Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
ФІС
(назва факультету)
Кафедра Програмної Інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

магістр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: Розробка веб-застосунку для управління процесом створення програмного забезпечення з використанням Angular для ПП «Магніс»

Виконав: студент (ка) 6 курсу, групи СПм-62
спеціальності (напряму підготовки)
121 Програмна інженерія
(шифр і назва спеціальності (напряму підготовки))

(підпис) (прізвище та ініціали)

Керівник Петрик М. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Магістерська робота на тему «Розробка веб-застосунку для управління процесом створення програмного забезпечення з використанням Angular для ПП «Магніс»» Жилавського Богдана Петровича. – Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–62 // Тернопіль, 2019.

С. – ___, рис. – ___, табл. – ___, додат. – ___, бібліогр. –___.

Метою дипломної роботи є розробка інформаційної системи, для управління процесом створення програмного забезпечення. Розроблений продукт надасть можливість планувати та контролювати весь процес життєвого циклу, дозволить швидко реагувати на зміни до вимог, забезпечить комунікацію між усіма учасниками робочого процесу.

Методи та програмні засоби, використані при виконанні розробки системи: мова програмування JavaScript та Typescript, фреймворк Angular, платформа NodeJs.

Результатом роботи є готовий програмний продукт, який задовольняє вимогам які описані в постановці задачі.

Ключові слова: МЕТОДОЛОГІЯ, ЖИТТЄВИЙ ЦИКЛ, ПРОЦЕС, УПРАВЛІННЯ, ПРОГРАМНА СИСТЕМА, ГНУЧКІ МОДЕЛІ ЖИТТЄВОГО ЦИКЛУ.

ABSTRACT

Master thesis « Development of web-application to manage the software creation process using Angular for enterprise «Magnis»» by student Zhylavskyi Bohdan Petrovych. – Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Software engineering department, group SPm-62 // Ternopil, 2019.

Pages. – ___, pictures. – ___, tables. – ___ , add. – __, bibl.ref. – __.

The purpose of the thesis is to develop an information system to manage the process of creating software. The developed product will allow to plan and control the whole life cycle process, will allow to react quickly to changes to requirements, will provide communication between all participants of the work process.

Methods and software used in system development: JavaScript and Typescript programming language, Angular framework, NodeJs platformThe result of the work is a finished software product that meets the requirements described in the statement of the problem.

Keywords: METHODOLOGY, LIFE CYCLE, PROCESS, MANAGEMENT, SOFTWARE, AGILE LIFE CYCLE MODELS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

ПС – програмна система, комплекс програмного забезпечення.

ПК – персональний комп’ютер, робоча машина для розробки та виконання програм.

UML – (Unified Modeling Language) уніфікована мова графічного представлення та об’єктного моделювання в області розробки програмного забезпечення парадигми об’єктно-орієнтованого програмування.

Алгоритм – набір інструкцій, які описують порядок виконання дій, що дозволяють досягти результату за скінченну кількість кроків.

ОП – охорона праці.

SCRUM - підхід управління проектами для гнучкої розробки програмного забезпечення. Scrum чітко робить акцент на якісному контролі процесу розробки.

CLI – Command Line Interface (інтерфейс командного рядка).

HTTP – HyperText Transfer Protocol.

DOM – Document Object Model.

ЗМІСТ

ВСТУП .....

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ .....

1.1 Аналіз вимог до предметної області .....

1.1.1 Аналіз предметної області ....

1.1.2 Постановка задачі ....

1.1.3 Визначення акторів та опис ключових варіантів використання системи ....

1.2 Проектування програмної системи .....

1.2.1 Вибір процесу розробки ....

1.2.2 Виявлення основних сутностей ....

1.2.3 Побудова схеми бази даних.....

1.2.4 Моделювання архітектури системи.....

1.3 Конструювання програмної системи .....

1.3.1 Опис використаних технологій ....

1.3.2 Реалізація основних класів та методів ..... ….

2 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОГРАМНОЇ СИСТЕМИ .....

2.1 Тестування програмної системи ....

2.2 Розгортання програмної системи ....

2.3 Ілюстровані сценарії використання ....

3 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ .....

3.1 Загальний підхід до визначення економічної ефективності розробки .....

3.2 Визначення ключових витрат .....

3.3 Визначення періоду окупності та собівартості…..

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ .....

4.1 Охорона праці .....

4.2 Забезпечення електробезпеки користувачів ПК.....

ВИСНОВКИ.....

ПЕРЕЛІК ПОСИЛАНЬ……..

ДОДАТКИ……..

ВСТУП

Перехід від ручних засобів розроблення ПЗ до промислового виробництва програм потребував розвитку теоретичних основ розроблення ПЗ. Постійна необхідність внесення змін у програми як спричинена помилками, так і розвитком вимог до них, є принциповою властивістю програмного забезпечення. Діяльність, пов’язана з рішенням широкого ряду завдань для постійного розвитку, отримала назву супроводу програмного забезпечення. Якщо зусилля, спрямовані на модернізацію ПЗ, перевищують вигоду від його використання, говорять про моральне старіння програм.

Оскільки розроблення та супровід ПЗ фактично є проектною діяльністю, частина ключових понять управління проектів знайшла широке застосування у програмній інженерії. Таким є і поняття життєвого циклу проекту (Project Lifecycle Management, PLM), що в програмній інженерії трансформувалось у поняття життєвого циклу програмного забезпечення.

Життєвий цикл програмного забезпечення - період часу, що починається з моменту прийняття рішення про необхідність створення програмного продукту і закінчується в момент його повного вилучення з експлуатації. Цей цикл - процес побудови і розвитку ПЗ.

Поняття ЖЦ виникло під впливом потреби у систематизації робіт у процесі розроблення ПЗ. Систематизація була першим етапом на шляху до автоматизації процесу розроблення ПЗ. Наступними кроками переходу до автоматизації процесу розроблення ПЗ були такі: встановлення технологічних маршрутів діяльності розробників ПЗ, визначення можливості їх автоматизації та виявлення ризиків, розроблення інструментів для автоматизації.

Виявлення закономірностей розвитку програмного забезпечення одразу показало нерозвиненість методик конструювання ПЗ та недостатність тестування для визначення якості програмних продуктів. Також на цьому етапі стало зрозуміло, що нечіткість завдання на створення ПЗ викликає більшість проблем розроблення та перевірки програм. У результаті виникли вимоги до

постановки завдання, сформувалися підходи до управління вимогами на етапі аналізу та інструменти зв’язку вимог на етапах аналізу та реалізації.

Діяльність під час розроблення ПЗ, як і будь-що, складається з виконання операцій і проектів. Ті й інші мають багато спільного, наприклад, виконуються людьми та на їх виконання виділяються обмежені ресурси.

Головна відмінність операцій від проектів полягає в тому, що операції виконуються постійно і повторюються, тоді як проект тимчасовий і унікальний. Виходячи з цього, проект визначається як тимчасове зусилля, розпочате для створення унікального продукту чи послуги. «Тимчасове» означає, що кожен проект має точно визначені дати початку та закінчення. Говорячи про унікальність продукту, ми маємо на увазі, що вони мають помітні відмінності від усіх аналогічних продуктів або послуг. Таким чином, розроблення ПЗ відповідає визначенню проекту і для організації цього процесу можна застосовувати методи та інструментарій управління проектами.

У кожного проекту є чітко визначені початок і кінець. Кінець проекту настає разом із досягненням усіх його цілей або коли стає зрозумілим, що ці цілі не будуть або не можуть бути досягнуті. Тимчасовість не означає короткостроковість проекту – розроблення складної програмної системи може тривати кілька років, хоча, як правило проекти мають обмежені часові рамки для створення ПЗ, оскільки сприятлива для них ситуація на ринку складається на обмежений час. Крім того, проектна команда після його закінчення розпадається, а її члени переходять в інші проекти [1].

Для успішного завершення проекту необхідно використовувати всі доступні засоби, які допомагають планувати та контролювати весь процес життєвого циклу, дозволяють швидко реагувати на зміни до вимог, забезпечують комунікацію між усіма учасниками робочого процесу.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

Процес розробки програмного забезпечення (життєвий цикл програмного забезпечення) – це сукупність послідових етапів спрямованих на розробку програмного забезпечення [2]. Ці етапи можуть включати як створення так і модифікацію програмного продукту.

Процес розробки програмного забезпечення включає наступні етапи:

- аналіз та планування – визначення необхідних ресурсів для реалізації системи;
- специфікація програмного забезпечення (інженерія вимог) – визначає основні функціональні можливості програмного забезпечення та їх обмеження;
- проектування та реалізація програмного забезпечення – розробка та реалізація програмних компонентів системи;
- верифікація та валідація програмного забезпечення – процес перевірки програмного забезпечення на наявність помилок або дефекті, а також на відповідність продукту вимогам замовника;
- супровід програмного забезпечення - процес покращення, оптимізації та виправлення дефектів у програмному забезпеченні після його вводу до експлуатації.

Мета будь-якого процесу розробки програмного забезпечення - створити продукт, який буде доставлений вчасно, у межах виділеного бюджету, та з функціоналом, який очікує замовник [3]. Для досягнення поставленою мети, в процесі задіяні фахівці, кожен з яких має свою область відповідальності та виконує певну роль.

Виділяють наступні основні ролі в процесі створення програмного забезпечення:

- замовник – особа (фізична або юридична), яка є ініціатором процесу;
- менеджер проекту – особа, яка займається плануванням та керуванням процесу виконання робіт, розставляє пріоритети, спілкується з замовником тощо;
- розробник – особа, яка займається розробкою компонентів програмної системи;
- тестувальник – особа, яка перевіряє програмний продукт на наявність помилок та відповідність до вимог;
- аналітик – особа, яка займається формуванням вимог до програмного продукту, створює документацію.

Процес розробки програмного забезпечення представляють у вигляді моделей.

Модель життєвого циклу ПЗ – це структура, яка описує взаємозв'язок та послідовність етапів життєвого циклу.

Найпоширенішими моделями є: каскадна, інкрементна та спіральна.

Каскадна модель – класична модель життєвого циклу програмного забезпечення. Її часто називають водопадною, підкреслюючи те, що весь життєвий цикл розглядається як послідовність етапів, при цьому наступний етап розпочинається лише після завершення попереднього (див. рис. 1.1). Результат виконання попереднього етапу є вхідними даними для наступного.

На даний момент дана модель рідко використовується, проте вона стала основою для інших моделей життєвого циклу.

Переваги каскадної моделі:

- простота;
- в один момент часу виконується лише один етап життєвого циклу;
- чітке розмежування кожного з етапів життєвого циклу;
- детальна документація результатів кожного з етапів життєвого циклу;

– простота при класифікації та розставленні пріоритетів завдань;
– легко розрахувати термін виконання проекту;

Недоліки каскадної моделі:

– не підходить для проектів, де часто змінюються вимоги та довготривалих проектів;
– неможливо повернутися до попередніх етапів для доопрацювання;
– важко визначити прогрес розробки;
– неможливо оцінити продукт до завершення усіх етапів життєвого циклу.

Рисунок 1.1 – Каскадна модель життєвого циклу ПЗ

Підстави для використання каскадної моделі:

– чітка документація вимог;
– чіткий перелік інструментів та технології, які будуть використовуватись під час розробки;
– малий або середній розмір проекту.

Ітеративна модель розглядає життєвий цикл ПЗ як послідовність етапів, які виконуються декілька раз, при цьому програмний продукт покращується з кожною ітерацією (див. рис. 1.2).

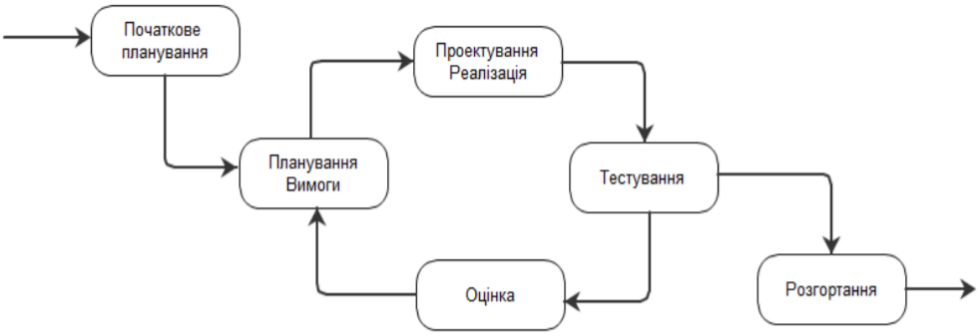

Рисунок 1.2 – Ітеративна модель життєвого циклу

На відміну від каскадної моделі, яка не передбачає повернення до попередніх етапів життєвого циклу, при ітеративній моделі дозволяється коригування результатів кожного з етапів. При використанні даної моделі, наявність усіх вимог до початку проекту не обов’язкова.

Використання ітеративної моделі найчастіше спостерігається при розробці складних систем, при цьому можлива розробка версіями, причинами якої можуть бути:

– відсутність у замовника бюджету для розробки проекту у повному обсязі;
– відсутність необхідних ресурсів, які б дозволили розробнику реалізувати проект;
– впровадження цілого проекту може викликати неприйняття у користувачів.

Переваги ітеративної моделі:

– замовник має можливість оцінити результат кожного з етапів, залишити відгук про виконану роботу та коригувати процес створення;
– так як дана модель передбачає можливість розробки версіями, користувач буде мати можливість ознайомитись з продуктом набагато раніше ніж при використанні каскадної моделі;
– гнучкість процесу дозволяє легко підлаштовуватись до змін у вимогах;

– скорочується сукупність документації порівняно з каскадною моделлю.

Недоліки ітеративної моделі:

– оскільки, замовник має можливість постійно вносити зміни до вимог, то при додаванні нових компонентів існує ймовірність зламати вже існуючий функціонал. Необхідно виділяти ресурси для рефакторингу;
– потребує більше ресурсів ніж каскадна модель;
– потребує постійного менеджменту.

Підстави для використання ітеративної моделі:

– відсутність чітких вимог до початку проекту;
– існує ймовірність частих змін до вимог програмного продукту;
– середній або великий розмір проекту.

Спіральна модель поєднує в собі риси ітеративної та каскадної моделей. Основним завданням спіральної моделі є аналіз ризиків життєвого циклу ПЗ. Дану модель відображають у вигляді спіралі (див. рис. 1.3).

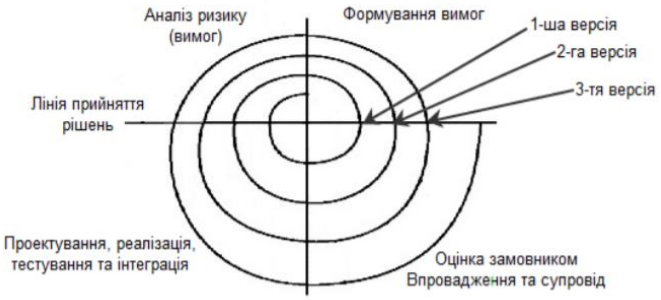

Рисунок 1.3 – Спіральна модель життєвого циклу

При використанні спіральної моделі передбачається чотири етапи для кожної версії програмного продуку (кожного витка):

– формування вимог;
– аналіз ризиків;
– конструювання;
– оцінка результатів та формування вимог для нової версії продукту.

Переваги спіральної моделі:

– детальний аналіз ризиків;
– підходить для критично важливих проектів;
– сувора верифікація та контроль документації;
– забезпечує зворотній зв’язок між користувачами та розробниками, завдяки чому забезпечується висока якість продукту.

Недоліки спіральної моделі:

– складна структура життєвого циклу;
– висока вартість.

Підстави для використання спіральної моделі:

– відсутність повної картини кінцевого продукту;
– необхідно швидко вийти на ринок.

V-подібна модель розширяє каскадну модель за рахунок виконання тестування після кожного етапу розробки (див рис. 1.4). Після виконання кожного процесу, відбувається валідація та верифікація результатів. Якщо результат виконання етапу задовільняє вимогам, відбувається перехід до наступного етапу життєвого циклу.

Рисунок 1.4 – V-подібна модель життєвого циклу

Переваги V-подібної моделі:

– валідація та верифікація результутів кожного етапу життєвого циклу.

Недоліки V-подібної моделі:

– успадковує недоліки каскадної моделі.

Підстави використання V-подібної моделі:

– при розробці програмних продуктів, в яких коректність роботи критично важлива.

Гнучкі моделі – це сімейство методологій розробки ПЗ, основою яких є ітеративна розробка (див. рис. 1.5). Ітерації зазвичай тривають від одного до чотирьох тижнів. Використання ітеративної розробки дозволяє зменшити ризики, які виникають в процесі розробки. Основні ідеї гнучкої моделі:

– учасники процесу розробки важливіші за будь-які процеси та інструменти;
– наявність робочого ПЗ важливіше наявності повної документації;
– комунікація з замовником важливіша ніж контракти;
– здатність відповідати на зміни вимог важливіші ніж дотримання плану.

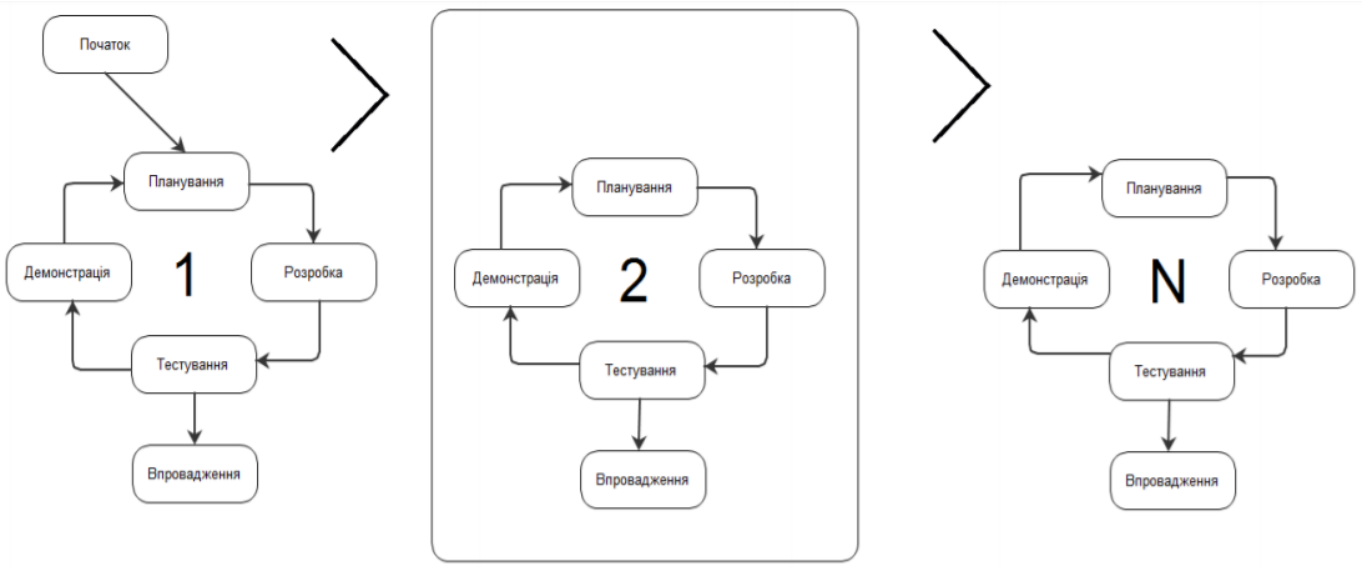

Рисунок 1.5 – Гнучка модель життєвого циклу ПЗ

Переваги гнучкої моделі:

– передбачає можливість внесення змін до вимог;
– швидкий вихід програмного продукту на ринок;
– легко побачити прогрес розробки;
– постійна комунікація з замовником дозволяє покращувати продукт після кожної ітерації.

Недоліки гнучкої моделі:

– оскільки, дана модель передбачає створення моделі раніше ніж документації, то можливі проблеми пов’язані з відсутністю документації;
– можлива втрата контролю над процесом розробки, через відсутність чіткого бачення готового продукту в замовника.

Однією з найпопулярніших методологій гнучкої моделі життєвого цикла ПЗ є методологія Scrum. Дана метолологія зазвичай використовується в проетах, в яких необхідно кожні два-чотири тижні доставляти новий функціонал замовнику [3].

Основні поняття в методології Scrum:

– спринт (Sprint) – одна ітерація процесу розробки, яка триває від 1 до 4 тижнів. При зменшенні тривалості спринта, збільшується гнучкість процесу розробки, користувачі мають можливість швидше досліджувати нові можливості продукту;
– користувацька історія (User Story) – короткий опис функціоналу, необхідний користувачу; (Наприклад, користувач має мати можливість ділитися своїми фотографіями);
– завдання (Task) – складова будь-якої користувацької історії. Детальний опис функціоналу, який необхідно реалізувати; (Наприклад, реалізувати графічний інтерфейс відповідно до дизайну);
– story point – відносна одиниця вимірювання складності користувацьких історій та завдань.

Основними елементами методології Scrum є: ролі, події, артифакти та правила.

Методологія передбачає наступні ролі:

– власник продукту (Product Owner) – основний зацікавлений учасник процесу розробки, основним завданням якого є донесення вимог замовника до команди розробників. Власник продукту також займається розставленням приоритетів завдань.
– керівник (ScrumMaster) – учасник процесу розробки, основним завданням якого є контроль за процесом, контроль за дотриманням усіх принципів та правил Scrum.
– команда розробки – відповідальна за виконання технічних завдань, а саме: аналіз, дизайн, програмування, тестування тощо.

Події в методології Scrum:

– плануванні спринта – захід, в якому беруть участь всі члени команди, в процесі якого відбувається розставлення пріоритетів та оцінка завдань, які необхідно виконати в майбутньому спринті;

– щоденний мітинг – захід, під час якого, команда має можливість обговорити завдання та труднощі пов’язані з ними;
– огляд спринта – демонстрація замовнику функціоналу, який був розроблений в рамках попереднього спрінта;
– ретроспектива спринта – захід, під час якого відбувається обговорення та аналіз успіхів та невдач команди під час минулого спринта;

Основними артефактами методології Scrum є: project backlog, sprint backlog, increment.

Project backlog – список функціоналу (завдань), який необхідно реалізувати для завершення готового продукту.

Sprint backlog – список функціоналу, який необхідно реалізувати протягом спринта.

Increment – готовий продукт в кінці спрінта.

Переваги використання методології Scrum:

– висока продуктивність команди розробників;
– висока якість створеного продукту;
– можливість швидкого виходу продукту на ринок.

Оскільки, найпопулярнішою на даний час моделлю життєвого циклу ПЗ є гнучка модель, а саме методологія Scrum, то було прийняте рішення розробляти інформаційну систему, яка б дозволила керувати процесом створення програмного забезпечення використовуючи ідеї та принципи, які стали основою даної методології.

1.1.2 Постановка задачі

В процесі виконання магістерської роботи необхідно реалізувати веб-застосунок призначений для управління процесом створення програмного забезпечення. Розроблений програмний продукт повинен надати функціонал, який дозволить керувати процесом розробки використовуючи ідеї, покладені в основну гнучких методологій розробки ПЗ, а саме методологію Scrum.

Використання програмного продукту дозволить автоматизувати процеси життєвого циклу ПЗ, мінімізувати ризики, які виникають при розробці, надасть можливість аналізу процесів.

Вимоги до функціоналу програмного продукту:

- необхідно реалізувати реєстрація та авторизацію;
- система повинна передбачати наявність таких ролей користувачів: керівник проекту (Project Manager) та член команди розробки (Team). Роль, яку виконує користувач, описує який функціонал йому доступний в межах проекту;
- користувач має мати можливість редагувати власний профіль, а саме: змінювати нікнейм, аватар, інформацію про себе тощо;
- користувач має мати можливість створити проект. При створенні проекту, користувач вказує детальну інформацію про проект, а саме: назва проекту, завдання проекту, тривалість проекту тощо. Передбачити можливість редагування інформації про проект.
- користувач, який виконує роль менеджера проекту (Project Manager) має мати можливість створювати завдання (Task), формувати команду розробки, формувати спринти (вказувати часові рамки, формувати списки завдань, які необхідно виконати під час спринта);
- при створенні завдання, користувач має мати можливість вказати детальну інформацію, а саме: назву, опис, пріоритет, виконавця тощо;
- передбачити можливість вказувати статус завдання (Напр. В процесі);
- надати можливість відслідковувати прогрес виконання завдання;
- передбачити можливість збору звітності та аналітики проекту.

Інтерфейс продукту повинен бути зрозумілим, простим в користуванні. Програмний продукт повинен бути доступним на всіх платформах.

1.1.3 Визначення акторів та ключових варіантів використання системи

В результаті аналізу предметної області і постановки задачі можна виділити лише одного основного актора – користувач. Будь-якому користувачеві доступні наступні варіанти використання:

- реєстрація та авторизація;
- налаштування профілю.

Кожен користувач, має певну роль. Кожна роль описує які обов’язки та права має користувач в межах проекту. Відповідно до конкретної ролі користувачеві необхідно надати функціонал, за допомогою якого він зможе виконувати покладені на нього обов’язки.

Система передбачає наявність таких ролей:

- менеджер проекту (Project Manager);
- член команди розробки (Team).

Менеджер проекту – це користувач, який створив проект. Кожен користувач, який створює проект, автоматично стає менеджером даного проекту.

Нижче наведено варіанти використання для користувача з роллю “Менеджер проекту”:

- редагування інформації про проект;
- створення та редагування завдань, необхідних для завершення проекту;
- перегляд завдань проекту;
- додавання коментарів до завдань проекту;
- призначення виконавця завдання;
- створення/редагування спринтів проекту;
- перегляд спринтів проекту;
- активація спринтів;
- перегляд та формування команди розробки проекту.

Діаграма прицидентів користувача з роллю “Менеджер проекту” зображена на рисунку 1.6.

Рисунок 1.6 - Діаграма прицидентів користувача з роллю “Менеджер проекту”

Користувач з роллю “Член команди розробки” має обмежені права в межах проекту порівняно з користувачем з роллю “Менеджер проекту”.

Варіанти використання користувача з роллю “Член команди розробки”:

– зміна статусу завдання;
– перегляд завдань проекту;
– перегляд інформації про проект;
– перегляд інформації про учасників розробки проекту;
– створення коментарів до завдання проекту.

Діаграма прицидентів користувача з роллю “Член команди розробки” зображена на рисунку 1.7.

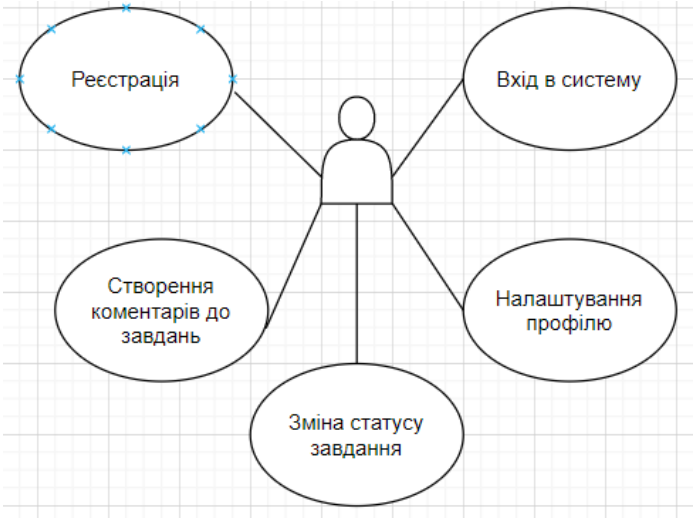

Рисунок 1.7 - Діаграма прицидентів користувача з роллю “ Член команди розробки”

1.2 Проектування програмної системи

1.2.1 Вибір процесу розробки

Методологією розробки обрано каскадну модель. Зважаючи на всі недоліки даної моделі, її використання виправдано, тому що:

- на початку проекту відомі всі вимоги до функціоналу, який необхідно реалізувати;
- оскільки, всі вимоги відомі, то одразу можна розробити програмну архітектуру;
- малий розмір проекту дозволяє уникнути концептуальних помилок;
- виконавцем кожного з етапів життєвого циклу є одна особа, тому немає необхідності використовувати бізнес-правила, діаграми тощо.

1.2.2 Вивлення основних сутностей

В результаті аналізу предметної області та постановки задачі, можна виділити наступні сутності:

- User – сутність, яка містить інформацію про користувачів;

– Project – сутність, що описує проект;
– Task – сутність, яка описує завдання (Task);
– Sprint – сутність, що містить інформацію про спринт;
– TaskComment – сутність, яка містить інформацію про коментарі до завдань.

Сутність User містить наступні атрибути:

– Id (ідентифікатор);
– Username (ім'я користувача);
– UserImg (зображення-аватар користувача);
– Email (електронна пошта користувача);
– Created (час реєстрації користувача).

Сутність Project містить наступні атрибути:

– Id (ідентифікатор);
– Name (назва проекту);
– Description (опис проекту).

Сутність Task містить наступні атрибути:

– Id (ідентифікатор);
– ProjectId (ідентифікатор проекту);
– Name (назва завдання);
– Description (опис завдання);
– Priority (пріоритет);
– Status (статус завдання).

Сутність Sprint містить наступні атрибути:

– Id (ідентифікатор);
– ProjectId (ідентифікатор проекту);
– Name (назва спринта);
– Description (опис спринта).

Сутність TaskComment містить наступні атрибути:

– Id (ідентифікатор);
– CreatorId (ідентифікатор користувача, що створив коментар);
– TaskId (ідентифікатор завдання);
– Text (текст коментаря).

Сутність User пов’язана із сутністю Project зв’язком один-до-багатьох, адже один користувач може створити декілька проектів.

Сутність Project пов’язана із сутністю Sprint зв’язком один-до-багатьох, адже проект може включати декілька спринтів, необхідних для завершення проекту.

Сутність Sprint пов’язана із сутністю Task зв’язком один-до-багатьох, адже спринт може включати декілька завдань, необхідних для завершення спринта.

Сутність Task пов’язана із сутністю TaskComment зв’язком один-до-багатьох, адже до одного завдання можуть адресуватися декілька коментарів.

1.2.3 Побудова схеми бази даних

Так як перелік основних сутностей та атрибутів визначено, можна перейти до побудови фізичної моделі бази даних. Для реалізації програмного продукту, було прийняте рішення використовувати систему управління базами даних (СУБД) MYSQL Server. Гнучкість даної СУБД забезпечується підтримкою великої кількості типів таблиць, швидкістю роботи тощо. MYSQL Server найкраще підходить для використання в малих та середніх застосунках.

Структура таблиці user містить наступні поля:
– id, числовий тип, первинний ключ;
– username, текстовий тип, розмір поля – 20;
– user_img, текстовий тип, розмір поля - 45;
– email, текстовий тип, розмір поля – 30;
– create_time, часова мітка;

Таблиця project містить наступні поля:

– id, числовий тип, первинний ключ;
– name, текстовий тип, розмір поля – 20;
– description, текстовий тип, розмір поля - 140;
– creatorId, числовий тип, зовнішній ключ до таблиці user.

Таблиця task містить наступні поля:

– id, числовий тип, первинний ключ;
– projectId, числовий тип, зовнішній ключ до таблиці project;
– name, текстовий тип, розмір поля – 20;
– description, текстовий тип, розмір поля - 140;
– priority, числовий тип;
– creatorId, числовий тип, зовнішній ключ до таблиці user;
– created, часова мітка.

Таблиця taskComment містить наступні поля:

– id, числовий тип, первинний ключ;
– creatorId, числовий тип, зовнішній ключ до таблиці user;
– taskId, числовий тип, зовнішній ключ до таблиці task;
– text, текстовий тип, розмір поля - 140;
– created, часова мітка.

Оскільки, один користувач може створити декілька проектів, то зв’язок між таблицями user та project встановлюється як один-до-багатьох (див. рис. 1.8). При цьому, в таблиці project використовується зовншній ключ creatorId, який вказує на автора проекту в таблиці user.

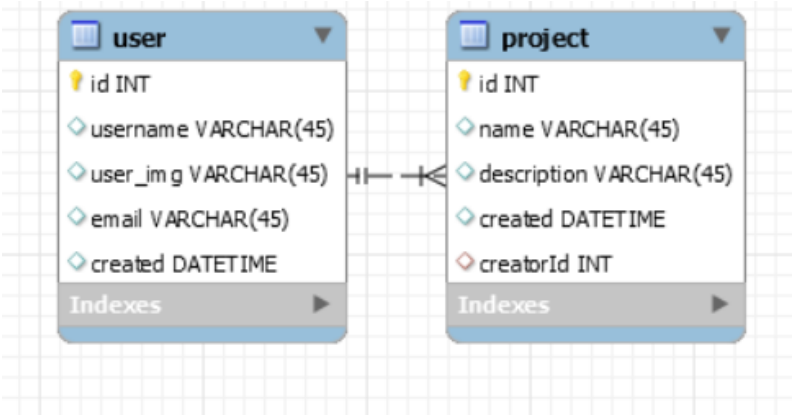

Рисунок 1.8 – Схема зв’язку таблиць user та project

Кожен проект включає декілька завдань, тому зв’язок між таблицями project та task – один-до-багатьох. При цьому, один користувач може створити декілька завдань, тому зв’язок між таблицями user та task також буде один-до-багатьох. Схема зв’язку вищеописаних таблиць наведена на рисунку 1.9.

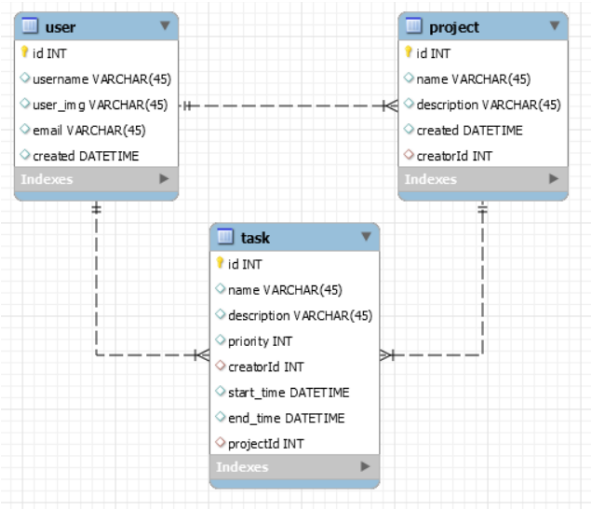

Рисунок 1.9 – Схема зв’язку таблиць user, project та task

Так як один користувач може бути учасником розробки декількох проектів і над одним проектом можуть працювати декілька користувачів, та в даному випадку зв’язок між сутностями User та Project буде багато-до-багатьох (див.рис. 1.10). Для реалізації даного виду зв’язку використовується додаткова таблиця. В даному випадку це таблиця ProjectMembers.

Таблиця ProjectMembers містить наступні атрибути:

– Id, числовий тип, первинний ключ;

– projectId, числовий тип, зовнішній ключ до таблиці project;

– userId, числовий тип, зовнішній ключ до таблиці user.

Таблиця taskComment містить інформацію про коментарі до завдань. Оскільки, до одного завдання може бути декілька коментарів, то зв’язок між

таблицею task і taskComment буде один-до-багатьох. Так як один користувач може бути автором декількох коментарів, то зв’язок між таблицями user та taskComment також буде один-до-багатьох. Схема зв’язку даних таблиць наведена на рисунку 1.11.

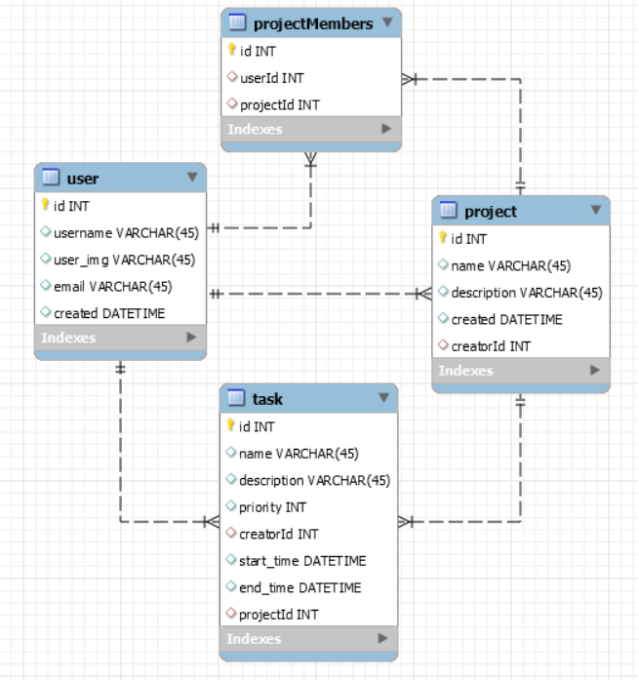

Рисунок 1.10 – Схема зв’язку таблиць user, project та projectMembers

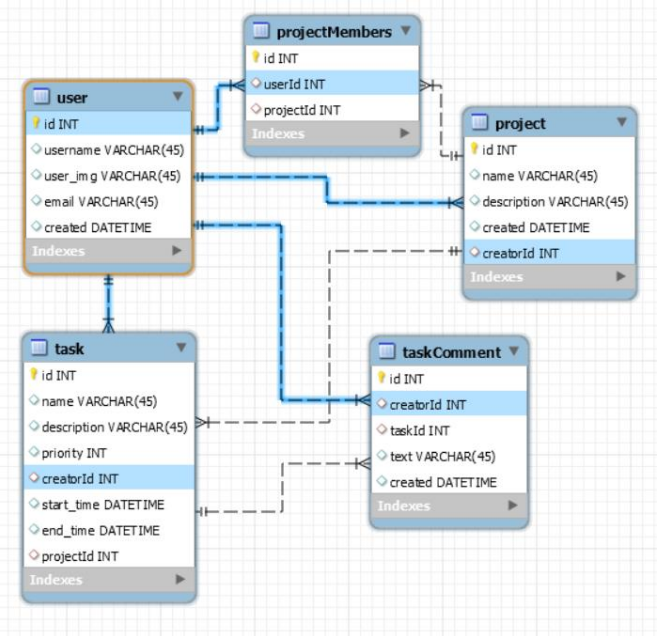

Рисунок 1.11– Схема зв’язку таблиць user, task та taskComment

Проект може містити лише один активний спринт. Для цього було прийнято рішення використовувати окрему таблицю activeSprints. Дана таблиця містить інформацію про активні спринти в проектах.

Загальна схема зв’язку зображена на рисунку 1.12.

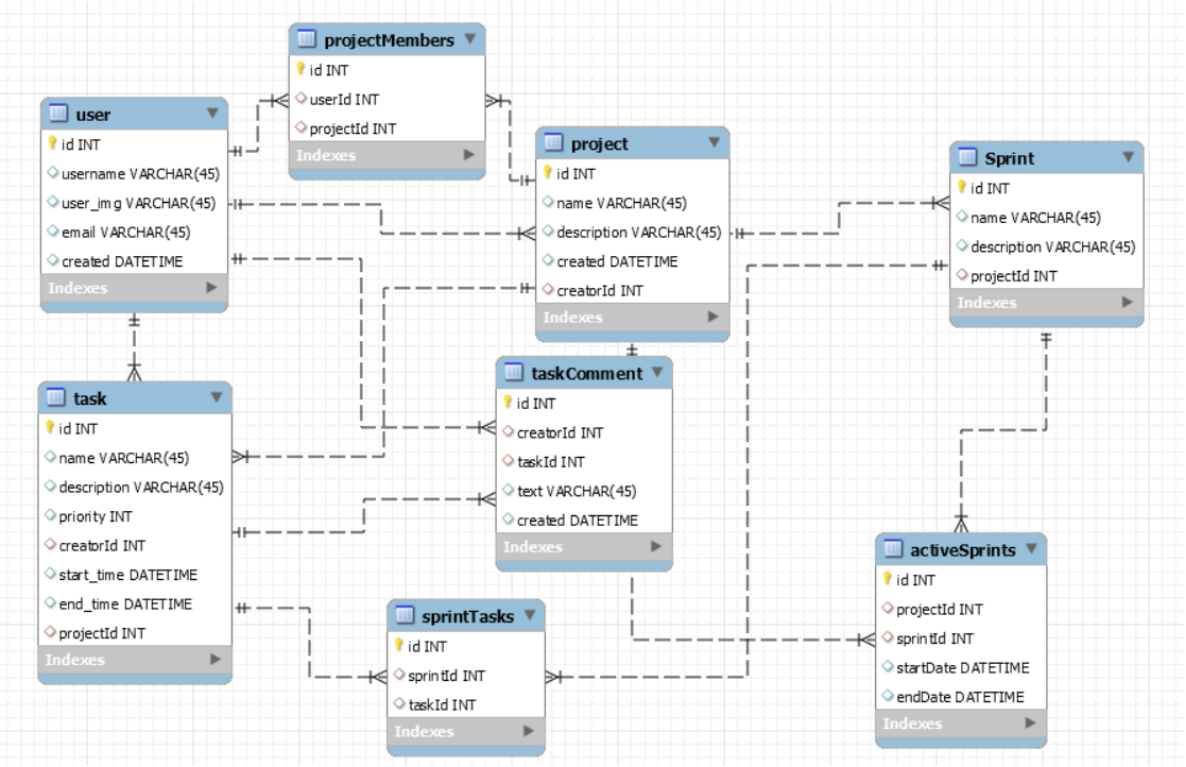

Рисунок 1.12 – Загальна схема зв’язку

При розробці програмного забезпечення з використанням реляційної бази даних важливим етапом проектування є нормалізація схеми бази даних.

Нормалізація запобігає дублюванню інформації, її надмірності та можливості отримання помилкових результатів при вибірці або модифікації даних [4]. Характеристикою такої нормалізації є нормальна форма – сукупність вимог, яким має задовільняти відношення.

Надлишковість даних в БД призводить до збільшення об'єму пам'яті та уповільнення роботи БД. Причиною надлишковості даних є дублювання даних. Розрізняють два види дублювання даних: незбиткове та збиткове. Повне усунення надлишковості є недоцільним, так як це унеможливить підтримку БД

як єдиного цілого. Варто мінімізувати надлишковість, при цьому залишити необхідне дублювання.

Відношення знаходиться у першій нормальній формі, якщо задовільняються такі вимоги:

1. Значення усіх атрибутів повинні бути атомарними;
2. Відсутність атрибутів, які повторяються.

Перша з умов виконується для будь-якого відношення автоматично, оскільки воно виходить із властивостей відношень. Для безпомилкового доступу до бази даних раціонально використовувати унікальні ключові поля (ідентифікатори), які дозволять міняти вміст певного кортежу без можливих помилок зміни даних у іншому.

Відношення знаходиться в другій нормальній формі, якщо воно знаходиться в першій нормальній формі і кожний не функціональний атрибут повністю залежить від ключа. Тобто знаходження певного значення по атрибуту, який не являється ключовим – неможливо.

Дані сутності будуть пов’язані ключовими полями і будуть містити тільки основні ключі.

Відношення знаходиться в третій нормальній формі, якщо воно знаходиться в другій нормальній формі і кожний неключовий атрибут нетранзитивно залежить від первинного ключа.

Для доведення того, що розроблена база даних знаходиться у третій нормальній формі, достатньо розглянути таблиці user та task. В таблиці task, кожний атрибут не залежить від інших.

Нормалізація даної бази даних до четвертої і п’ятої нормальної форми являється нерентабельною. Адже подана сутність вже є спроектованою якісно, у ній відсутня надлишковість інформації і всі зв’язки-ідентифікатори є унікальними, складові ключі відсутні. Продовження нормалізації приведе до збільшення зв’язків між сутностями і більшої складності управління вмістом бази даних.

1.2.4 Моделювання архітектури системи

Для реалізації поставленого завдання загальною архітектурою обрано клієнт-серверну архітектуру.

Клієнт-серверна архітектура – це архітектура, при якій відповідальність розподіляється між двома основними компонентами: сервером, який надає певні послуги (сервіси), та клієнтом, який використовує сервіси надані сервером. [5] Взаємодія між цими компонентами забезпечується мережею. Принципова схема архітектури зображена на рисунку 1.13.

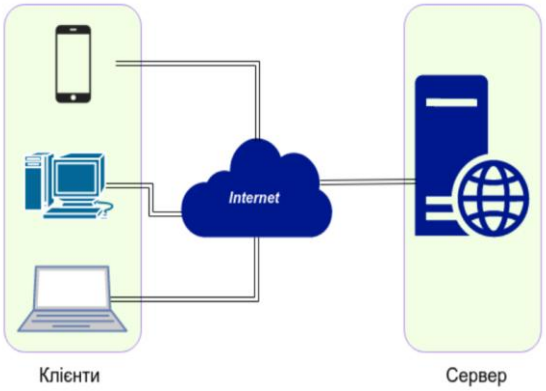

Рисунок 1.13 – Клієнт серверна архітектура

Сервер – це програмна система, яка надає послуги (сервіси) іншим програмам і обслуговує запити клієнтів для отримання певних ресурсів або виконання певних операцій.

Клієнт – це програмна система, яка використовує ресурси або послуги (сервіси), які надає сервер.

Клієнт-серверна архітектура є багаторівневою архітектурою, яка передбачає розділення функціональності на логічні рівні. [5] Основні рівні архітектури:

– рівень представлення даних – це інтерфейс користувача (або програмний інтерфейс), основним завданням якого є представлення даних клієнту та опрацювання команд клієнта;

– рівень бізнес-логіки – описує основну логіку системи, займається обробкою інформації;
– рівень керування даними – описує логіку зберігання та доступу до даних;

Перевагами багаторівневої архітектури є:

– можливість незалежної модифікації кожного з рівнів;
– розділення відповідальності між рівнями;
– можливість багаторазового використання кожного з рівнів.

Дворівнева клієнт-серверна архітектура передбачає наявність двох рівні: клієнтський та серверний. Залежно від того, як розподіляється який з рівнів містить функціонал бізнес логіки, розрізняють: модель тонкого клієнта та модель товстого клієнта. [5]

При використанні моделі тонкого клієнта, весь функціонал, який описує рівень бізнес логіки зосереджений на сервері. Клієнт тільки реалізує функціонал з рівня представлення даних.

Переваги моделі тонкого клієнта:

– відсутність дублювання коду бізнес логіки при розробці клієнтів під різні платформи;
– оскільки, всі обчислення здійснюються на сервері, то вимоги до машин клієнтів значно зменшуються;
– всі дані зберігаються на сервері – це забезпечує централізований захист інформації, спрощується організація контролю доступу клієнтів до інформації з обмеженим доступом.

Недоліки моделі тонкого клієнта:

– перевантаження сервера – при великій кількості запитів, серверу може не вистачити ресурсів для їхнього опрацювання;
– відмова критично важливого сервера, може спричинити відмову цілої системи.

При використанні моделі товстого клієнта, функціонал рівня бізнес логіки зосереджений на клієнті, при цьому сервер тільки реалізує функціонал з рівня керування даними.

Переваги моделі товстого клієнта:

– швидкодія, оскільки взаємодія з сервером використовується лише для збереження або модифікації даних;
– простота;
– можливість роботи при втраті з'єднання з сервером;
– низьке навантаження на сервер.

Недоліки:

– оскільки, бізнес логіка розміщена на стороні клієнта, то при її модифікації необхідно повністю оновити клієнтське програмне забезпечення;
– складність масштабування;
– необхідна наявність потужних ресурсів, які б дозволили забезпечити коректне функціонування системи;
– залежність від платформи;
– дублювання коду бізнес-логіки при розробці клієнтів під різні платформи.

Трирівнева клієнт-серверна архітектура передбачає наявність трьох рівнів: рівень представлення даних, рівень бізнес логіки та рівень керування даними. Схема роботи даної архітектури показана на рисунку 1.14.

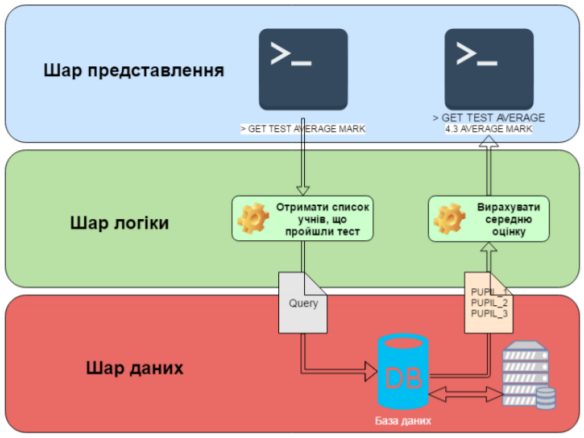

Рисунок 1.14 – Трьохшарова клієнт-серверна архітектура

На відміну від двохшарової архітектури, трьохшарова архітектура забезпечує кращу масштабованість (горизонтальне масштабування шару бізнес логіки), простіше конфігурування системи (оскільки, шари не залажать один від одного). Проте, розробка систем з використання трьохшарової архітектури значно складніша ніж з використанням двохшарової і потребує додаткових витрат на адміністрування такої системи.

В результаті аналізу варіантів використання клієнт-серверної архітектури, було прийнято рішення використовувати трьохшарову клієнт-серверну архітектуру.

Для реалізації клієнтської частини використовується архітектурний шаблон MVVM (Model-View-ViewModel) (див. рис. 1.15). Використання даного шаблону дозволяє забезпечити розділення візуальної частини (інтерфейсу користувача) від логіки системи, як результат – можливість незалежно змінювати бізнес-логіку та логіку відображення [6].

MVVM складається з таких компонентів: модель (Model), презентаційна модель (ViewModel) та представлення (View).

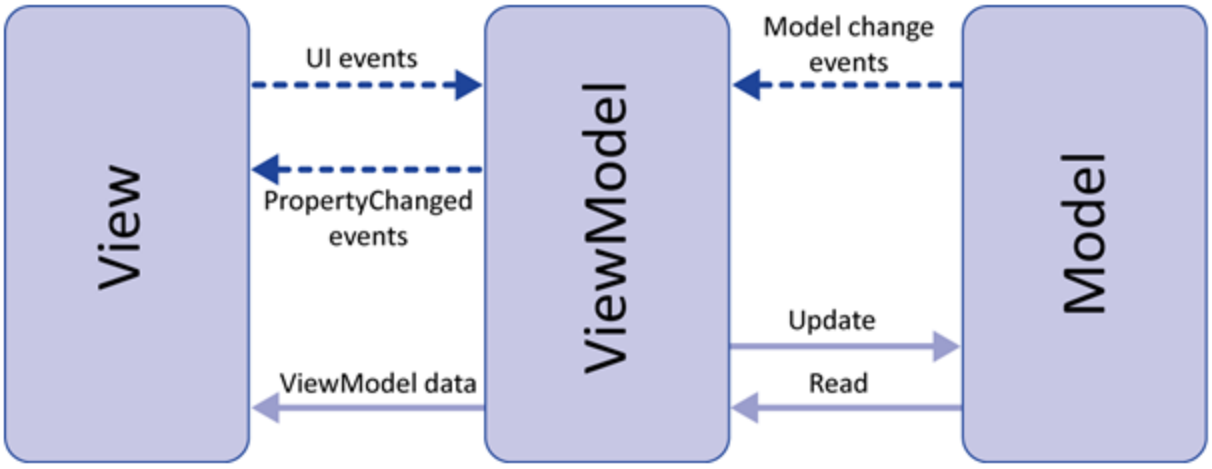

Рисунок 1.15 – MVVM

Модель (Model) описує дані, які використовуються в системі та логіку пов’язану з даними (наприклад валідація). Згідно з шаблоном MVVM, модель

не має містити логіку, яка пов’язана з відображенням даних та взаємодією з графічними елементами інтерфейсу. Для сповіщення інших частин системи про зміни в моделі, використовується шаблон проектування спостерігач (Observer).

Представлення (View) описує інтерфейс, з яким взаємодіє користувач. Згідно з шаблоном MVVM, представлення (View) повинно містити тільки логіку відображення. Основним завдання представлення (View) є відображення відформатованих даних користувачеві. Весь інший функціонал представлення (View) делегує презентаційній моделі (ViewModel).

Презентаційна модель (ViewModel) зв’язує модель (Model) та представлення (View) за допомогою механізму двохсторонньої прив’язки даних (Data Binding). Презентаційна модель (ViewModel) містить логіку отримання даних із моделі (Model), а також логіку оновлення даних в моделі (Model).

Переваги архітектурного шаблону MVVM:

- розширюваність;
- простота в підтримці;
- простота тестування.

1.3 Конструювання програмної системи

1.3.1 Огляд використаних технологій

Для реалізації клієнтської частини використовується веб-застосунок (односторінковий додаток).

Односторінковий додаток (Single Page Application) – це веб-застосунок, основною особливістю якого є наявність лише одного HTML документа, який виконує роль контейнера для інших веб-сторінок. В даному типі веб-застосунків, інтерактивність графічного інтерфейсу досягається за допомогою динамічного завантаження HTML, CSS, JavaScript засобами AJAX, а також динамічною генерацією елементів графічного інтерфейсу.

Будь-який веб-застосунок розробляється з використанням таких основних технологій:

– HTML;
– CSS;
– JavaScript.

HTML (Hypertext Markup Language) – мова розмітки тексту, що використовується для опису структури веб-сторінок та веб-додатків.

CSS (Cascading Style Sheets) – каскадні таблиці стилів, мова, яка надає можливість описувати зовнішній вигляд елементів веб-сторінки, а саме: шрифт, верстку, кольори тощо.

JavaScript – динамічна мова програмування, разом із HTML і CSS, JavaScript одна з основних технологій всесвітньої павутини.

Як мульти-парадигмна мова, JavaScript підтримує фунціональний та імперативний (об'єктно орієнтовний та прототипний) стилі програмування, а також відповідні архітектурні властивості: динамічна типізація, автоматичне управління пам'яттю, прототипне наслідування, функції вищого порядку.

JavaScript використовується для:

– динамічної генерації HTML документа;
– валідації введених даних і відправлення інформації на сервер;
– забезпечення інтерактивного графічного інтерфейсу (обробка подій, виведення повідомлень для користувача тощо).

Переваги використання мови JavaScript:

– простота вивчення та реалізації;
– швидкодія;
– широке розмаїття бібліотек та фреймворків;
– кросплатформність.

Одним з основних недоліків JavaScript є відсутність статичної типізації. Особливо цей недолік проявляється при написанні великих проектів з великої кількістю розробників. Для покращення розробки та підтримки таких проектів,

компанія Microsoft восени 2012 року представила світу мову програмування TypeScript.

TypeScript – відкрита мова програмування, синтаксична надбудова над мовою JavaScript. TypeScript створенна для розробки великих проектів з великою кількістю розробників. Код на мові TypeScript компілюється в JavaScript код. Так як TypeScript є надбудовою над JavaScript, то будь-який валідний код на JavaScript є валідним кодом на TypeScript. Причиною створення TypeScript стали недоліки JavaScript, через які як і компанії Microsoft так і в клієнтів компанії виникли проблеми при розробці та підтримці проектів з великою кодовою базою. В результаті цього утворився попит на інструмент, який би спростив розробку.

Переваги TypeScript над JavaScript:

- простота в підтримці великих проектів;
- збільшує продуктивність роботи розробників;
- підтримка інтерфейсів, абстрактних класів та закритих методів і полей класу;
- підтримка модулів;
- дозволяє виявляти помилки ще на етапі компіляції;
- можливість використовувати функціонал, який ще не підтримується браузерами (за допомогою поліфілів);
- спрощується рефакторинг коду.

В якості основних мов програмування використовуються JavaScript та TypeScript.

Для спрощення розробки веб-застосунків використовуються фреймворки.

Фреймворк – це сукупність програмних рішень, які полегшують розробку складних систем, дозволяють структурувати програмний код та задають стиль написання коду [7].

Для спрощення розробки веб-застосунку було принято рішення використовувати фреймворк Angular.

Angular – фреймворк з відкритим кодом, призначений для розробки веб-додатків (SPA). Розробляється компанією Google з використанням мови програмування TypeScript. Angular являється нащадком фреймворка AngularJS, який відзначився неоптимальністю та складністю написання застосунків. Розробники AngularJS врахували всі надоліки та труднощі, які виникають з фреймворком і розробили новий, концептуально простіший та зрозуміліший фреймворк Angular.

Фреймворк Angular реалізує:

- модульність;
- маршрутизацію;
- роботу з сервером;
- роботу з шаблонами та формами.

Основні компоненти Angular:

- модулі (Modules);
- компоненти (Components);
- шаблони (Templates);
- сервіси (Services);
- роутер (Router);
- пайпи (Pipes);
- дерективи (Directives);
- інжектор залежностей (Dependency Injector).

Компонент в Angular – клас, який містить дані та описує логіку відображення цих даних у шаблоні. Кожен компонент містить шаблон, який описує представлення компонента. Шаблон має доступ до даних компонента завдяки механізму зв’язування даних. Відповідно до архітектурного шаблону MVVM, компонент виконує функції моделі представлення (View Model), а шаблон – функції представлення (View). Приклад компонента наведено на рисунку 1.16.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
    `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

Рисунок 1.16 – Приклад компонента Angular

Кожен компонент може містити інші компоненти. При цьому формується ієрархія компонентів (див. рис. 1.17). Компонент, який містить інші компоненти називається батьківським (Parent) відносно них. Компоненти, які є складовою іншого компонента, називаються дочірніми (Child) відносно нього. Батьківські і дочірні компоненти можуть обмінюватись даними.

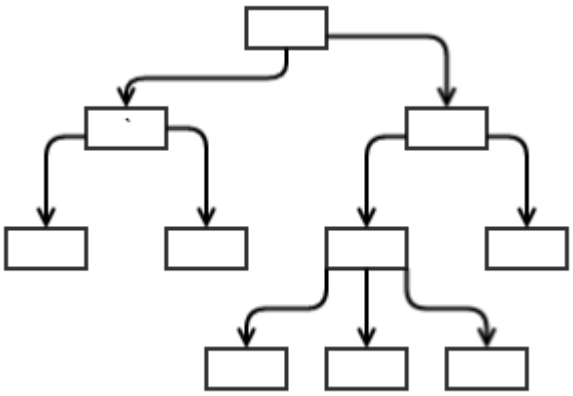

Рисунок 1.17 – Ієрархія компонентів

Таким чином весь графічний інтерфейс користувача можна представити у вигляді ієрархії компонентів.

Деректива в Angular – те ж саме що і компонент, але не містить шаблона. Основне завдання дерективи – змінити поведінку елемента, до якого вона застосовується. Приклад дерективи наведено на рисуку 1.18.

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
    constructor(el: ElementRef) {
       el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

Рисунок 1.18 – Приклад дерективи в Angular

Наведена вище деректива задає елементу жовтий колір фону.

Сервіс в Angular - клас, який містить логіку зберігання та опрацювання даних. Відповідно до архітектурного шаблону MVVM, сервіси виконують функції моделі (Model). Сервіси напряму не взаємодіють з представленням (View). Приклад сервіса наведено на рисунку 1.19.

```
import { Injectable } from '@angular/core';
import { HEROES }     from './mock-heroes';
import { Logger }     from '../logger.service';

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor(private logger: Logger) {  }

  getHeroes() {
    this.logger.log('Getting heroes ...');
    return HEROES;
  }
}
```

Рисунок 1.19 – Приклад сервіса Angular

Пайп в Angular – клас, основним завданням якого є трансформація даних. Використовується в шаблоні. Приклад пайпа наведено на рисунку 1.20.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent?: number): number {
    return Math.pow(value, isNaN(exponent) ? 1 : exponent);
  }
}
```

Рисунок 1.20 – Приклад пайпа в Angular

Модуль в Angular – клас, який описує основну структурну одиницю Angular застосунку. Модуль об'єднує компоненти, директиви, пайпи та сервіси, що пов'язані певною логікою. Крім цього, модулі описують свої залежності. Приклад модуля наведено на рисунку 1.21.

```
// imports
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

// @NgModule decorator with its metadata
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Рисунок 1.21– Приклад модуля в Angular

Роутер – маршрутизатор, основним завданням якого є відображення конкретних компонентів в залежності від URL адреси [8].

Для розробки серверної частини обрано платформу Node Js, яка задовільняє усім висунутим вимогам.

NodeJs – платформа, середовище виконання, яке дозволяє виконувати програмний код на мові програмування JavaScript поза браузером. Платформа надає програмний інтерфейс для роботи з мережею та файловою системою. Завдяки асинхронній моделі виконання, платформа NodeJs прекрасно показує себе при розробці серверних застосунків з великою кількістю запитів.

Переваги використання Node Js:

- для розробки клієнтської і серверної частини використовується одна мова програмування – JavaScript;
- легкість масштабування;
- наявність великої кількості бібліотек;
- неблокуючий ввід/вивід;
- асинхронна модель розробки.

Вибір СУБД MySQLServer пов'язаний насамперед із наявністю безплатної версії, що дозволило зменшити затрати на розробку системи. Також ця СУБД відзначається високою стійкістю до навантажень, що дозволяє великій кількості клієнтів працювати із базою даних одночасно.

MySQLServer є оптимальним рішенням для малих і середніх застосунків. Основні особливості сервера MySQL:

- простота;
- необмежена кількість одночасних користувачів;
- швидкість;
- ефективна система безпеки.

Для управління Angular проекту використовується Angular CLI. Angular CLI – командний застосунок, інструмент, який використовується для створення, розробки, масштабування та підтримки застосунків розроблених засобами Angular. Щоб встановити Angular CLI необхідно використати пакетний менеджер npm, виконавши в командній стрічці команду, яка наведена на рисунку 1.22.

Рисунок 1.22 – Команда для встановлення Angular CLI

Після встановлення Angular CLI доступний наступний перелік команд, які використовуються для управління проектом, що розробляється засобами Angular:

- ng new – команда для створення нового проекту Angular;
- ng serve – команда для запуску проекту на локальній машині;
- ng build – команда для компіляції проекту. Результат компіляції можна завантажити на сервер;
- ng update – команда для оновлення залежностей проекту;
- ng g component – команда для створення компоненту;
- ng g module – команда для створення модуля;
- ng g service – команда для створення сервісу.

1.3.2 Реалізація основних класів та методів

Одним з основних завдань, які необхідно реалізувати як в клієнтській так і серверній частині є авторизація. Одним із засобів, які використовуються для реалізації авторизації є HTTP інтерцептор. HTTP інтерцептор дозволяє перехоплювати HTTP-запити і перед їх відправкою на сервер вносити певні зміни, наприклад додати заголовки, необхідні для авторизації. Код реалізації HTTP інтерцептора наведено в лістингу.

Лістинг 1.1 – Реалізація HTTP інтерцептора

```
import {Inject, Injectable, Injector} from '@angular/core';
import {
    HttpErrorResponse,
    HttpEvent,
    HttpHandler,
    HttpInterceptor,
    HttpRequest,
```

```
        HttpResponse
    } from "@angular/common/http";
    import {BehaviorSubject, Observable, throwError} from "rxjs";
    import {catchError, filter, switchMap, take, tap} from "rxjs/operators";
    import {Router} from "@angular/router";
    import {ProfileService} from "./profile.service";
    import {AppConfig} from "../app.config";

    @Injectable()
    export class AuthInterceptor implements HttpInterceptor {
        private refreshTokenInProgress = false;
        private refreshTokenSubject: BehaviorSubject<any> = new
BehaviorSubject<any>(null);

        _profileService;

        constructor(private router: Router,
                    @Inject(AppConfig) private _appConfig: AppConfig,
                    private injector: Injector) {
        }

        getProfileService() {
            if(!this._profileService)
                this._profileService = this.injector.get(ProfileService);

            return this._profileService;
        }

        intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
            if(!this._appConfig.http || !this._appConfig.http.users)
                return next.handle(request);

            const profile = this.getProfileService();

            return <any>next.handle(request).pipe(
                catchError(error => {
                    console.log('ERROR', error);
                    if (
                        request.url.includes("refreshToken") ||
                        request.url.includes("login")
                    ) {
                        if (request.url.includes("refreshToken")) {
                            profile.logout();
                        }

                        return throwError(error);
                    }

                    if (error.status !== 401) {
                        return throwError(error);
                    }

                    if (this.refreshTokenInProgress) {
                        return this.refreshTokenSubject
                            .pipe(
                                filter(result => result !== null),
                                take(1),
                                switchMap(() => next.handle(request)),
                            );
                    } else {
                        this.refreshTokenInProgress = true;
```

```
                        this.refreshTokenSubject.next(null);

                        return profile.refreshToken().pipe(
                                switchMap((token: any) => {
                                    this.refreshTokenInProgress = false;
                                    this.refreshTokenSubject.next(token);

                                    return next.handle(request);
                                }),
                                catchError((err: any) => {
                                    this.refreshTokenInProgress = false;

                                    profile.logout();
                                    return throwError(error);
                                }));

                    }
                }))
            }
        }
```

За обробку та отримання даних в Angular відповідають сервіси. Оскільки, взаємодія з сервером базується на принципах REST, за якими інформація, яка міститься на сервері розглядається як абстрактний ресурс, то було прийняте рішення реалізувати базовий клас сервіса (провайдера), який описує інтерфейс для взаємодії з сервером. Реалізація базового сервіса (провайдера) наведена в лістингу 1.2.

Лістинг 1.2 – Реалізація базового сервіса (провайдера)

```
import {forkJoin, Observable, of} from 'rxjs';
import {HttpClient, HttpParams} from '@angular/common/http';
import {Inject, Injectable} from '@angular/core';
import {ExcludeId, Provider} from '../common/provider';
import {CommunicationConfig} from '../../communication.config';
import {IIdObject} from '../../models/id.object';
import {map, tap} from 'rxjs/operators';

@Injectable()
export abstract class HttpProvider<T extends IIdObject> extends
Provider<T> {
    protected _baseUrl: string;

    constructor(@Inject(HttpClient) protected _http: HttpClient,
                @Inject(CommunicationConfig) protected
_communicationConfig: CommunicationConfig) {
        super();
        try {
            this._baseUrl = this._getURL(_communicationConfig) || 'Invalid
proxy configuration';
        } catch (e) {
            console.error('Invalid base url configuration', e);
```

```
            }
        }

        protected abstract _getURL(config: CommunicationConfig): string;

        getItemById(id: number): Observable<T> {
            return this._http.get<T>(this._getRESTURL(id));
        }

        getItems(obj?: any): Observable<T[]> {
            let params = {};
            if (obj) {
                params = new HttpParams({fromObject: obj});
            }

            return this._http.get<T[]>(this._getRESTURL(), {params});
        }

        createItem(item: ExcludeId<T>): Observable<any> {
            return this._http.post(this._getRESTURL(), item)
                .pipe(
                    map(this.combineResponse(item)),
                    tap(this.onCreate)
                );
        }

        updateItem(item: T): Observable<any> {
            return this._http.put<any>(this._getRESTURL(item.id), item)
                .pipe(
                    map(this.combineResponse(item)),
                    tap(this.onUpdate)
                );
        }

        deleteItem(id: number | string): Observable<any> {
            return this._http.delete(this._getRESTURL(id))
                .pipe(
                    tap(() => this.onDelete(id))
                );
        }

        getItemsByIds(ids?: number[]): Observable<T[]> {
            if (!ids || !ids.length) {
                return of([]);
            }

            return forkJoin(ids.map(id => this.getItemById(id)));
        }

        protected _concatUrl(...params: (string | number)[]): string {
            return                                  `${this._baseUrl}`.concat('/',
params.filter(Boolean).map(toString).join('/'));
        }

        protected _getRESTURL(id?) {
            return this._concatUrl(id);
        }

        protected arrayToUrl(...params: (string | number)[]) {
            return params.filter(Boolean).map(toString).join('/');
        }
    }
```

```
function toString(i) {
    return i.toString();
}
```

2 РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Розгортання програмної системи

Для розгортання клієнтської частини розробленої засобами Angular достатньо скомпілювати проект та результат компіляції завантажити на веб-сервер.

Для компіляції проектів Angular використовується інструмент Angular CLI. Для компіляції проекту використовується команда, яка наведена на рисунку

```
ng build --prod
```

Рисунок 2.1– Команда для компіляції проекту Angular

В процесі компіляції проекту відбувається компіляція коду TypeScript, компіляція стилів а також оптимізаційні заходи, а саме:

- AOT (Ahead-of-Time) компіляція – процес, під час якого відбувається прекомпіляція компонентів. На відміну від JIT (Just-in-Time) компіляції в Angular, AOT компіляція відбувається в процесі компіляції проекту, а не у браузерів в момент виконання;
- зв’язування (Bundling) – процес, під час якого весь програмний конкатенується в декілька файлів, завдяки чому браузеру необхідно виконати менше запитів для завантаження необхідного коду;
- мініфікація (Minification) – процес, під час якого з програмного коду видаляються непотрібні дані (табуляція, коментарі тощо);
- обфускація (Obfuscation) – процес, під час якого відбувається перейменування функцій та змінних на більш короткі;
- видалення коду, який не використовується (Dead code elimination).

Наведені вище заходи по оптимізації застосунків з використанням Angular, дозволяють значно зменшити розмір файлів, які необхідно завантажити для роботи застосунку.

Після завершення компіляції, результат компіляції (файли) необхідно завантажити на веб-сервер. Клієнтська частина проекту готова до використання.

Для розгортання серверної частини, необхідно скористуватись платформами, які дозволяють розгортати застосунки, розроблені засобами платформи NodeJs. Процес розгортання для кожної платформи зазначений у відповідній документації.

2.2 Тестування програмної системи

Тестування один з основних етапів життєвого циклу, завданням якого є пошук дефектів системи, валідація та верифікація. Одним з найефективніших методик тестування є модульне тестування (Unit Testing). Відповідно до даної методики, кожен незалежний модуль тестується окремо.

Для тестування Angular проектів використовуються фреймворк для тестування Jasmine та платформа для запуску тестів Karma.

Jasmine – відкритий фреймворк для тестування з використанням мови програмування JavaScript, який надає інструменти для тестування компонентів програмної системи.

Переваги використання фреймворку Jasmine:

- зрозумілість і простота написання тестів;
- підтримує асинхронне тестування;
- для виконання тестів не потрібно використовувати DOM (Document Object Model).

Приклад тесту з використанням Jasmine наведено на рисунку 2.2.

```
describe('Hello world', function() {
  it('says hello', function() {
    expect(helloWorld()).toEqual('Hello world!');
  });
});
```

Рисунок 2.2 – Приклад тесту з використанням фреймворку Jasmine

Форма входу користувача один з найголовніших компонентів графічного інтерфейсу системи, який потребує обов'язкового тестування. Код тестування форми входу наведено в лістингу 2.1.

Лістинг 2.1 – Код тестування форми входу

```
/* tslint:disable:no-unused-variable */
import { TestBed, ComponentFixture, inject, async } from
'@angular/core/testing';
import { LoginComponent, User } from './login.component';
import { Component, DebugElement } from "@angular/core";
import { By } from "@angular/platform-browser";


describe('Component: Login', () => {

    let component: LoginComponent;
    let fixture: ComponentFixture<LoginComponent>;
    let submitEl: DebugElement;
    let loginEl: DebugElement;
    let passwordEl: DebugElement;

    beforeEach(() => {
        TestBed.configureTestingModule({
            declarations: [LoginComponent]
        });

        fixture = TestBed.createComponent(LoginComponent);
        component = fixture.componentInstance;

        submitEl = fixture.debugElement.query(By.css('button'));
        loginEl = fixture.debugElement.query(By.css('input[type=email]'));
        passwordEl = fixture.debugElement.query(By.css('input[type=password]'));
    });

    it('Setting enabled to false disabled the submit button', () => {
        component.enabled = false;
        fixture.detectChanges();
        expect(submitEl.nativeElement.disabled).toBeTruthy();
```

```
    });

    it('Setting enabled to true enables the submit button', () => {
        component.enabled = true;
        fixture.detectChanges();
        expect(submitEl.nativeElement.disabled).toBeFalsy();
    });

    it('Entering email and password emits loggedIn event', () => {
        let user: User;
        loginEl.nativeElement.value = "test@example.com";
        passwordEl.nativeElement.value = "123456";

        component.loggedIn.subscribe((value) => user =
value);submitEl.triggerEventHandler('click', null);
        expect(user.email).toBe("test@example.com");
        expect(user.password).toBe("123456");
    });
});
```

Результат тестування форми входу зображено на рисунку 2.3.

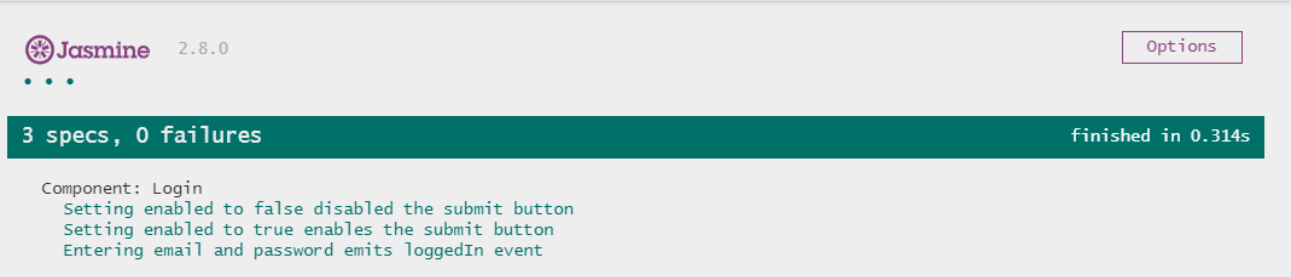

Рисунок 2.3 – Результат тестування форми входу

2.3 Ілюстровані сценарії використання

При використанні даного веб-застосунку, першою веб-сторінкою, яку бачить користувач є веб-сторінка “Вхід” (див. рис. 2.4);

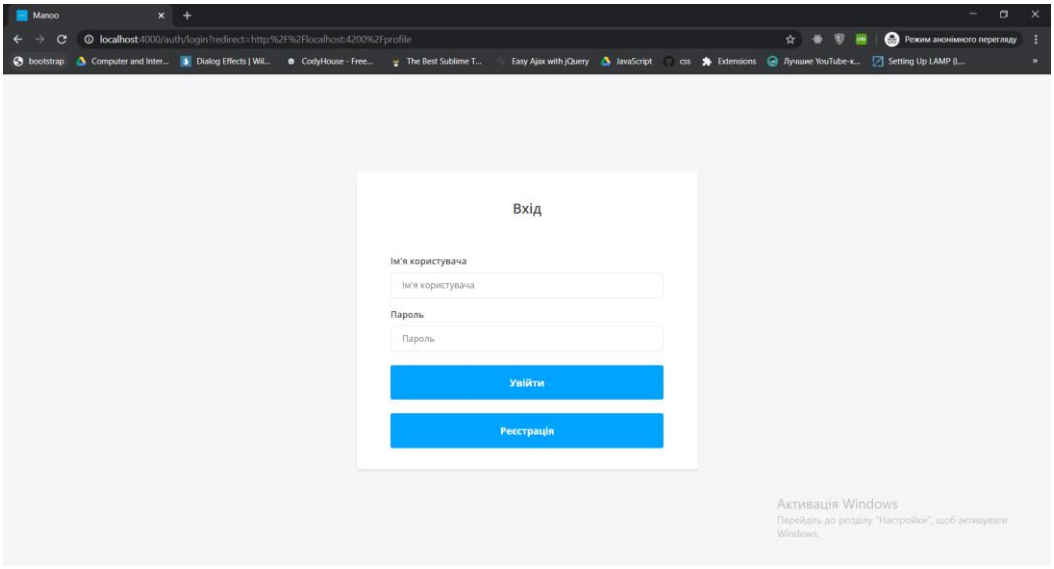

Рисунок 2.4 – Сторінка входу в систему

Для авторизації користувачеві необхідно бути зареєстрованим в системі. Для реєстрації необхідно натиснути на кнопку “Реєстрація”, в результаті чого, користувач буде перенаправлений на сторінку “Реєстрація” (див. рис. 2.5).

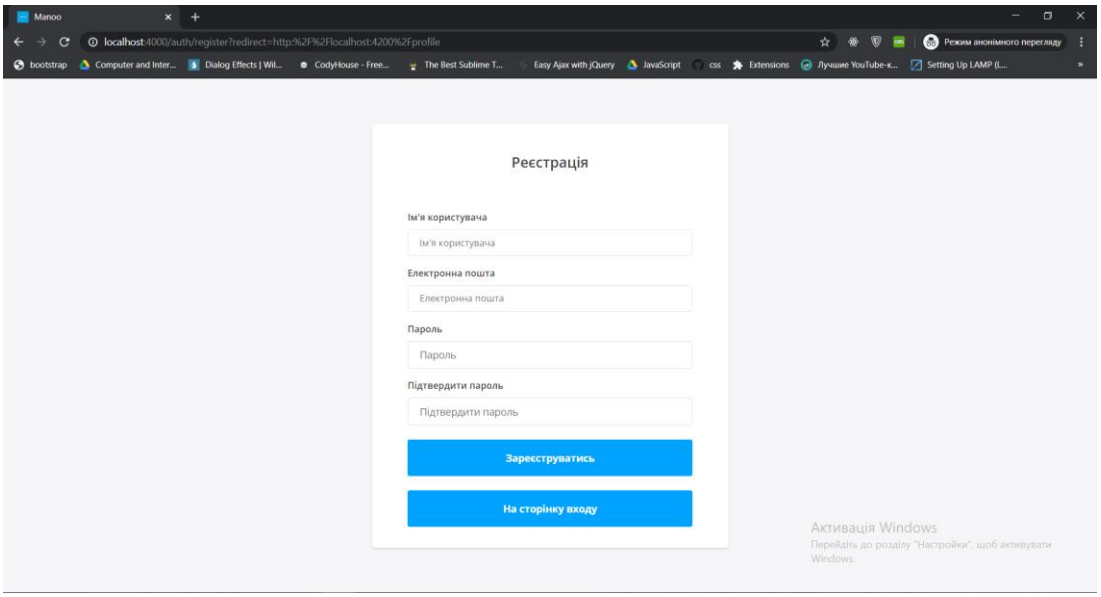

Рисунок 2.5 – Сторінка реєстрації

Після успішної авторизації користувач буде перенаправлений на головну сторінку системи. (див. рис. 2.6).

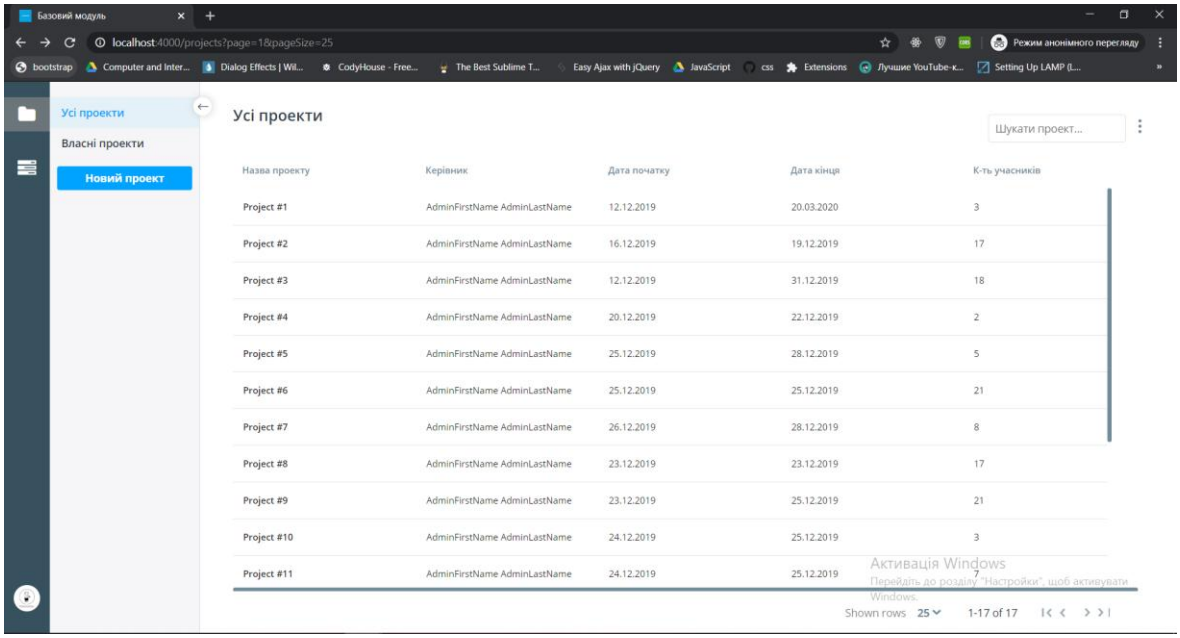

Рисунок 2.6 – Головна сторінка

Головна сторінка містить навігацію з наступними пунктами:

– проекти (піктограма у вігляді папки);

– завдання (піктограма у вигляді списку);

– деталі користувача (пункт у вигляді аватару користувача).

Пункт меню “Проекти” містить підпункти “Усі проекти” та “Власні проекти”. Справа відображено інформацію про проекти.

Для створеня нового проекту користувачеві необхідно натиснути кнопку “Новий проект”. В результаті цього буде відображено модальне вікно (див. рис. 2.7), в якому користувач може вказати детальну інформацію про проект, а саме:

– назву;

– опис;

– часові рамки.

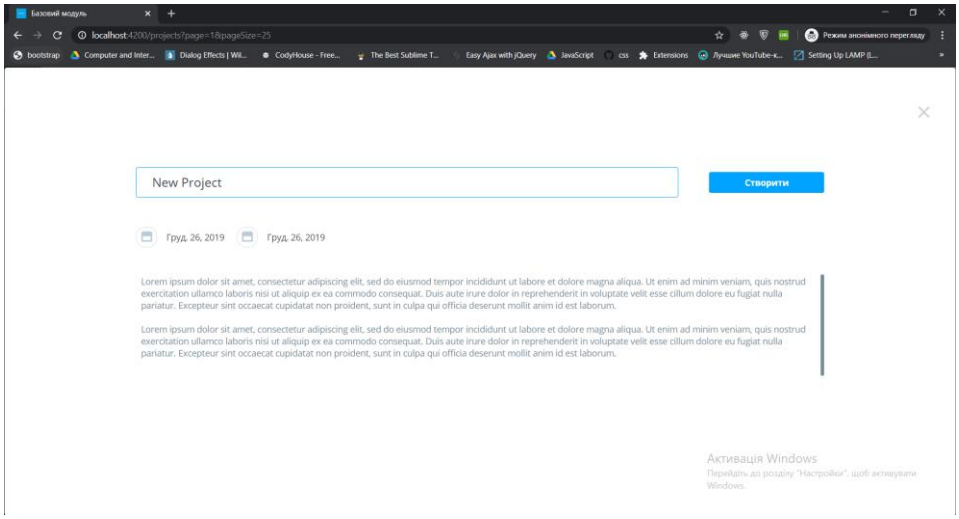

Рисунок 2.7 – Створення проекту

Для перегляду детальної інформації про проект, користувачеві необхідно обрати проект із списку (клацнути на назву проекту). В результаті цього, користувач буде перенаправлений на сторінку (див. рис. 2.8), яка містить інформацію про проект, а також інформацію про:

– завдання проекту;
– спринти проекту;
– команду розробки проекту.

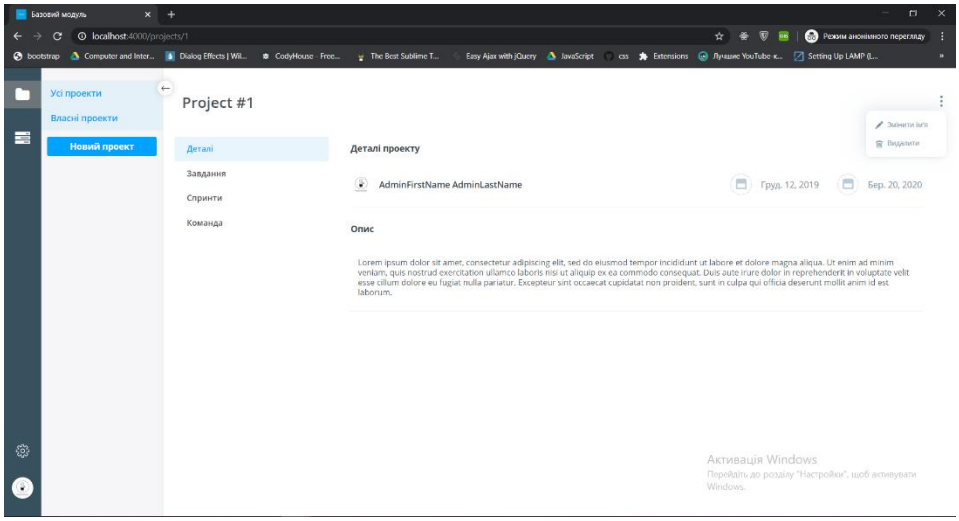

Рисунок 2.8 – Детальна інформація про проект

Крім цього, користувач має можливість редагувати інформацію про проект, а також видаляти проект.

Пункт меню “Завдання” містить інформацію про всі завдання, які необхідно виконати для завершення проекту (див. рис. 2.9). При цьому відображається така інформація:

– назва завдання;

– опис завдання;

– статус завдання;

– виконавець завдання.

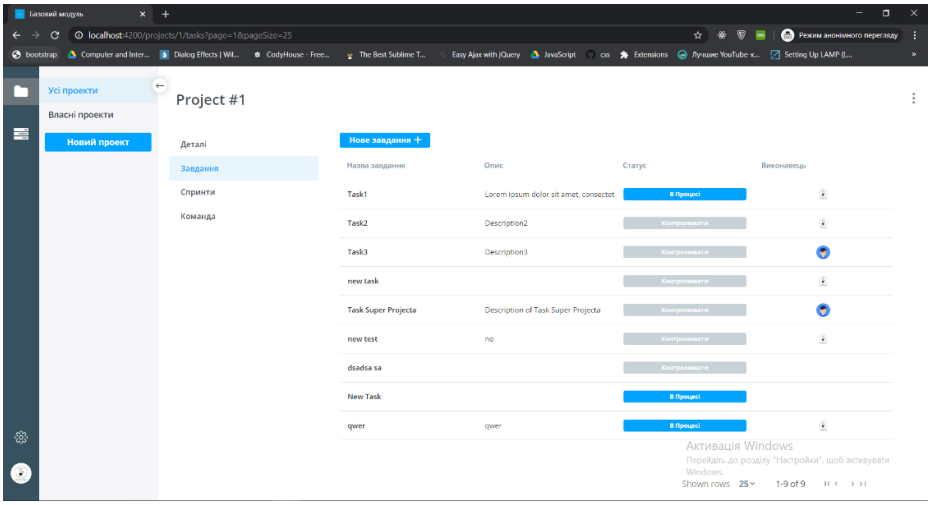

Рисунок 2.9 – Завдання проекту

Форма для створення/редагування завдання зображена на рисунку 2.10.

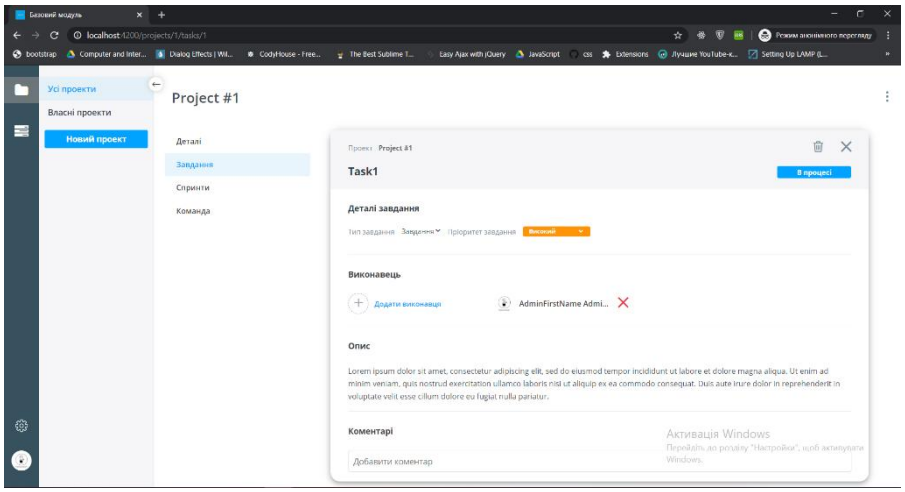

Рисунок 2.10 – Форма створення/редагування завдання

Використовуючи дану форму, користувач має можливість вказати назву завдання, надати детальний опис завдання, вказати тип, статус та пріоритет. Також користувачі мають можливість коментувати завдання.

Доступні наступні типи завдань:

– “Завдання” – вказує, на те, що необхідно реалізувати новий функціонал;
– “Помилка” – вказує на те, що в розробленому функціоналі є помилки роботи.

Доступні наступні пріоритети завдань:

– високий;
– середній;
– низький.

Доступні наступні статуси завдань:

– “В процесі”;
– “Завершено”;
– “Скасовано”;
– “Очікується”.

Пункт меню “Спринти” (див. рис. 2.11) містить інформацію про спринти проекту, а також активний спринт. Форма створення спринта показана на рисунку 2.12.

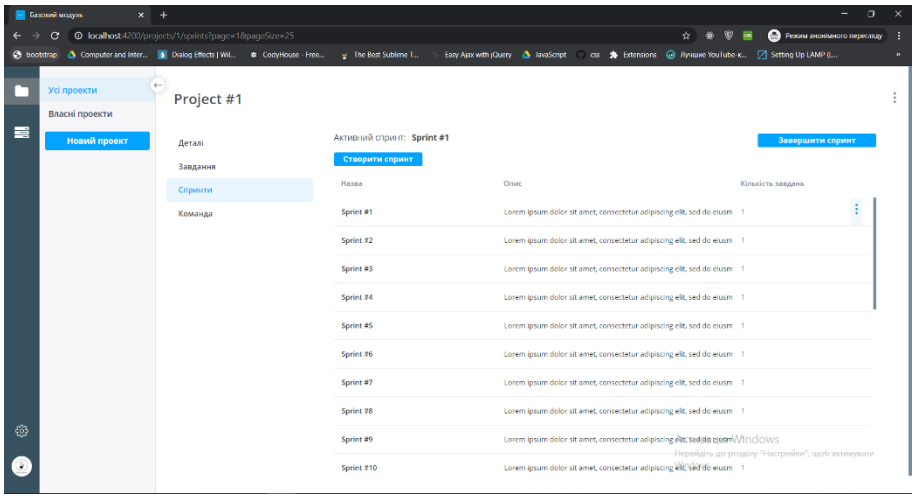

Рисунок 2.11 – Спринти проекту

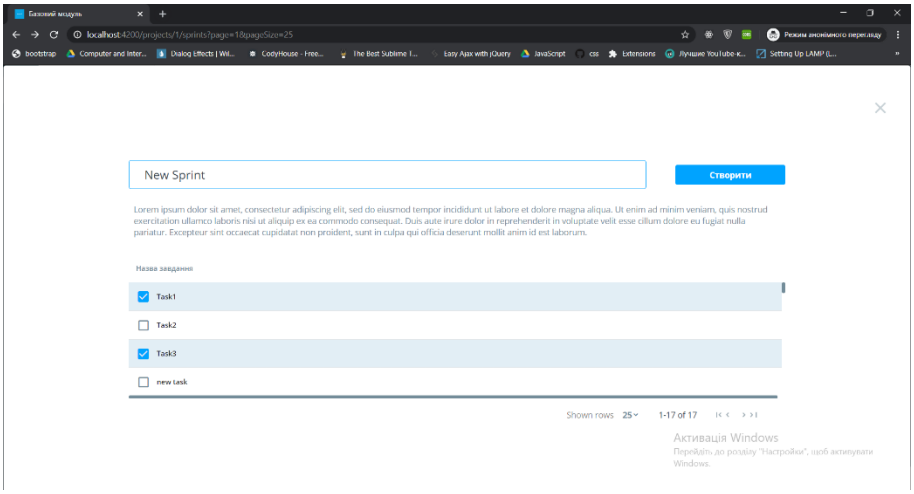

Рисунок 2.12 – Форма створення спринта

Використовуючи дану форму, користувач має можливість вказати назву спринта, детальний опис мети спринта, а також які завдання необхідно виконати впродовж спринта.

Для активації (запуску) спрінта, необхідно вибрати з контекстного меню спринта пункт “Розпочати спринт”, при цьому користувачеві буде надана можливість вказати часові рамки спринта. В один момент часу можливий лише один активний спринт. Для завершення спринта необхідно натиснути кнопку “Завершити спринт”.

Пункт меню “Команда” містить інформацію про учасників процесу розробки (див. рис. 2.13).

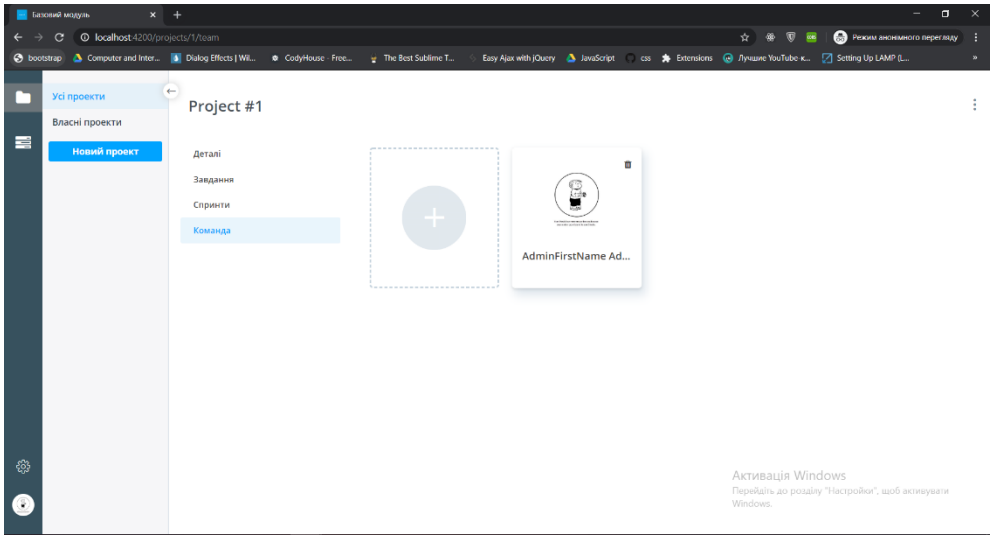

Рисунок 2.13 – Пункт меню “Команда”

Використовуючи даний функціонал, користувач має можливість сформувати команду розробки. Форма запрошення користувачів до проекту зображена на рисунку 2.14.

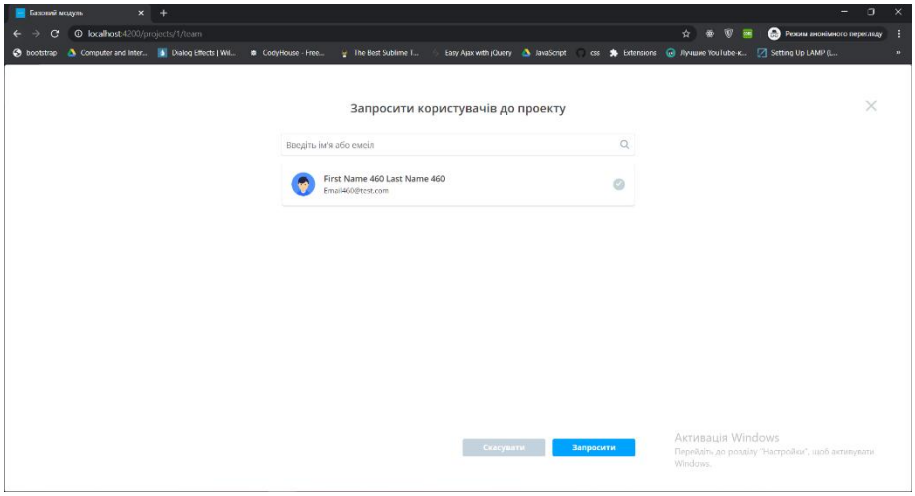

Рисунок 2.14 – Форма запрошення користувачів до проекту

Пункт головного меню “Завдання”, який позначено піктограмою списку, дозволяє користувачеві переглянути всі завдання, які йому необхідно виконати (див. рис. 2.15).

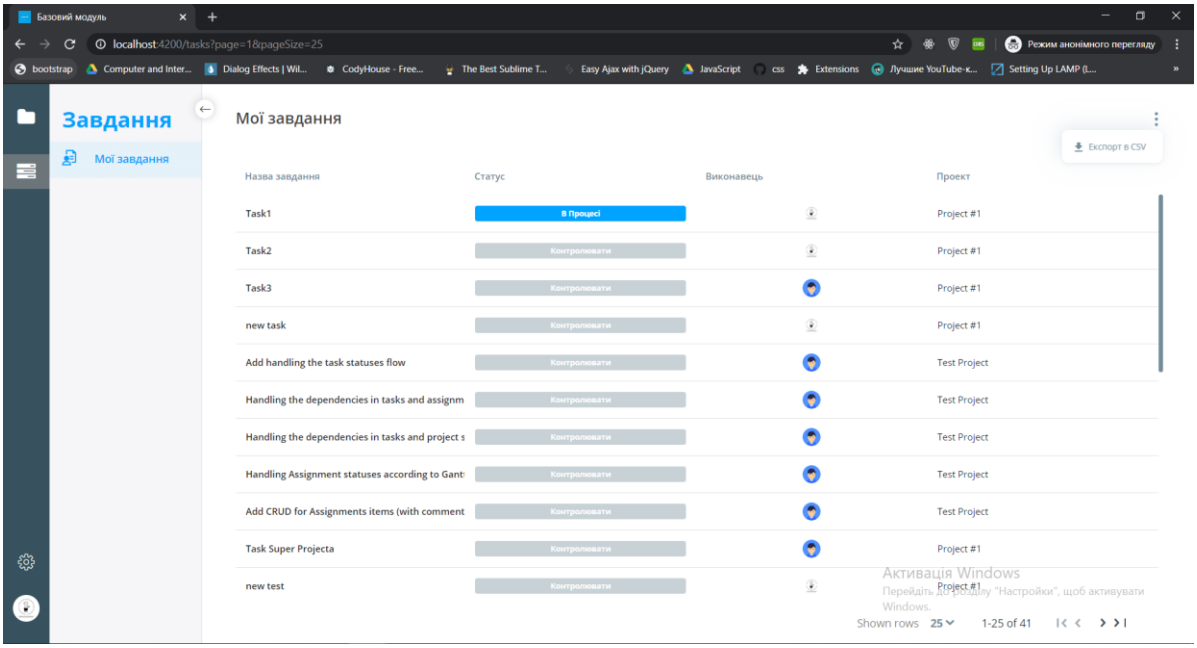

Рисунок 2.15 – Пункт головного меню “Завдання”

Для ведення обліку, в системі передбачена можливість експорту даних. Наприклад, користувач має можливість експортувати всі свої завдання у файл з розширенням csv.

Для редагування інформації про користувача передбачений пункт меню “Налаштування профілю”. Для того щоб переглянути/редагувати інформацію про користувача, необхідно натиснути на аватарку користувача (зліва внизу). В результаті буде відображена форма з інформацією (див. рис. 2.16).

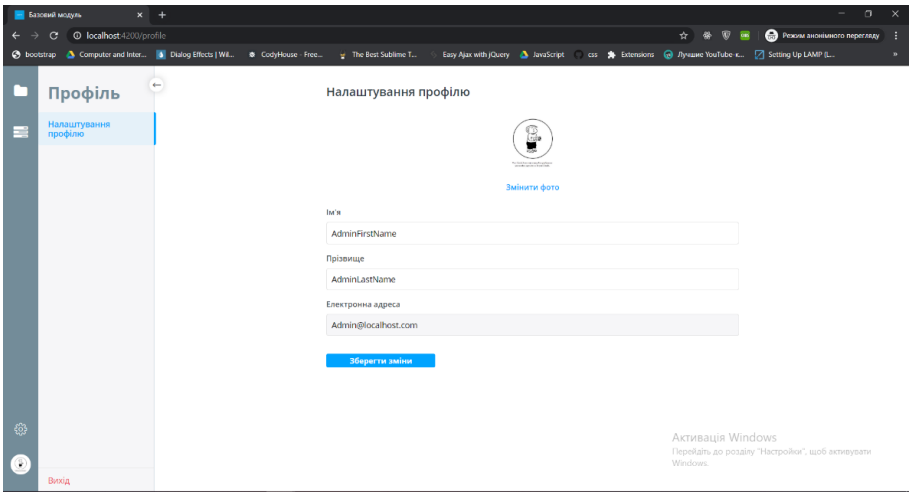

Рисунок 2.16 – Налаштування профілю користувача

3 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

3.1 Загальний підхід до визначення економічної ефективності розробки

Розробка будь-якого проекту неможлива без фінансових витрат, які необхідні для фінансування кожного з етапів розробки. Одним з основних аспектів, які відображають успішність проекту є його економічна ефективність. Оцінка економічної ефективності розробленого програмного продукту, розрахунок передбачених витрат та оцінка ризиків непередбачуваних витрат є головною метою даного розділу.

Для виконання проекту може буде залучено 1 керівника, 10 розробників, 1 DevOps інженера та 4 тестувальник.

Витрати часу по окремих етапах розробки програмного забезпечення відображено в таблиці 3.1.

Таблиця 3.1 – Операції процесу розробки ПЗ і часові затрати

	Місячна зарплата грн.	Денна зарплата грн	Трудомісткість, людино-дні		Основна заробітня плата, грн	
			ООП підхід	Процедурний підхід	ООП підхід	Процедурний підхід
Керівник	8000	360	1	1	360	360
Розробник	12000	545	10	13	5450	7085
Тестувальник	7000	320	4	6	1280	1920

Продовження таблиці 3.1

DevOps інженер	9000	405	2	2	810	810
Всього			15	20	7900	10175

При розробці програмного забезпечення найчастіше використовуються наступні підходи: процедурний та об’єктно-орієнтовний. Використання кожного з підходів передбачає проектування та розробку системи з врахуванням певних принципів та ідей. Процедурний підхід передбачає використання процедур та функцій, що використовуються для проектування системи як монолітного композиту. Об’єктно-орієнтований підхід ґрунтується на основі об’єктів певних класів, що описують певну область, описують певну поведінку (методи) та володіють властивостями (атрибутами). При розробці програмної системи використовувався об’єктно-орієнтовний підхід, що дозволило оптимізувати вартість розробки. Вартість розробки з використанням процедурного підходу наведено для порівняння.

3.2 Визначення ключових витрат

Ключовою витратою є заробітна плата.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата нараховується ра виконану роботу за тарифними ставками, відрядними розцінками чи посадовими окладами і не залежить від результатів господарської діяльності підприємства.

$$ЗП_{осн1} = 7900 \text{ грн};\ ЗП_{осн2} = 10175 \text{ грн}$$

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов'язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, умов виробництва, кваліфікації виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

$$ЗП_{дод} = 0{,}2 \cdot ЗП_{осн} \quad (3.1)$$

$$ЗП_{дод1} = 0{,}2 \cdot ЗП_{осн1} = 1580 \text{ грн.};$$

$$ЗП_{дод1} = 0{,}2 \cdot ЗП_{осн2} = 2035 \text{ грн.}$$

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$$ФЗП = ЗП_{осн} + ЗП_{дод}. \quad (3.2)$$

$$ФЗП_1 = 7900 + 1580 = 9480 \text{ грн.};$$

$$ФЗП_2 = 10175 + 2035 = 12210 \text{ грн.}$$

Крім того, слід визначити відрахування на соціальні заходи:

- єдиний соціальний внесок – 3,6 %;
- військовий збір – 1,5 %;
- ПДФО (прибутковий податок) – 15 %.

Отже, сума відрахувань на соціальні заходи буде становити:

$$Відр_{ЄСВ1} = 0{,}036 \cdot ФЗП = 341{,}30 \text{ грн.};$$

$$Відр_{ЄСВ2} = 0{,}036 \cdot ФЗП = 439{,}60 \text{ грн.};$$

$$Відр_{вз1} = 0{,}015 \cdot ФЗП = 142{,}2 \text{ грн.};$$

$$Відр_{вз2} = 0{,}015 \cdot ФЗП = 183{,}15 \text{грн.}$$

$$Відр_{ПДФО1} = 0{,}15 \cdot ФЗП = 1422 \text{ грн.};$$

$$\text{Відр}_{\text{ПДФО2}} = 0{,}15 \cdot \text{ФЗП} = 1831{,}5\text{грн}..$$

Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3% [10].

Зокрема, видання програмного забезпечення – 36,77%.

Нарахування на Фонд оплати праці (ФОП): $\text{ФОП}_{\text{ЄСВ}} = 0{,}3677 \cdot \text{ФЗП}$

$$\text{ФОП}_{\text{ЄСВ1}} = 0{,}3677 \cdot \text{ФЗП} = 3485.8 \text{ грн.};$$

$$\text{ФОП}_{\text{ЄСВ2}} = 0{,}3677 \cdot \text{ФЗП} = 4489{,}6 \text{ грн}$$

Вього витрат:

$$\text{В}_{\text{ЗП1}} = \text{ФЗП}_1 + \text{ФОП}_{\text{ЄСВ1}} + \text{Відр}_{\text{ЄСВ1}} + \text{Відр}_{\text{вз1}} + \text{Відр}_{\text{ПДФО1}}$$
$$= 9480 + 3485.8 + 341{,}30 + 142{,}2 + 1422 = 14871.3 \text{ грн.};$$

$$\text{В}_{\text{ЗП2}} = \text{ФЗП}_2 + \text{ФОП}_{\text{ЄСВ2}} + \text{Відр}_{\text{ЄСВ2}} + \text{Відр}_{\text{вз2}} + \text{Відр}_{\text{ПДФО2}}$$
$$= 12210 + 4489{,}6 + 439{,}60 + 183{,}15 + 1831{,}5 = 19153.85 \text{ грн.};$$

Також важливою складовою витрат є матеріальні витрати. Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{ві} = q_{\text{і}} \cdot p_{\text{і}}\,, \qquad (3.3)$$

де: $q_{\text{і}}$ – кількість витраченого матеріалу і–го виду;

$p_{\text{і}}$ – ціна матеріалу і–го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{ві}\,. \qquad (3.4)$$

Таблиця 3.2 – Зведені розрахунки матеріальних витрат.

Найменування матеріальних ресурсів	Один. Виміру	Фактично витрачено матеріалів	Ціна 1-ці., грн.	Загальна сума витрат, грн
Папір формату А4	листів	500	0,5	250
Маркер	шт	10	6	60
Тонер для принтера	шт	2	80	160
Флеш-накопичувач	шт	4	80	320
Всього				790

Отже, загальна сума матеріальних витрат становить 790 гривень.

В багатьох випадках існує ряд додаткових витрат які пов’язані із реалізацією проекту, але в більшості випадків їхня сума не перевищує десяти відсотків від загальної собівартості реалізації проекту.

Також варто врахувати електроенергію. Затрати на електроенергію використану 1-цею обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S, \qquad (3.5)$$

де W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Вартість кіловат-години електроенергії слід приймати згідно існуючих на даний час тарифів. Отже, 1 кВт з ПДВ коштує 2,50 грн.

Потужність комп’ютера для створення проекту – 750 Вт, кількість годин роботи необхідних для проекту – 160 годин при об'єктно-орієнтованому підході та 200 години при процедурному підході.

$$З_{е1} = 0{,}4 \cdot 160 \cdot 2{,}50 = 160$$

$$З_{е2} = 0{,}4 \cdot 200 \cdot 2{,}50 = 200$$

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$А = \frac{С_Б \cdot N_A \cdot T_{ФАК}}{Т_{год}} \qquad (3.6)$$

де $С_Б$ – балансова вартість обладнання, грн;

N_A – норма амортизаційнихвідрахувань в рік, %;

$Т_{год}$ – річний робочий фонд часу, год;

$T_{ФАК}$ –фактичний час роботи обладнання по написанню програми, год.

Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Отже, використовуючи в роботі 1 комп’ютер балансовою вартістю 15000 грн. Отже, амортизаційні відрахування будуть рівні:

A_1 = (15000 · 0,6 · 160) / 2080 = 692.5 грн.

A_2 = (15000 · 0,6 · 200) / 2080 = 865.4 грн.

Варто врахувати і накладні витрати, адже вони пов’язані з обслуговуванням виробництва, утриманням апарату управління спілкою та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20-60 % від суми основної та додаткової заробітної плати працівників.

$$НВ = 0{,}5 \cdot ЗП_{осн} \qquad (3.7)$$

де $H_в$ – накладні витрати.

Отже, накладні витрати:

$H_{в1}$ = 7900 · 0,5 = 3950 грн. $H_{в2}$ = 10175 · 0,5 = 5087.5 грн.

3.3 Визначення періоду окупності та собівартості

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг [10].

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 20500 грн (Св1) використовуючи об’єктно-орієнтований підхід, 26100 грн (Св2) при процедурному підході розробки.

Приймемо прибуток на рівні 30%. Для нових інноваційних продуктів, що користуються високим попитом на ринку, ринкову вартість Вр можна встановити вищу.

Отже, вартість розробленого програмного забезпечення:

Вр1= Св1 + 0,3 · Св1 = 20500 + 0,3 · 20500 = 26650 грн..

Вр2= Св2 + 0,3 · Св2 = 26100 + 0,3 · 26100 = 33930 грн..

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу. Економічна ефективність (E_p) полягає у відношенні результату виробництва до затрачених ресурсів [10]:

$$E_p = \frac{П}{C_B} \tag{3.8}$$

де $П$ – прибуток;

$C_в$ – собівартість.

Плановий прибуток ($П_{пл}$) знаходимо за формулою:

$$П_{пл} = В_р - C_в \tag{3.9}$$

Розраховуємо плановий прибуток:

$П_{пл1} = В_{р1} – С_{в1} = 26650 - 20500 = 6150$ грн.

$П_{пл2} = В_{р2} – С_{в2} = 33930 - 26100 = 7830$ грн.

Отже, формула для визначення економічної ефективності набуде вигляду:

$$E_p = \frac{П_{пл}}{C_в}. \tag{3.10}$$

$$Е_{р1} = 6150 / 20500 = 0,3.$$

$$Е_{р2} = 7830 / 26100 = 0,3.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень (T_p):

$$Т_{ок} = \frac{1}{Е} \tag{3.11}$$

Термін окупності дорівнює:

$$Т_{ок} = 1 / 0,3 = 3,3 \text{ роки}$$

У нашому випадку Ток1= Ток2 =1/0,30=3,33 років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Даний розрахунок виконаний у розрахунку на 1 екземпляр програмного продукту без врахування його тиражування.

Загальна вартість пропонованих робіт по розробці програмного продукту становить 20500 грн. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,3, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3 роки та три місяці. Також слід врахувати можливість не одиничного замовлення програми, відповідно її ціна в такому випадку значно понизиться, а при продажі понад план прибуток зросте.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за процедурним методом 50% від початкових витрат, а за об'єктно-орієнтованим – 20%.

Собівартість модернізації:

$$С_в М_1 = 0{,}2 \cdot С_{в1} = 0{,}2 \cdot 20500 = 4100 \text{ грн.},$$

$$С_в М_2 = 0{,}5 \cdot С_{в2} = 0{,}5 \cdot 26100 = 13500 \text{ грн.}$$

Для споживача вартість модернізації:

$$М_1 = 0{,}2 \cdot В_1 = 0{,}2 \cdot 26650 = 5330 \text{ грн};$$

$$М_2 = 0{,}5 \cdot В_1 = 0{,}5 \cdot 33930 = 16965.$$

Таким чином, уже після першої модернізації, загальні витрати на створення і супровід ПЗ для виробника за об'єктно-орієнтованим методом менші, ніж за процедурним.

$$ЗВ_{1\,(вир)} = 20500 + 4100 = 24600 \text{ грн.};$$

$$ЗВ_{2\,(вир)} = 26100 + 13500 = 39600 \text{ грн.}.$$

Як і для споживача:

$ЗВ_1 = 26650 + 5330 = 31980$ грн.;

$ЗВ_2 = 33930 + 16965 = 50895$ грн..

Річна економія витрат за всіма можливими напрямами і додатковими витратами, пов'язаними з супроводом і тільки одноразовою модернізацією (у розрахунку на одиницю продукції) при об’єктно-орієнтованому методі порівняно із процедурним:

$\Delta С_{(вир)} = ЗВ_{2\,(вир)} - ЗВ_{1\,(вир)} = 39600 - 24600 = 15000$ грн.;

$\Delta С = ЗВ_2 - ЗВ_1 = 50895 - 31980 = 18915$ грн..

Чистий приведений дохід (ЧПД) визначається як різниця між сукупними доходами (сукупний грошовий потік) і сукупними витратами (сукупними інвестиціями) взятими за весь період життя інвестицій і дисконтова ними в кожному році на фактор часу. Дисконтування являє собою визначення вартості майбутніх грошових потоків у теперішній момент часу.

Коефіцієнт дисконтування показує, яку величину грошових коштів ми отримаємо з урахуванням фактору часу та ризиків. Він дозволяє перетворити майбутню вартість у вартість на даний момент.

Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$\alpha = \frac{1}{(1+i)^n}; \qquad (3.12)$$

де i – ставка дисконтування або норма дисконту, i = 0,2;

n – час або кількість періодів (років), протягом якого планується отримання доходу.

$$\alpha_0 = 1, \alpha_1 = \frac{1}{1+0,2} = 0,60.$$

Вважатимемо, що обидва програмних продукта одинаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми однаково вплинуть на розмір його додаткових доходів на вкладений капітал. Тому приймемо цю величину за постійну, а порівняння дохідності двох проектів проведемо тільки за витратами.

$$ЧПД_1' = ГП + 0,60 \cdot ГП = 19500 - 0,60 \cdot 7090 = 1,83ГП - 15246 \text{ грн.;}$$

$$ЧПД_1' = ГП + 0,60 \cdot ГП = 19500 - 0,60 \cdot 3900 = 1,83ГП - 17160 \text{ грн..}$$

Чим менші витрати, тим більша дохідність проекту.

$$ЗВ_1 = 19500 + 7090 = 26590 \text{ грн.;}$$

$$ЗВ_2 = 19500 + 3900 = 23400 \text{ грн..}$$

Таблиця 3.3 – Техніко–економічні показники програмного продукту

Показник	Об'єктно-орієнтований підхід	Процедурний підхід
Зарплата основна, грн	7900	10175
Зарплата додаткова, грн	1580	2035
Фонд заробітньої плати, грн	9480	12210
Відрахування на ФОП, грн	3485.8	4489.6
Військовий збір 1,5%	142,2	183,15
Єдиний соціальний внесок 3.6%	341,3	439,6
ПДВ, 15%	1422	1831,5
Разом на виплату плаці, грн	14871.3	19153.85
Матеріальні витрати, грн	790 (730)	790 (730)

Електроенергія, грн	160	200
Амортизація, грн	692.5	865.4

Продовження таблиці 3.3

Накладні витрати, грн	3950	5087.5
Разом на ін.витрати, грн	5592.5	6942.9
Собівартість	20500	26100
Прибуток	6150	7830
Вартість розробленого ПЗ	26650	33930
Економічна ефективність	0,30	0,30
Термін окупності, років	3,33	3,33
Собівартість модернізації	4100	13500
Загальні витрати на розробку	30750	47430
Економія	16680	-

Економія витрат у випадку придбання, супроводу і одноразової модернізації програмного продукту, створеного за об’єктно-орієнтованим підходом, становить 16680 грн.

Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,3, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3 роки та два місяці, бо нормальним терміном окупності є термін, який коливається від 1 до 3 років, в даному випадку допуск 2 місяці можна вважати допустим, тоді розробка вважається доцільною і економічно вигідною.

При використанні об’єктно-орієнтовного підходу зменшується кількість працівників, які залучаються у проект, та зменшуються витрати на реалізацію проекту, але для підтримки проекту і його подальшої модернізації.

Отже, програмний продукт може бути впроваджений та мати подальший розвиток, оскільки він є економічно вигідним за всіма основними техніко-економічними показниками.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1 Охорона праці

Метою даної роботи є веб-застосунку для управління процесом створення програмного забезпечення. Відповідно до постановки задачі, більшість користувачів будуть використовувати розроблену інформаційну систему за допомогою персонального комп'ютера. Отже, користувачі даної програмної системи повинні дотримуватися вимог безпеки під час роботи з екранними пристроями.

Вимоги безпеки до робочих місць працівників з екранними пристроями:

1. Робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів.

2. Для забезпечення безпеки та захисту здоров'я працівників усе випромінювання від екранних пристроїв має бути зведене до гранично допустимого рівня (вплив на людину факторів довкілля - шуму, вібрації, забруднювачів, температури тощо, який не спричиняє соматичних або психічних розладів, а також змін стану здоров'я, працездатності, поведінки, що виходять за межі пристосувальних реакцій) з погляду безпеки та охорони здоров'я працівників [11].

3. Організація робочого місця працівника з екранними пристроями має забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.

4. Освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [11].

5. Мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями має підтримуватись на постійному рівні та відповідати

вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року № 42 (далі - ДСН 3.3.6.042-99) [11].

6. Робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування.

7. Робоче крісло має бути стійким і дозволяти працівнику з екранними пристроями легко рухатися та займати зручне положення.

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

1. Щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.

2. Після закінчення роботи екранні пристрої слід відключати від електричної мережі.

3. У разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі.

4. Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

5. Під час виконання робіт операторського типу, пов’язаних з нервово-емоційним напруженням, у приміщеннях під час роботи з екранними пристроями, на пультах і постах керування технологічними процесами та в

інших приміщеннях мають дотримуватися оптимальні умови мікроклімату відповідно до вимог ДСН 3.3.6.042-99 [11].

Мінімальні вимоги безпеки до екранних пристроїв:

1. Екранні пристрої не мають бути джерелом ризику для працівників.

2. Усе випромінювання, за винятком видимої частини електромагнітного спектра, має бути зведене до незначного рівня з погляду безпеки і охорони здоров'я працівників.

3. Символи на екранних пристроях мають бути чіткими, відповідного розміру. Між символами і рядками символів має бути належна відстань.

4. Зображення на екрані має бути стабільним, без миготінь або інших видів нестабільності.

5. Яскравість та/або контрастність символів має легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов.

6. Вибираючи екрани, слід надавати перевагу таким екранам, які легко та вільно повертаються і нахиляються відповідно до потреби працівника.

7. За необхідності може використовуватись окрема підставка або регульований стіл для розміщення екрана.

8. Екран не має відблискувати або відбивати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями.

9. Вибираючи клавіатуру, слід надавати перевагу такій клавіатурі, яка відкидається і є автономною (відокремленою від екрана), щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук (кисті і верхньої частини руки).

10. Поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання. Розташування клавіш і самі клавіші мають полегшувати роботу із клавіатурою. Позначення клавіш повинно бути достатньо контрастним і розбірливим.

11. Устаткування, яке входить до робочої станції, не має виділяти надлишкового тепла, що може спричинити незручності працівникам під час роботи з екранними пристроями.

12. Під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв'язуваним завданням і є простим у використанні, а де необхідно - адаптованим до рівня знань і досвіду працівника.

Під час розробки, тестування та впровадження інформаційної системи були дотримані всі вимоги, норми та державні стандарти з охорони праці.

4.2 Забезпечення електробезпеки користувачів ПК

Широке застосування електроенергії у всіх галузях промисловості, в побуті, в медицині та ін. вимагає правильного поводження з нею, оскільки порушення правил електробезпеки може призвести до важкої і навіть смертельної травми. Електротравматизм - явище, що характеризується сукупністю електротравм. Електробезпека - це система організаційних та технічних заходів і засобів, які забезпечують захист людей від шкідливої і небезпечної дії струму, електричної дуги, електромагнітного поля і статичної електрики.

Ураження електричним струмом – одна з основних небезпек, які виникають при використанні ПК. Електричний струм, який проходить через організм людини, викликає термічну, електролітичну, біологічну і механічну дії.

. Термічна дія електричного струму призводить до опіків шкіри, нагріву до високої температури кров'яних судин, нервів, серця, мозку та інших органів, які знаходяться на шляху струму, і викликає в них серйозні фукнкціональні

розлади. Термічна дія струму може призвести до руйнування тканин аж до їх обвуглення.

Електролітична дія електричного струму виявляється в електролізі (розкладі) рідин, в тому числі крові, що спричиняє зміну їх фізико-хімічного складу і органів загалом, а також суттєво змінює функціональний склад клітин.

Біологічна дія електричного струму виявляється в подразненні та збудженні живих тканин організму, внаслідок чого спостерігається судомне скорочення м'язів, що може призвести до зупинки дихання, розриву тканин і органів, вивихів кінцівок, спазмів голосових зв'язок.

Механічна дія електричного струму виявляється в розшаруванні тканин і навіть у відриванні частин тіла.

Основні технічні засоби і заходи забезпечення електробезпеки включають:

- ізоляцію струмовідних частин;
- недоступність струмовідних частин;
- електричний розподіл мереж;
- захисне заземлення;
- блокування безпеки;
- засоби орієнтації;
- вирівнювання потенціалів.

Із метою підвищення рівня безпеки, залежно від призначення, умов експлуатації і конструкції, застосовується одночасно більшість з перерахованих технічних засобів і заходів.

Ізоляція струмовідних частин забезпечує технічну працездатність ПК, зменшує вірогідність потраплянь людини під напругу, замикань на землю і на корпус, зменшує струм через людину при доторканні до неізольованих струмовідних частин, що живляться від ізольованої від землі мережі за умови відсутності фаз із пошкодженою ізоляцією [11].

Забезпечення недоступності струмовідних частин. Статистичні дані щодо електротравматизму свідчать, що більшість електротравм пов'язані з дотиком до струмовідних частин (близько 55%). Якщо в установках до 1000В небезпека електротравм пов'язана, переважно, з дотиком до неізольованих струмовідних елементів електроустановок, то за напруги більше 1000В електротравм и можливі і при дотику до ізольованих струмовідних частин.

Основними заходами забезпечення недоступності струмовідних частин є:

- застосування захисних огороджень;
- закритих комутаційних апаратів (пакетних вимикачів, комплектних пускових пристроїв, дистанційних електромагнітних приладів управління споживачами електроенергії);
- розміщення неізольованих струмовідних частин на недосяжній для ненавмисного доторкання до них висоті;
- обмеження доступу сторонніх осіб.

Мета електричного розподілу мереж – зменшення величини ємнісного струму замикання на землю, що збільшує комплексний опір ізоляції фаз відносно землі.

Електричний розподіл мереж застосовують у протяжних або розгалужених мережах з ізольованою нейтраллю, що характеризуються значними ємнісними струмами замикання на землю. Цей метод реалізують шляхом підключення окремих споживачів електричної енергії через розділові трансформатори, що живляться від магістральної мережі.

Захисне заземлення — заземлення точки або точок у системі чи в процесі монтажу системи або в обладнанні, з метою забезпечення електробезпеки. Це спеціальне електричне сполучення із землею або її еквівалентом струмопровідних елементів обладнання, які не повинні перебувати під напругою, але в процесі експлуатації можуть опинитися під напругою, наприклад, у разі пошкодження ізоляції, дефектів дугогасних пристроїв, комутаційних апаратів, в аварійних випадках тощо.

Застосування блокування безпеки. Блокування безпеки застосовуються в електроустановках, експлуатація яких пов'язана з періодичним доступом до огороджених струмовідних частин (випробувальні і дослідні стенди, установки для випробування ізоляції підвищеною напругою), в комутаційних апаратах, помилки в оперативних переключеннях яких можуть призвести до аварії і нещасних випадків, в рубильниках, пусковій апаратурі, автоматичних вимикачах, які працюють в умовах підвищеної небезпеки (електроустановки на плавзасобах, в гірничодобувній промисловості).

Призначення блокування безпеки: унеможливити доступ до неізольованих струмовідних частин без попереднього зняття з них напруги, попередити помилкові оперативні та керуючі дії персоналу при експлуатації, не допустити порушення рівня електробезпеки та вибухозахисту електрообладнання без попереднього відключення його від джерела живлення. Основними видами блокування безпеки є механічні, електричні і електромагнітні.

Засоби орієнтації дають можливість персоналу чітко орієнтуватись при монтажі, виконанні ремонтних робіт і запобігають помилковим діям. До засобів орієнтації належать: маркування частин електрообладнання, проводів і струмопроводів (шин), бирки на проводах, кольорові рішення неізольованих струмовідних частин, ізоляції, внутрішніх поверхонь електричних шаф і щитів керування, попереджувальні сигнали, написи, таблички, комутаційні схеми, знаки високої електричної напруги, знаки постійно попереджувальні тощо.

Вирівнювання потенціалів застосовується з метою зниження можливих напруг дотику і кроку при експлуатації ПК або потраплянні людини під ці напруги за інших обставин. Вирівнювання потенціалів досягається за рахунок навмисного підвищення потенціалу опорної поверхні, на якій може стояти людина, до рівня потенціалу струмовідних частин, яких вона може торкатись, або за рахунок зменшення перепаду потенціалів на поверхні землі чи підлозі приміщень в зоні можливого розтікання струму [11].

При забезпеченні електробезпеки користувача необхідно взяти до уваги таке явище як статична електрика.

Статична електрика - сукупність явищ, що виникає внаслідок накопичення вільного електричного заряду на поверхні.

Заряди статичної електрики можуть утворюватись чи передаватись тілу людини контактним або індукційним шляхом. Статична електрика може діяти на організм людини у вигляді малого струму, який тривалий час проходить через тіло людини, короткочасного електричного розряду, а також електричного поля. Для людини розряди статичної електрики прямої небезпеки не становлять. Однак, враховуючи неочікуваність такого розряду, у людини може виникнути переляк, внаслідок якого може відбутися рефлекторний рух, що в низці випадків призводить до травмування (робота на висоті, біля рухомих незахищених частин устаткування тощо). Тіло людини легко електризується. Його потенціал може досягати 15 кВ, але струми розряду досить малі - мікроампери. Іскрові розряди викликають у людини відчуття слабого або гострого поколювання і лише при різниці потенціалів ЗО кВ викликають тимчасову судому.

Захист від статичної електрики та її небезпечних проявів досягається трьома основними способами:

- запобіганням виникненню та накопиченню статичної електрики;
- прискоренням стікання електростатичних зарядів;
- нейтралізацією електростатичних зарядів.

Електромережа живлення має бути трипровідною з фазовим, нульовим робочим і нульовим захисним провідниками, площа перерізу яких має бути не меншою площі перерізу фазового провідника. Нульовий захисний провідник використовують для заземлення електрообладнання, але використовувати, як нульовий робочий його не можна. Підключення нульового робочого і нульового захисного провідників до одного контактного затискача щита живлення заборонено. Усі провідники мають відповідати номінальним

параметрам мережі, її навантаженню, умовам навколишнього середовища, температурному режиму, типам апаратури та вимогам.

Для підключення переносної електроапаратури застосовують гнучкі проводи в надійній ізоляції. Недопустимими є:

- експлуатація кабелів і проводів з пошкодженою ізоляцією або з такою, що втратила захисні функції, та залишати під напругою кабелі та проводи з неізольованими провідниками;
- застосування саморобних подовжувачів, які не відповідають вимогам ПУЕ до переносних електропроводок;
- застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання;
- користування пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має сліди затемнення або випинання;
- підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною та іншими горючими матеріалами, експлуатація їх із знятими ковпаками (розсіювачами);
- використання електроапаратури та приладів в умовах, то не відповідають рекомендаціям підприємств-виготовлювачів.

ВИСНОВКИ

В результаті виконання дипломної роботи було розроблено веб-застосунок для управління процесом створення програмного забезпечення.

Для реалізації даного програмного продукту було проаналізовано предметну область. Досліджено методології та моделі життєвого циклу програмного забезпечення. Визначено необхідні ролі для успішної реалізації проектів.

Розроблене програмне забезпечення дозволяє планувати та контролювати весь процес життєвого циклу, дозволяє швидко реагувати на зміни до вимог, забезпечує комунікацію між усіма учасниками робочого процесу.

Техніко-економічні показники підтверджують доцільність розробки. Так було розраховано витрати на розробку та впровадження, підтримку проекту. З точки зору організації процесу реалізації у вигляді програмного продукту було досліджено, що використання об’єктно-орієнтованого підходу є більш прийнятним для такого типу проектів.

Задля дотримання державних стандартів та норм з ОП у галузі розробки програмного забезпечення, було проаналізовано нормативно-правові акти, правила та норми. Було створено та забезпечено всі необхідні умови праці для розробки та тестування програмного забезпечення з дотриманням правил експлуатації комп’ютерної техніки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка ПЗ як проектна діяльність [Електронний ресурс] - Режим доступу : URL: https://studfile.net/preview/5203612/page:9/.

2. Життєвий цикл програмного забезпечення - Вікіпедія [Електронний ресурс] - Режим доступу: URL: https://uk.wikipedia.org/wiki/%D0%96%D0%B8%D1%82%D1%82%D1%94%D0%B2%D0%B8%D0%B9_%D1%86%D0%B8%D0%BA%D0%BB_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F.

3. Randall W. Jensen. Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management 2014;160:309-316.

4. Linn B. Learn SQL, 2012; 130:309-316.

5. Richards M. Fundamentals of Software Architecture, 2019

6. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design, 2019 – 830 с.

7. Фреймворк [Електронний ресурс] Режим доступу : URL: https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA

8. Savkin V. Angular Router, 2018 - 340 с.

9. Syed B., Beginning Node.js, 2014 - 520 с.

10. Форма №2 “Звіт про фінансові результати”: методика підготовки [Електронний ресурс]. – Режим доступу: URL: http://osvita.ua/vnz/reports/accountant/17368/.

11. Жидецький В. Ц. Охорона праці користувачів комп’ютерів. – Львів: Афіша, 2000. - 176 с.

ДОДАТКИ

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою
програмної інженерії

"___ "_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської дипломної роботи

на тему: «Розробка веб-застосунку для управління процесом створення програмного забезпечення з використанням Angular для ПП «Магніс»»

Жилавського Богдана Петровича <СПм- - 62 > ТЗ

Керівник роботи:
д. ф-м. н., професор Петрик М.Р.

"___" _____ 2019р.

_____

Виконавець:

студент групи СПм-62

Жилавський Богдан Петрович

"___" _____ 2019р.

_____

м. Тернопіль – 2019

ЗМІСТ

Вступ

1. Підстави до розробки
2. Призначення до розробки
3. Вимоги до програмного продукту
   - 3.1 Функціональні характеристики
   - 3.2 Склад та параметри технічних засобів
   - 3.3 Інформаційна та програмна сполучність
4. Стадії розробки
5. Програмна документація
6. Порядок контролю та приймання

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка веб-застосунку для управління процесом створення програмного забезпечення з використанням Angular для ПП «Магніс»».

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Дипломна робота присвячена створенню програмного забезпечення для управління процесами створення програмного забезпечення.

Предметом дослідження є процес створення та життєвий цикл програмного забезпечення.

Мета роботи - автоматизувати процес створення програмного забезпечення, розробити продукт, який дозволить планувати та контролювати весь процес життєвого циклу, дозволить швидко реагувати на зміни до вимог, забезпечить комунікацію між усіма учасниками робочого процесу

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмне забезпечення має містити наступний функціонал:

- створення облікових записів для користувачів системи;
- створення проектів та завдань проекту;
- створення та налаштування спринтів;

– генерування необхідних звітів для можливості аналізу процесу та прогресу розробки.

3.2 Склад та параметри технічних засобів

ПК з 8 Гб оперативної пам‘яті, встановленою системою Windows, Linux. Не менше 1000 Мб вільного місця на жорсткому диску. Двоядерний процесор з тактовою частотою від 2.4 GHz і більше.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати в різних операційних системах. Розроблювана програмна система повинна бути пристосована до використання у різних бібліотеках. Розробку виконувати з використанням веб-технологій та фреймворку Angular, середовищем розробки PhpStorm, СУБД MySQL.

4. СТАДІЇ РОЗРОБКИ

В ходів реалізації роботи проект повинен пройти крізь наступні стадії розробки:

- аналіз предметної області;
- проектування архітектури;
- реалізація класів і модулів;
- тестування результатів розробки;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Пояснювальна записка;

- Технічне завдання;

- Презентаційний матеріал;

- Інструкція користувача;

- Додатки.

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Приймання проводиться спеціально створеною екзаменаційною комісією в термін до:

“___” грудня 2019р.

Лістинг коду створення бази даних

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET                                                    @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`user` (
  `id` INT NOT NULL,
  `username` VARCHAR(45) NULL,
  `user_img` VARCHAR(45) NULL,
  `email` VARCHAR(45) NULL,
  `created` DATETIME NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`project`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`project` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `created` DATETIME NULL,
  `creatorId` INT NULL,
  PRIMARY KEY (`id`),
```

```
  INDEX `projectToUser_idx` (`creatorId` ASC) VISIBLE,
  CONSTRAINT `projectToUser`
    FOREIGN KEY (`creatorId`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`task`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`task` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `priority` INT NULL,
  `creatorId` INT NULL,
  `start_time` DATETIME NULL,
  `end_time` DATETIME NULL,
  `projectId` INT NULL,
  PRIMARY KEY (`id`),
  INDEX () VISIBLE,
  INDEX `userToTask_idx` (`creatorId` ASC) VISIBLE,
  INDEX `projectToTask_idx` (`projectId` ASC) VISIBLE,
  CONSTRAINT `userToTask`
    FOREIGN KEY (`creatorId`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `projectToTask`
    FOREIGN KEY (`projectId`)
    REFERENCES `mydb`.`project` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`projectMembers`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`projectMembers` (
  `id` INT NOT NULL,
  `userId` INT NULL,
  `projectId` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `toUser_idx` (`userId` ASC) VISIBLE,
  INDEX `toProject_idx` (`projectId` ASC) VISIBLE,
  CONSTRAINT `toUser`
    FOREIGN KEY (`userId`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `toProject`
    FOREIGN KEY (`projectId`)
    REFERENCES `mydb`.`project` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`taskComment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`taskComment` (
  `id` INT NOT NULL,
  `creatorId` INT NULL,
  `taskId` INT NULL,
  `text` VARCHAR(45) NULL,
  `created` DATETIME NULL,
  PRIMARY KEY (`id`),
  INDEX `toUser_idx` (`creatorId` ASC) VISIBLE,
  INDEX `toTask_idx` (`taskId` ASC) VISIBLE,
  CONSTRAINT `toUser`
    FOREIGN KEY (`creatorId`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `toTask`
    FOREIGN KEY (`taskId`)
    REFERENCES `mydb`.`task` (`id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Sprint`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Sprint` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `projectId` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `projecId_idx` (`projectId` ASC) VISIBLE,
  CONSTRAINT `projecId`
    FOREIGN KEY (`projectId`)
    REFERENCES `mydb`.`project` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`sprintTasks`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`sprintTasks` (
  `id` INT NOT NULL,
  `sprintId` INT NULL,
  `taskId` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `dsa_idx` (`sprintId` ASC) VISIBLE,
  INDEX `task_idx` (`taskId` ASC) VISIBLE,
  CONSTRAINT `sprint`
    FOREIGN KEY (`sprintId`)
    REFERENCES `mydb`.`Sprint` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `task`
    FOREIGN KEY (`taskId`)
    REFERENCES `mydb`.`task` (`id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`activeSprints`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`activeSprints` (
  `id` INT NOT NULL,
  `projectId` INT NULL,
  `sprintId` INT NULL,
  `startDate` DATETIME NULL,
  `endDate` DATETIME NULL,
  PRIMARY KEY (`id`),
  INDEX `sprint_idx` (`sprintId` ASC) VISIBLE,
  INDEX `project_idx` (`projectId` ASC) VISIBLE,
  CONSTRAINT `sprint`
    FOREIGN KEY (`sprintId`)
    REFERENCES `mydb`.`Sprint` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `project`
    FOREIGN KEY (`projectId`)
    REFERENCES `mydb`.`project` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Лістинг коду компонента входу в систему

```
import {AppConfig} from 'src/app/app.config';
import {NotifierService} from './../../notifier/notifier.service';
import {UsersProvider} from 'communication';
import {ActivatedRoute} from '@angular/router';
import {ProfileService} from 'src/app/identify/profile.service';
import {IUser} from 'communication';
import {Component, Inject, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from '@angular/forms';
import {Observable} from 'rxjs';
import {FormComponent} from 'components';

@Component({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['../auth.scss', './login.component.scss'],
})
export class LoginComponent extends FormComponent<IUser> implements OnInit {
    autoSave = false;
    loadDataOnParamsChange = false;

    errorMessages = {
        userName: {
            required: 'errors.userName.required',
            minlength: 'errors.userName.minlength',
            maxlength: 'errors.userName.maxlength',
        },
        password: {
            required: 'errors.password.required',
            minlength: 'errors.password.minlength',
            maxlength: 'errors.password.maxlength',
        },
    };

    constructor(
        private _profileService: ProfileService,
        protected _route: ActivatedRoute,
        protected _provider: UsersProvider,
        protected _notifier: NotifierService,
        @Inject(AppConfig) private _communicationConfig: AppConfig,
```

```
) {
    super();
}

ngOnInit() {
    this.loadDataOnParamsChange = false;
    this.loadDataOnInit = false;
    super.ngOnInit();
}

protected createForm(): FormGroup {
    return new FormGroup(
        {
            userName: new FormControl('', [
                Validators.required,
                Validators.minLength(3),
                Validators.maxLength(50),
            ]),
            password: new FormControl('', [
                Validators.required,
                Validators.minLength(6),
                Validators.maxLength(50),
            ]),
        },
        {
            updateOn: 'submit',
        },
    );
}

protected _create(obj: IUser): Observable<any> {
    return this._profileService.login(this.form.value);
}

onSubmit(event) {
    event.preventDefault();
    this.valueChanged = true;
    this.apply();
}

protected _handleErrorCreate() {
```

```
            this.notifier.showError('login.invalid');
        }

        protected _handleSuccessCreate() {
            const redirect = this.route.snapshot.queryParamMap.get('redirect'),
                config = this._communicationConfig,
                authentication = config && config.authentication,
                url = redirect || (authentication && authentication.redirect);

            if (url) window.location.replace(url);
            else this.notifier.showError('Please provide valid redirect URL');
        }
    }
```