, StringValues>();
        dictionary.Add("PageSize", new StringValues(projectListRequest.PageSize.ToString()));
        dictionary.Add("PageNumber", new StringValues(projectListRequest.PageNumber.ToString()));

        var requestUrl = QueryHelpers.AddQueryString("Project/List", dictionary);

        //act
        var response = await _httpClient.GetAsync(requestUrl);
        var userList = await response.Content.ReadFromJsonAsync<List<ProjectReadDTO>>();

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.OK);
        userList.ShouldNotBeNull();
    }

    [Theory(DisplayName = "Fail get list with invalid page size and page or page number")]
    [InlineData(-5,1)]
    [InlineData(10,-1)]
    public async Task ProjectListRequestWithInvalidPageSizeAndNumber_Fail(int pageSize, int pageNumber)
    {
        //arrange
        await _authorizationHelper.LoginUserAsync("admin@gmail.com", "StrongPassword_Kj8_Dn3456_ty5&");
        var dictionary = new Dictionary<string, StringValues>();
        dictionary.Add("PageSize", new StringValues(pageSize.ToString()));
        dictionary.Add("PageNumber", new StringValues(pageNumber.ToString()));

        var requestUrl = QueryHelpers.AddQueryString("Project/List", dictionary);

        //act
        var response = await _httpClient.GetAsync(requestUrl);

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.BadRequest);
    }
}
public class Post: TestBase
{
    public Post(IntegrationTestWebAppFactory integrationTestWebAppFactory): base(integrationTestWebAppFactory)
    {
        _authorizationHelper.RegisterUserAsync("admin@gmail.com","Alex","Hlushko","Olegovich",new
Guid("6266ad9b-a32b-452e-b34f-32a0cc3b1d2b"),"StrongPassword_Kj8_Dn3456_ty5&");
        var response = _authorizationHelper.LoginUserAsync("admin@gmail.com",
"StrongPassword_Kj8_Dn3456_ty5&").Result;
    }

    [Fact(DisplayName = "Success post with valid body and valid request id")]
    public async Task SuccessPostWithValidBodyAndValidRequestId()
    {
        //arrange
        var body = new ProjectWriteDTO()
        {
            Name = "Test valid body success post project",
            CreatedDateTime = DateTime.UtcNow,
            Description = null,
        };
```

```
    _httpClient.DefaultRequestHeaders.Add("requestId", Guid.NewGuid().ToString());

    //act
    var response = await _httpClient.PostAsJsonAsync("project", body);
    var result = await response.Content.ReadFromJsonAsync<ProjectReadDTO>();

    //assert
    response.StatusCode.ShouldBe(HttpStatusCode.Created);
    result.ShouldNotBeNull();
}

[Fact(DisplayName = "Fail post with repeated body and request id")]
public async Task FailPostWithRepeatedBodyAndRequestId()
{
    //arrange
    var body = new ProjectWriteDTO()
    {
        Name = "Test fail post with repeated body and requestId",
        CreatedDateTime = DateTime.UtcNow,
        Description = null,
    };
    _httpClient.DefaultRequestHeaders.Add("requestId", Guid.NewGuid().ToString());
    await _httpClient.PostAsJsonAsync("project", body);

    //act
    var response = await _httpClient.PostAsJsonAsync("project", body);

    //assert
    response.StatusCode.ShouldBe(HttpStatusCode.Conflict);
}

[Fact(DisplayName = "Success post with repeated request id and new body")]
public async Task SuccessPostWithRepeatedRequestIdAndNewBody()
{
    //arrange
    _httpClient.DefaultRequestHeaders.Add("requestId", Guid.NewGuid().ToString());
    await _httpClient.PostAsJsonAsync("project", new ProjectWriteDTO()
    {
        Name = "First body name",
        CreatedDateTime = DateTime.UtcNow,
        Description = null,
    });

    //act
    var response = await _httpClient.PostAsJsonAsync("project", new ProjectWriteDTO()
    {
        Name = "Second body name",
        CreatedDateTime = DateTime.UtcNow,
        Description = null,
    });
    var result = await response.Content.ReadFromJsonAsync<ProjectReadDTO>();

    //assert
    response.StatusCode.ShouldBe(HttpStatusCode.Created);
    result.ShouldNotBeNull();
}

[Fact(DisplayName = "Success post with repeated body and new request id")]
public async Task SuccessPostWithRepeatedBodyAndNewRequestId()
{
    //arrange
    var body = new ProjectWriteDTO()
    {
        Name = "SuccessPostWithRepeatedBodyAndNewRequestId",
        CreatedDateTime = DateTime.UtcNow,
        Description = null,
    };
    _httpClient.DefaultRequestHeaders.Add("requestId", Guid.NewGuid().ToString());
    await _httpClient.PostAsJsonAsync("project", body);

    _httpClient.DefaultRequestHeaders.Remove("requestId");
    _httpClient.DefaultRequestHeaders.Add("requestId", Guid.NewGuid().ToString());

    //act
    var response = await _httpClient.PostAsJsonAsync("project", body);
    var result = await response.Content.ReadFromJsonAsync<ProjectReadDTO>();

    //assert
    response.StatusCode.ShouldBe(HttpStatusCode.Created);
    result.ShouldNotBeNull();
}

[Theory(DisplayName = "Fail post with invalid request id")]
[InlineData("")]
[InlineData(null)]
public async Task FailPostWithInvalidRequestId(string? requestId)
{
    //arrange
    var body = new ProjectWriteDTO()
    {
```

```
            Name = "FailPostWithInvalidRequestId",
            CreatedDateTime = DateTime.UtcNow,
            Description = null,
        };

        _httpClient.DefaultRequestHeaders.Add("requestId", requestId);

        //act
        var response = await _httpClient.PostAsJsonAsync("project", body);

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.BadRequest);
    }

    [Fact(DisplayName = "Fail post with invalid data")]
    public async Task InvalidData_FailPost()
    {
        //act
        _httpClient.DefaultRequestHeaders.Add("requestId", Guid.NewGuid().ToString());
        var response = await _httpClient.PostAsJsonAsync("project",  new ProjectWriteDTO()
        {
            Name = " ",
            CreatedDateTime = DateTime.UtcNow,
            Description = null,
        });

        //assert
        response.StatusCode.ShouldBe(HttpStatusCode.BadRequest);
    }
}
```

Додаток В. Код клієнтської частини

```
import { fetchBaseQuery } from "@reduxjs/toolkit/query";
import { createApi } from "@reduxjs/toolkit/query/react";
import {isRejectedWithValue} from "@reduxjs/toolkit";
import {setColor, setMessage, setOpen} from "../features/notificationSlice";

export const apiUrl = "http://localhost:8080";

const baseQuery = fetchBaseQuery({
  baseUrl: apiUrl,
  credentials: "same-origin",
  prepareHeaders: (headers, { getState }) => {
    headers.set("Access-Control-Allow-Origin", "*");
    const token = JSON.parse(localStorage.getItem("accessToken"));

    if (token) {
      headers.set("authorization", `Bearer ${token}`);
    }
    return headers;
  },
});

const baseQueryWithReauth = async (args, api, extraOptions) => {
  let result = await baseQuery(args, api, extraOptions);

  if (result.error && result.error.status === 401) {
    // try to get a new token
    const refreshResult = await baseQuery(
      {
        url: "User/:refresh",
        method: "POST",
        body: {
          accessToken: JSON.parse(localStorage.getItem("accessToken")),
          refreshToken: JSON.parse(localStorage.getItem("refreshToken")),
        },
      },
      api,
      extraOptions
    );

    if (refreshResult) {
      // store the new token
      localStorage.setItem(
        "accessToken",
        JSON.stringify(refreshResult.error.data)
      );
      // retry the initial query
      result = await baseQuery(args, api, extraOptions);
    } else {
      window.location.href = "/signin";
    }
  }
  return result;
};

export const applicationApi = createApi({
  reducerPath: "applicationApi",
  baseQuery: baseQueryWithReauth,
  endpoints: (builder) => ({}),
});
import { applicationApi } from "./applicationApi";
```

```
const authApiSlice = applicationApi.injectEndpoints({
    endpoints: (builder) => ({
        register: builder.mutation({
            query: (body) => ({
                url: "Authentication/:register",
                method: "POST",
                body: body,
            }),
        }),
        login: builder.mutation({
            query: (body) => ({
                url: "Authentication/:login",
                method: "POST",
                body: body,
            }),
        }),
    })
});

export const {
    useLoginMutation,
    useRegisterMutation,
} = authApiSlice;


import {apiUrl, applicationApi} from "./applicationApi";

const getFileNameFromResponse = (response) => {
    console.log("response.headers", response.headers.get("content-
disposition"));

    const contentDisposition = response.headers.get("content-disposition");
    if (!contentDisposition) return "downloaded_file";

    // 1. Проверяем `filename*=UTF-8''...` (приоритетный вариант)
    const utf8FilenameMatch = contentDisposition.match(/filename\*=UTF-
8''(.+?)(?:;|$)/);
    if (utf8FilenameMatch) {
        return decodeURIComponent(utf8FilenameMatch[1]); // Декодируем имя
    }

    // 2. Если `filename*=UTF-8''` нет, используем `filename="..."` или
`filename=...`
    const filenameMatch =
contentDisposition.match(/filename="?(.+?)"?(?:;|$)/);
    if (filenameMatch) {
        return filenameMatch[1];
    }

    return "downloaded_file.pdf";
};
const fileApiSlice = applicationApi.injectEndpoints({
    endpoints: (builder) => ({
        filePost: builder.mutation({
            query: (body) => ({
                url: "File",
                method: "POST",
                body: body,
            }),
            invalidatesTags: ["Files"],
        }),
        fileList: builder.query({
            query: ({projectId, taskId}) => {
```

```
                let url = new URL(apiUrl+'/File/List');

                if(projectId){
                    url.searchParams.set('projectId', projectId);
                }
                if(taskId){
                    url.searchParams.set('taskId', taskId);
                }
                return { url: url.toString() };
            },
            providesTags: ["Files"],
        }),
        fileDelete: builder.mutation({
            query: ({objectName, bucketName}) => ({
                url: `File/${objectName }/${bucketName}`,
                method: "DELETE",
            }),
            invalidatesTags: ["Files"],
        })
    }),
});
export const {
    useFilePostMutation,
    useFileListQuery,
    useFileDeleteMutation,
} = fileApiSlice;


import { applicationApi } from "./applicationApi";

export const operationApiSlice = applicationApi.injectEndpoints({
  endpoints: (builder) => ({
    list: builder.query({
      query: () => `Operation/List`,
    }),
  }),
});

export const { useListQuery } = operationApiSlice;


import { applicationApi } from "./applicationApi";

const priorityApiSLice = applicationApi.injectEndpoints({
    endpoints: (builder)=> ({
        priorityList: builder.query({
            query: () => 'Priority/List'
        })
    })
})

export const {usePriorityListQuery} = priorityApiSLice;


import {apiUrl, applicationApi} from "./applicationApi";

const projectApiSlice = applicationApi.injectEndpoints({
  endpoints: (builder) => ({
    projectList: builder.query({
      query: ({name, showDeleted,pageSize, pageNumber}) => {
        let url = new URL(apiUrl+'/Project/List');
        if(name){
```

```
          url.searchParams.set('Name', name);
        }
        if(showDeleted){
          url.searchParams.set('ShowDeleted', showDeleted);
        }
        if(pageSize){
          url.searchParams.set('PageSize', pageSize);
        }
        if(pageNumber){
          url.searchParams.set('PageNumber', pageNumber);
        }
        return { url: url.toString() };
      },
      providesTags: ["Projects"],
    }),
    projectPost: builder.mutation({
      query: ({ body, idempotentKey }) => ({
        url: "Project",
        method: "POST",
        body: body,
        headers: { requestId: idempotentKey },
      }),
      invalidatesTags: ["Projects"],
    }),
    generateInviteToken: builder.mutation({
      query: (projectId) => ({
        url: `Project/:generateInviteToken/${projectId}`,
        method: "POST",
        responseHandler: (response) => response.text(),
      }),
    }),
    joinToProject: builder.mutation({
      query: (token) => ({
        url: `Project/:join/${token}`,
        method: "POST",
      }),
    }),
    deleteProject: builder.mutation({
      query: (id) => ({
        url: `http://localhost:8080/Project/${id}`,
        method: "DELETE",
      }),
      invalidatesTags: ["Projects"],
    })
  }),
});

export const {
  useProjectListQuery,
  useProjectPostMutation,
  useGenerateInviteTokenMutation,
  useJoinToProjectMutation,
    useDeleteProjectMutation,
} = projectApiSlice;


export const FileTable = ({isLoading,
data,downloadHandle,deleteHandle,projectId}) => {
    return(
       <>{
               isLoading? (
                   <Box
                       sx={{
```

```
                            display: 'flex',
                            justifyContent: 'center',
                            alignItems: 'center',
                            height: '100%',
                        }}>
                            <CircularProgress />
                    </Box>
                ):( data && data.length > 0 ? (
                            <Sheet
                                classNamе="OrderTableContainer"
                                variant="outlined"
                                sx={{
                                    display: { xs: 'none', sm: 'initial' },
                                    width: '100%',
                                    borderRadius: 'sm',
                                    flexShrink: 1,
                                    overflow: 'auto',
                                    minHeight: 0,
                                }}>
                                <Table stripe="2n" hoverRow stickyHeader>
                                    {/*Заголовки*/}
                                    <thead>
                                    <tr>
                                        <th>Файл</th>
                                        <th>Останній раз змінено</th>
                                        <th>Розмір</th>
                                        <th style={{ width: '15%' }}></th>
                                    </tr>
                                    </thead>

                                    {/* Тело таблицы   */}
                                    <tbody>
                                    {data&& data.map((item,index) => (
                                        <tr key={`${item.id}-${index}`}>
                                            <td>
                                                <Typography
startDecorator={<InsertDriveFileIcon/>}>{item.key}</Typography>
                                            </td>
                                            <td>

<Typography>{item.lastModifiedDateTime}</Typography>
                                            </td>
                                            <td>

<Typography>{item.size}</Typography>
                                            </td>
                                            <td>
                                                <IconButton
onClick={()=>downloadHandle(item.key)}>
                                                    <DownloadIcon/>
                                                </IconButton>

                                                <IconButton onClick={() =>
deleteHandle(item.key, projectId)}>
                                                    <DeleteIcon/>
                                                </IconButton>

                                                <IconButton>
                                                    <HelpIcon/>
                                                </IconButton>
                                            </td>
                                        </tr>
                                    ))}
```

```
                                </tbody>
                            </Table>
                        </Sheet>
                    ):(
                        <NoDataFound/>
                    )
                )
            }
        </>
    )
}


export const FileTableFilterPanel = ({setOpen}) => {
    return (
        <Box
            className="SearchAndFilters-tabletUp"
            sx={{
                borderRadius: 'sm',
                py: 2,
                display: { xs: 'none', sm: 'flex' },
                flexWrap: 'wrap',
                gap: 1.5,
                '& > *': {
                    minWidth: { xs: '120px', md: '160px' },
                },
            }}>

            {/*search*/}
            <FormControl sx={{ flex: 1 }} size="sm">
                <FormLabel>Пошук завдання</FormLabel>
                <Input
                    size="sm"
                    placeholder="Search"
                    startDecorator={<SearchIcon />}/>
            </FormControl>

            {/* status*/}
            <FormControl size="sm">
                <FormLabel>Статус</FormLabel>
                <Select
                    size="sm"
                    placeholder="Filter by status"
                    slotProps={{ button: { sx: { whiteSpace: 'nowrap' } } }}>
                </Select>
            </FormControl>

            <Button onClick={() => setOpen(true)} endDecorator={
                <UploadFileIcon />
            }>
            </Button>
        </Box>
    )
}
export default function Sidebar() {
    return (
        <Sheet
            className="Sidebar"
            sx={{
                position: { xs: 'fixed', md: 'sticky' },
                transform: {
                    xs: 'translateX(calc(100% * (var(--SideNavigation-
slideIn, 0) - 1)))',
```

```
                    md: 'none',
                },
                transition: 'transform 0.4s, width 0.4s',
                // zIndex: 10000,
                height: '100dvh',
                width: 'var(--Sidebar-width)',
                top: 0,
                p: 2,
                flexShrink: 0,
                display: 'flex',
                flexDirection: 'column',
                gap: 2,
                borderRight: '1px solid',
                borderColor: 'divider',
            }}
        >

            <GlobalStyles
                styles={(theme) => ({
                    ':root': {
                        '--Sidebar-width': '220px',
                        [theme.breakpoints.up('lg')]: {
                            '--Sidebar-width': '240px',
                        },
                    },
                })}
            />


            <Logo/>
            <Divider/>


            <ProjectSelectMenu/>


            <SidebarMenuList/>
            <Divider />


            <AccountSidebarCard/>
        </Sheet>
    )
}
export default function useSignalR(){
    const dispatch = useDispatch();
    const [connection,setConnection] = useState(null);

    useEffect(()=>{
        const hubConnection = new HubConnectionBuilder()
            .withUrl("http://localhost:8080/notification",{
                skipNegotiation: true,  // skipNegotiation as we specify
WebSockets
                transport: HttpTransportType.WebSockets, // force WebSocket
transport
                accessTokenFactory: () =>
`${JSON.parse(localStorage.getItem("accessToken"))}`
            })
            .build();

        setConnection(hubConnection);
    },[]);
```

```
    useEffect(() => {
        if (connection) {
            connection.start()
                .then(result => {
                    connection.on('notification', message => {
                        console.log(message);
                        dispatch(setOpen(true));
                        dispatch(setMessage(message));
                        dispatch(setColor('primary'));
                    });
                })
                .catch(e => console.log('Connection failed: ', e));
        }
    },[connection]);

    return [connection];
}
```