Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФАКАЦІЙНА РОБОТА

Веб-система для розподілу та обліку завдань і контролю робочого часу ІТ-компанії
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 190141.01.18.ПЗ

Виконав студент IV курсу група ІПЗ-19-1 _____ К. Є. Сліпокуров
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ І. В. Гурман
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Ю. В. Форкун
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

6 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
05.02.2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сліпокуров Костянтин Євгенійович
Прізвище, ім'я, по батькові студента

1. Тема роботи Веб-система для розподілу та обліку завдань і контролю робочого часу IT-компанії
Керівник роботи Гурман Іван Васильович, кандидат технічних наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 18
2. Строк подання студентом роботи на кафедру 01.06.2023 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, Дослідження предметної області, Проектування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення А3 формату, ER-діаграма бази даних, діаграма використання, діаграма взаємодії

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Гурман І. В. к.т.н. доцент	3.06.2023	3.06.2023
Нормоконтроль	Форкун Ю. В. к.т.н. доцент	5.06.2023	5.06.2023

7. Дата видачі завдання «_02_» __січня_ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1. Збір матеріалу за темою кваліфікаційної роботи (КвР), дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення здач та вимог, розробка технічного завдання	02.01–31.01 2023	
2. Проектування програмного забезпечення	01.02–28.02 2023	
3. Програмна реалізація програмного забезпечення	01.03–10.04 2023	
4. Тестування програмного забезпечення	11.04–30.04 2023	
5. Написання вступу, загальних висновків, оформлення переліку джерел та посилання на застосунків. Оформлення пояснювальної записки КвР згідно вимог	01.05–25.05 2023	

Студент _____ К. Є. Сліпокуров
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ І. В. Гурман
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Веб-система для розподілу та обліку завдань і контролю робочого часу IT-компанії».

Автор проекту: Сліпокуров Костянтин Євгенійович.

Керівник проекту: Гурман Іван Васильович.

Пояснювальна записка: 62 с., 21 рис., 3 табл., 3 дод., 46 джерел.

Графічна частина: 3 креслення формату А3

Веб-система для розподілу та обліку завдань і контролю робочого часу IT-компанії Об'єктом дослідження є IT-компанія, яка займається проектуванням та розробкою різноманітного програмного забезпечення.

Метою проекту є створення веб-системи для контролю часу на різні етапи проектування та розробки програмних продуктів.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені вимоги до системи керування проектами, розроблена загальна архітектура застосунку, спроектована структура бази даних та структура застосунку.

Для розробки програмної системи використано мову програмування C#, сервер бази даних MySQL.

У результаті проектування здійснена програмна реалізація системи для розподілу та обліку завдань і контролю робочого часу IT-компанії

30.05.2023
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ. 190141.01.18.ПЗ	Пояснювальна записка	62		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	15		
5	А3	КвРІПЗ. 190141.01.18.Е8	ER-діаграма бази даних	1		
6	А3	КвРІПЗ. 190141.01.18.Е8	UML-діаграма варіантів використання	1		
7	А3	КвРІПЗ. 190141.01.18.Е8	UML-діаграма варіантів взаємодії	1		

					КвРІПЗ. 190141.01.18.ВД		
Змн.	Арк.	№ докум.	Підпис	Дата			
Виконав		Сліпокуров К. Є		5.06	Веб-система для розподілу та обліку завдань і контролю робочого часу ІТ-компанії	Літ.	Арк. 1 / Аркушів 1
Керівник		Гурман І. В.		5.06			
Н. контр.		Форкун Ю.В.		5.06		ХНУ, ІПЗ-19-1	
Зав. каф.		Бедратюк Л.П.		6.06			

Зміст

					КвРІПЗ. 190141.01.18.ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Виконав		Сліпокуров К. Є		5.06	Веб-система для розподілу та обліку завдань і контролю робочого часу ІТ-компанії	Літ.	Арк. 4 / Аркушів 62
Керівник		Гурман І. В.		5.06			
Н. контр.		Форкун Ю.В.		5.06		ХНУ, ІПЗ-19-1	
Зав. каф.		Бедратюк Л.П.		6.06			

ВСТУП

В інтернеті можна знайти безліч сайтів, що пропонують платні системи менеджменту проектів. Такі системи є надзвичайно зручними, оскільки дозволяють легко та зручно контролювати розподіл часу на різні етапи проектування. Вони дозволяють призначати відповідальних співробітників для кожного етапу та надають звіти про завершення робіт у встановлені строки.

Використання систем менеджменту проектів сприяє покращенню організації та керуванню проектами. Вони дозволяють створювати графіки, розподіляти завдання та встановлювати терміни їх виконання. Крім того, ці системи забезпечують ефективну комунікацію між учасниками проекту, що сприяє покращенню співпраці та обміну необхідною інформацією.

Завдяки системам менеджменту проектів керівництво може швидше та ефективніше контролювати прогрес виконання проектів, виявляти можливі затримки та приймати вчасні рішення. Крім того, такі системи дозволяють зберігати вся історію проекту, що може бути корисним для майбутніх аналізів та уроків, вивчених з попередніх проектів.

Загалом, використання систем менеджменту проектів є важливим кроком у поліпшенні організації проектів. Вони сприяють ефективному плануванню, контролю та звітуванню, що дозволяє досягти успішного завершення проектів вчасно та з високою якістю.

Як зазначено раніше, на ринку існує велика кількість систем менеджменту проектів, тому конкуренція в цій галузі є досить великою. Однак, деякі компанії виявляють бажання мати власну унікальну систему, яка відповідатиме їхнім потребам та специфіці.

Розробка власної системи менеджменту проектів може бути складним завданням. У короткостроковій перспективі це може вимагати значних фінансових витрат і зусиль. Однак, у подальшому це може призвести до зниження

витрат, оскільки компанія матиме повний контроль над системою та зможе вносити зміни та доповнення.

Однією з переваг власної системи є можливість додавання унікального функціоналу, що відповідає потребам конкретної компанії. Це дозволяє створити систему, яка відображає особливості роботи та процеси компанії, сприяючи більш ефективному виконанню завдань та досягненню цілей.

Крім того, власна система менеджменту проектів може забезпечити анонімність, що може бути важливим для деяких компаній. Збереження конфіденційності та захисту даних є пріоритетними завданнями, і власна система дозволяє зберегти контроль над цими аспектами.

У підсумку, розробка власної системи менеджменту проектів може бути складним процесом, проте вона надає компанії можливість мати унікальну та пристосовану до своїх потреб систему. Це може призвести до зниження витрат у подальшому та покращення продуктивності завдяки наявності специфічного функціоналу. Крім того, власна система може забезпечити анонімність та захист даних, що є важливими аспектами для багатьох компаній.

Актуальність обраної теми кваліфікаційної роботи полягає в потребі ІТ компанії мати власну систему розподілу та обліку завдань і часу. Ця потреба виникла внаслідок підвищення вартості місячної підписки на сервіс, яким компанія користувалася раніше.

Застосування власної системи дозволить компанії знизити витрати на підписку та мати повний контроль над функціоналом та розширеннями системи. Крім того, це дозволить компанії пристосувати систему до своїх специфічних потреб і вимог, що є важливим фактором для оптимізації робочих процесів та підвищення продуктивності.

Таким чином, створення власної системи розподілу та обліку завдань і часу є актуальним завданням для компанії з метою зниження витрат і підвищення ефективності роботи.

Метою кваліфікаційної роботи є розробка веб-сервісу, спрямованого на розподіл та облік завдань і робочого часу для IT компанії. Головним завданням проекту є створення функціонального та ефективного інструменту, який допоможе компанії в управлінні завданнями, контролі часу та забезпечить оптимальний розподіл ресурсів.

Для досягнення цієї мети, проект передбачає розробку інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам легко створювати, призначати та відстежувати завдання. Система також буде забезпечувати можливість створення календарних подій, нагадувань та генерації звітів для кращого контролю та аналізу продуктивності.

Крім того, розробка веб-сервісу передбачає забезпечення безпеки та конфіденційності даних. Будуть застосовані відповідні механізми авторизації та шифрування для забезпечення захисту інформації користувачів.

Окрім вищезазначеного, проект передбачає проведення тестування системи для перевірки її функціональності, надійності та відповідності вимогам. Таке тестування дозволить виявити та виправити можливі помилки та недоліки перед введенням системи в експлуатацію.

Узагальнюючи, мета кваліфікаційної роботи полягає у розробці веб-сервісу для розподілу та обліку завдань і робочого часу, який буде відповідати потребам IT компанії, забезпечуючи зручність, ефективність та безпеку в управлінні проектами та ресурсами.

Для досягнення поставленої мети розробки веб-сервісу для розподілу та обліку завдань і робочого часу в IT компанії необхідно вирішити кілька завдань:

– визначення специфіки діяльності IT компанії, яка спеціалізується на розробці програмних продуктів для замовників. Це включає аналіз поточних процесів, методів роботи та особливостей командної роботи в компанії;

– проведення аналізу вимог до інтерфейсу та функціоналу серед працівників IT компанії. Необхідно зібрати вимоги, побажання та пропозиції від

користувачів системи, а також узгодити їх з керівництвом компанії для забезпечення відповідності розроблюваного продукту потребам компанії;

– розробка моделей та алгоритмів майбутнього програмного продукту. На цьому етапі будуть створені моделі бази даних, діаграми взаємодії та інші моделі для опису функціональності та структури системи;

– виконання програмної реалізації майбутнього програмного продукту. За допомогою підходящих технологій та мов програмування будуть реалізовані всі необхідні функції та модулі системи відповідно до визначених вимог та моделей;

– виконання тестових робіт. Після реалізації програмного продукту необхідно провести ретельне тестування системи для виявлення та виправлення можливих помилок, недоліків та проблем. Це включає функціональне тестування, тестування продуктивності, тестування безпеки та інші види тестів.

Вирішення цих завдань дозволить досягти поставленої мети розробки веб-сервісу та створити функціональний, надійний та ефективний інструмент для розподілу та обліку завдань і робочого часу в IT компанії.

1. Дослідження предметної області та постановка задачі

1.1 Дослідження предметної області

Майбутнє програмне забезпечення буде використовуватись в IT-компанії з метою ефективного управління проектами. У зв’язку з цим, важливо провести аналіз взаємодії працівників компанії між собою.

Однією з ключових ролей у роботі над проектом є проектний менеджер (PM). PM – це професіонал, який відповідає за керування проектом в цілому. Він розробляє план проекту, складає технічне завдання, формує команду, налаштовує процес роботи над проектом, забезпечує ефективний зв’язок між командою і замовником. Крім того, він приділяє увагу вирішенню можливих труднощів, що виникають перед командою, контролює якість виконання робіт, щоб проект був завершений вчасно, якісно і в межах бюджету.

PM відіграє ключову роль у створенні ефективної команди, розподілі завдань між її учасниками і встановленні комунікаційних процедур. Він забезпечує злагоджену роботу між усіма членами команди, сприяє обміну ідеями та знаннями, а також стимулює співпрацю і взаємодопомогу між учасниками проекту. PM також відповідає за розподіл ресурсів, таких як людські, фінансові, технічні, з метою забезпечення успішності проекту.

Успішна взаємодія між PM та командою проекту забезпечує виконання проектних цілей, забезпечує високу якість виконання робіт, збільшує шанси на досягнення успіху в конкурентному IT-середовищі.

Основний обов’язок і відповідальність PM – довести ідею замовника до реалізації у встановлений термін, використовуючи наявні ресурси. У межах цього завдання PM необхідно побудувати план розроблення, організувати команду, налаштувати процес роботи над проектом, забезпечити зворотний зв’язок між командами і замовником, усувати перешкоди для команд, контролювати якість і поставку продукту в строк.

Тобто PM є керівником проекту тому у майбутньому програмному забезпечені йому слід забезпечити вищий рівень доступу у межах проекту, створювати, редагувати та видаляти завдання, призначати розробників на завдання, бачити звіти розробників.

Наступною важливою ланкою в роботі над проектом є розробник, також відомий як програміст. Розробник виконує поставлені перед ним завдання вчасно та відповідальний за якісне їх виконання. Він має глибокі знання в області програмування та технологій, що використовуються у проекті. Розробник здатен працювати над різними аспектами проекту, такими як розробка функціональності, веб-інтерфейсу, бази даних та інші. Важливо, щоб розробник володів хорошими аналітичними навичками, щоб вміти ефективно вирішувати завдання та уникати можливих помилок.

Останньою та найвищою ланкою в ієрархії є керівництво компанії. У майбутньому програмному продукті керівництво матиме найвищий рівень доступу, що дозволить їм виконувати важливі функції, такі як створення нових проектів та призначення проектних менеджерів на ці проекти. Керівництво відіграє стратегічну роль у визначенні цілей компанії, прийнятті стратегічних рішень та забезпеченні успішного розвитку бізнесу. Вони також відповідають за надання ресурсів та підтримку для виконання проектів, а також за контроль за їх виконанням у рамках бюджету та графіка.

Із описаного вище можна побудувати графік, який відображає взаємозв’язок між різними учасниками проекту. Діаграма використання відображає взаємодію акторів і системи у контексті функціональності веб-застосунка. Вона надає уявлення про те, які актори будуть взаємодіяти з системою та які функції вони будуть виконувати. На рисунку 1.1 представлено діаграму використання.

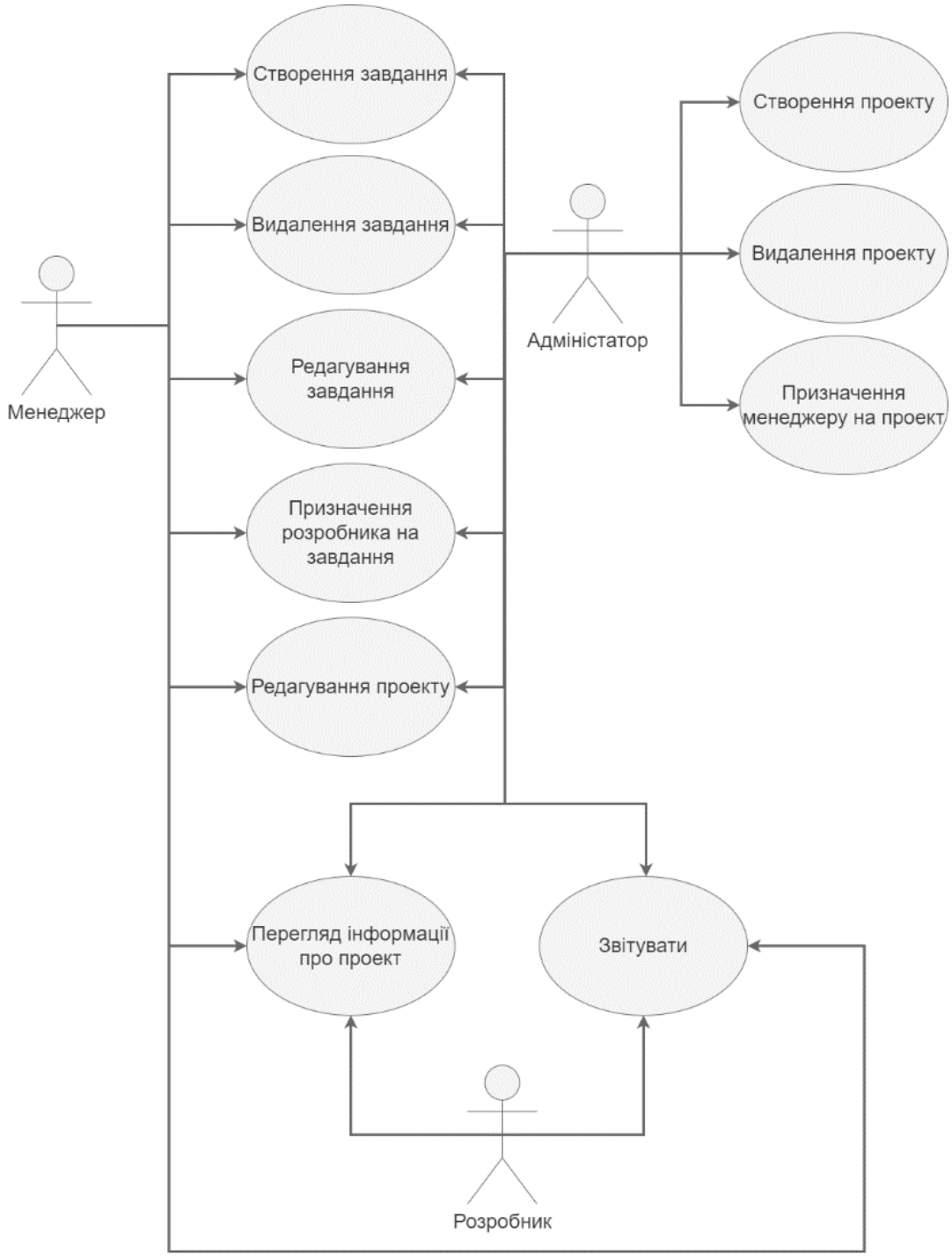


Рисунок 1.1 – Діаграма використання.

На рисунку вище, представлено основний функціонал трьох ролей у майбутньому програмному забезпеченні для управління проектами.

У результаті дослідження були визначені ключові вимоги і функціональні можливості, які має мати веб-система для розподілу та обліку завдань і контролю робочого часу. Виявилось, що така система повинна забезпечувати ефективну

організацію завдань, моніторинг робочого часу, зручний доступ до інформації про проекти та завдання, а також можливість генерування звітів та аналізу даних.

1.2 Аналіз вимог до програмного забезпечення

У попередньому розділі були визначені вимоги до програмного забезпечення (ПЗ), зокрема були визначені три ролі з різним функціоналом.

Перша роль - це адміністратор, який відповідає за керівництво компанією. Адміністратор матиме повний доступ до всього функціоналу програмного забезпечення. Він зможе створювати, редагувати та видаляти проекти. Також адміністратор матиме можливість призначати роль проджект менеджера (PM) на проекти та виконувати всі інші дії, що доступні іншим ролям.

Друга роль - це проджект менеджер. PM відповідає за керування проектом в цілому. Він матиме повний контроль над проектом та зможе створювати, редагувати та видаляти завдання в межах проекту. PM також матиме можливість призначати розробників на завдання та редагувати проект.

Остання роль - це розробник або програміст. Розробник буде мати обмежений функціонал, а саме звітування про виконання поставлених завдань. Звіт може включати текстовий документ з поясненнями та файл з кодом, як прикладом виконаної роботи. Оскільки звіт може складатись з декількох файлів, потрібно передбачити можливість звітування декількома файлами.

Враховуючи ці ролі та їхні функціональні можливості, програмне забезпечення буде забезпечувати ефективну взаємодію між працівниками компанії та керування проектами. Кожна роль матиме свої права та обмеження, що дозволить кожному учаснику внести свій внесок у проект та забезпечити успішне виконання завдання.

Після того, як розробник завершує свою роботу, він звітує про виконання поставленого завдання. Проектний менеджер переглядає звіт та вирішує, чи

виконано завдання належним чином. Він може позначити завдання як виконане або повідомити розробника про необхідність внесення змін або доопрацювання.

Для зручності комунікації між проектним менеджером і розробником, необхідний чат, де вони можуть обмінюватися повідомленнями. Це дозволяє проектному менеджеру легко спілкуватися з розробником та швидко повідомляти про помилки або зміни, які потрібно внести у виконану роботу.

Крім того, згідно з бажанням керівництва, проектний менеджер може виконувати роботу розробника. Це означає, що він має можливість приймати завдання на свою відповідальність і виконувати їх. Це може бути корисно в ситуаціях, коли необхідно розподілити роботу між учасниками команди або забезпечити продуктивність процесу розробки.

Щодо інтерфейсу програмного продукту, немає конкретних вимог, окрім того, щоб він був приємним для очей користувачів. Однак, для полегшення подальшої розробки рекомендується описати прототип інтерфейсу для різних ролей. Прототип допоможе уявити, як буде виглядати та функціонувати програмний продукт.

Головна сторінка адміністратора веб-системи для управління проектами повинна надавати зручний інтерфейс для перегляду і керування всіма проектами. На цій сторінці адміністратор має доступ до списку усіх проектів, включаючи їх назви, імена проектних менеджерів (PM), призначених на ці проекти, а також дати дедлайну для кожного проекту.

На головній сторінці проектного менеджера (PM) будуть відображатися всі завдання, які слід виконати. В цьому списку PM зможе бачити імена розробників, які призначені на кожне завдання, а також дати, до яких ці завдання повинні бути завершені. Це дозволить проектному менеджеру ефективно керувати роботою команди, відстежувати прогрес виконання завдань і вчасно реагувати на будь-які затримки або проблеми.

У розробників на головній сторінці буде присутній опис їх поточного завдання. Це дозволить розробникам швидко ознайомитися з власними завданнями та сконцентруватися на їх виконанні без зайвих відволікань.

Таким чином, кожна головна сторінка спрямована на надання корисної інформації, яка допомагає різним учасникам проекту здійснювати свої функції і взаємодіяти з системою ефективно і продуктивно.

Після узгодження вище описаного з керівництвом компанії можна сформулювати наступний список вимог до системи:

– ролі: Адміністратор, PM (проектний менеджер), розробник;

– адміністратор повинен мати повний доступ до функцій системи, включаючи створення, редагування та видалення проектів, призначення PM на ці проекти, а також інші функції;

– PM має повний контроль над проектом, включаючи редагування проекту, додавання, редагування та видалення завдань, а також призначення розробників на ці завдання;

– розробники повинні мати можливість звітувати про виконання їх поставлених завдань;

– слід розробити чат для спілкування між PM та розробником;

– інтерфейс системи повинен бути приємний та зручний для користувачів;

– головна сторінка для адміністратора має містити перелік всіх проектів, імена PM, які призначені на проекти, та дату дедлайну для проекту;

– головна сторінка PM повинна містити перелік завдань, імена розробників, які призначені на ці завдання, та дату дедлайну;

– головна сторінка розробника повинна містити опис поточного завдання та дату дедлайну.

Враховуючи ці вимоги, система буде здатна ефективно підтримувати процес управління проектами та спілкування між учасниками команди, спрощуючи роботу та покращуючи продуктивність в IT компанії.

В ході аналізу вимог до програмного забезпечення для веб-системи розподілу та обліку завдань і контролю робочого часу IT-компанії було проведено детальний огляд потреб та вимог користувачів. Основною метою було зрозуміти, які функціональні можливості та характеристики має мати система, щоб задовольняти потреби компанії.

1.3 Аналіз економічних показників

Після визначення вимог до програмного забезпечення, наступним кроком є визначення ціни розробки та аналіз вигідності проекту. Для визначення ціни розробки в даному випадку було вирішено використовувати функціонально-орієнтовані метрики, такі як FP (функціональні точки).

FP метрики є широко використовуваним підходом, оскільки вони дозволяють оцінити функціональність або корисність продукту, а не просто його розмір. Це дозволяє більш точно визначити обсяг робіт та ресурси, необхідні для розробки програмного забезпечення.

Основна ідея застосування FP метрик полягає в тому, щоб оцінити кількість функціональних точок, які будуть реалізовані в програмному продукті.

Для визначення ціни розробки за допомогою функціонально-орієнтованих метрик, таких як FP (функціональні точки), використовуються п'ять інформаційних характеристик:

– кількість зовнішніх введень: Підраховуються всі введення, які здійснює користувач і які передають різні прикладні дані програмному застосунку. Сюди включаються дані, які надходять від користувача для обробки. Введення повинні бути відокремлені від запитів, які розраховуються окремо;

– кількість зовнішніх виведень: Підраховуються всі виведення, які передаються користувачеві та які є результатами обчислень, виконаних програмним застосунком. Це можуть бути звіти, екрани, роздруківки, повідомлення про помилки;

– кількість зовнішніх запитів: Під запитом розуміється діалогове введення, яке призводить до негайної програмної відповіді у формі діалогового виведення. Підраховуються всі запити, кожний запит розраховується окремо. Діалогове введення не зберігається в застосунку, а діалогове виведення не вимагає виконання обчислень;

– кількість внутрішніх логічних файлів: Підраховуються всі логічні файли, які використовуються в програмному застосунку. Логічні файли можуть бути частиною бази даних або окремими файлами, які використовуються для зберігання даних;

– кількість зовнішніх інтерфейсних файлів: Підраховуються всі логічні файли, на які посилається даний застосунок.

Для початку було розраховани вище описані інформаційні характеристики, у кращому, гіршому, та вірогідному випадку. Перша характеристика це кількість зовнішніх введень, як зрозуміло з назви це кількість форм для введення даних, теоретично форм має бути 7, дві для редагування створення проекту, дві для створення та редагування завдань, ще два для додавання, та редагування робітників, та одна для звітування. Характеристика 7 була визначена як у кращому випадку, 15 у найгіршому, 10 у вірогідну.

Друга характеристика кількість зовнішніх виведень, ця характеристика значить всі можливі виведення, у тому числі повідомлення про помилки. Підрахувати їх зразу доволі тяжко тому було взято число по інтуїції, приблизно 20, виведення списку проектів, завдань, робітників, виведення поточного завдання, та ще декілька для звітування про помилки. Третя характеристика, кількість зовнішніх запитів, мається на увазі діалогове введення, яке приводить до негайної програмної відповіді у формі діалогового виведення. Подібна поведінка схожа на запити Ajax, та часткові представлення, які є у ASP.NET Core, мова на якій буде розроблятися застосунок. Розрахувати цю характеристику буде складно коли застосунок ще не розроблено, тому було взято 20 у кращому випадку 25 у вірогідному та 40 найгіршому. Четверта характеристика кількість

внутрішніх логічних файлів, під цю категорію можуть підпадати файли JS. У кращому випадку їх буде 3, у гіршому 5, та у ймовірному 3. П’ята та остання характеристика кількість зовнішніх інтерфейсних файлів, таких по ідеї не має бути, тому у кращому буде 1, у вірогідному 1, та у найгіршому 5.

Після підрахунку всіх характеристик, по формулі (1) було розраховано очікуване значення та занесенню у таблицю 1.

$$LOC_{\text{очік } i} = (LOC_{\text{кращ } i} + LOC_{\text{гірш } i} + 4 * LOC_{\text{ймовір } i})/6 \qquad (1)$$

Таблиця 1 – Таблиця розрахунку очікуваного значення

	Кращий	Ймовірний	Гірший	Очікуваний	Складність	Кількість
Зовнішні введення	11	15	20	15	4	61
Зовнішні виведення	10	12	15	12	5	61
Зовнішні запити	15	20	30	21	4	83
Внутрішні логічні файли	10	15	20	15	10	150
Зовнішні інтерфейсні файли	20	25	30	25	7	175
						530

Колонка «Кількість» з таблиці була отримана множенням очікуваного значення на значення складності, а складність була отримана із таблиці 2.

Таблиця 2. Вихідні дані для розрахунків Fp-Метрик

Ім’я характеристики	Ранг, складність, кількість			
	Низький	Середній	Високий	Разом
Зовнішні введення	0x3 = _____	0x4 = _	0x6 = _	= 0
Зовнішні виведення	0x4 = _____	0x5 = _	0x7 = _	= 0
Зовнішні запити	0x3 = _____	0x4 = _	0x6 = _	= 0
Внутрішні логічні файли	0x7 = _____	0x 10= _	0x15 = ___	= 0
Зовнішні інтерфейсні файли	0x5 = _____	0x7 = _	0x10 = _	= 0
Загальна кількість				= 0

Для подальшого розрахунку було визначено значення системних параметрів системи. Результати цих розрахунків наведені в таблиці 3

Таблиця 3 – Таблиця розрахунку системної складності

Коефіцієнт регулювання складності		Оцінка
№	Назва	
1	2	3
F1	Передачі даних	2
F2	Розподілена обробка даних	3
F3	Продуктивність	0
F4	Поширеність використовуваної конфігурації	4
F5	Швидкість транзакцій	5
F6	Оперативне введення даних	4
F7	Ефективність роботи кінцевого користувача	4
F8	Оперативне відновлення	3
F9	Складність обробки	3

Продовження таблиці 3.

1	2	3
F10	Повторна використовуваність	3
F11	Легкість інсталяції	1
F12	Легкість експлуатації	4
F13	Різноманітні умови розміщення	0
F14	Простота змін	4

Далі за формулою (2) було розраховано значення FP.

$$FP = \text{Загальна кількість x} \left(0{,}65 + 0{,}01 \text{ x} \sum_{i=1}^{14} F_i\right) \qquad (2)$$

де загальна кількість, це значення з таблиці на рисунку 1.2, $F_i -$ сумма з таблиці 3

Підставивши значення було отримано число 287, далі отримане значення було поділено на число 200 яке характеризує продуктивність, та було отримано значення витрат часу у люд./місяць, 1,43, та для знаходження вартості, витрати часу було помножено на 300, де 300 це націнка за роботу, загалом було отримано значення вартості у 430 умовних одиниць.

Після визначення вартості розробки програмного забезпечення, було проаналізовано ціна на готові рішення. Першим було розглянуто таск-менеджер під назвою Jira, який є самим популярним у світі, його використовують майже 20% команд. Його ціна 7,50 доларів за користувача на місяць, існує і безкоштовна версія але з вона має багато обмежень. Якщо у компанії працюють 20 робітників то це 150 доларів на місяць, або 5500 гривень. При такій вартості, розробка свого рішення окупиться приблизно за 3 місяця. Другий таск-менеджер Worksection, його ціна 49 доларів на місяць. За таких умов застосунок окупиться трішки більше ніж за 8 місяців.

Очевидно, що вигідність цього проекту є досить суперечливою. З одного боку, на ринку є рішення, які за представлену вартість можуть забезпечити той

же функціонал, а можливо й більший на протязі більш ніж півроку. З іншого боку, можна витратити ці кошти один раз і більше не перейматися про подібні витрати у майбутньому. Вибір залежить від того, яке значення має для компанії наявність готового рішення, що вже пройшло тестування та має підтримку, або виготовлення власного рішення з урахуванням індивідуальних потреб і можливостей компанії.

1.4 Висновки. Постановка задачі

Дослідження предметної області було проведено з метою отримання глибокого розуміння основних аспектів та особливостей, пов'язаних з проектом. Були детально проаналізовані існуючі рішення та конкурентні продукти, а також вивчені методики та практики, що широко використовуються у сфері, в якій розгортається наш проект. Проведене дослідження сприяло визначенню оптимальних стратегій та підходів, які будуть використовуватись в подальшій реалізації проекту.

Аналіз вимог до програмного забезпечення дозволив чітко визначити функціональні та нефункціональні вимоги до застосунку. Були ідентифіковані основні функції та можливості, які повинен надавати застосунок, а також встановлені вимоги до його надійності, продуктивності та безпеки.

Аналіз економічних показників був проведений з метою оцінки вартості розробки та окупності проекту. Використовуючи функціональні точки (FP) метрики та проводячи аналіз, було розраховано приблизну вартість розробки проекту. Також було проведено оцінку окупності проекту на основі розрахунків витрат та потенційних економічних вигод.

В результаті проведеного дослідження предметної області та постановки задачі було отримано чітке уявлення про основні вимоги до програмного забезпечення. Було визначено функціональні та нефункціональні вимоги, розраховано вартість розробки та оцінено економічну доцільність проекту.

Мета полягає у розробці веб-застосунка для ефективного управління проектами в організації. Застосунок має надавати зручний інтерфейс та функціональні можливості для планування, виконання та контролю проектів.

Основні завдання, що вирішуються в рамках проекту, включають:

– створення системи управління проектами: Застосунок повинен надавати засоби для створення та організації проектів. Користувачі повинні мати можливість створювати нові проекти, визначати їх параметри, такі як назва, опис, терміни виконання та бюджет;

– управління завданнями проекту: Застосунок має забезпечувати можливість створення, редагування та видалення завдань в межах проекту. Користувачі повинні мати можливість призначати завдання розробникам та встановлювати терміни їх виконання;

– моніторинг та звітність: Застосунок має забезпечувати моніторинг виконання проектів та завдань. Користувачі повинні мати можливість переглядати статус проектів, відстежувати прогрес завдань та отримувати звіти про продуктивність розробників;

– керування користувачами: Застосунок має підтримувати систему авторизації та аутентифікації користувачів. Адміністратор повинен мати можливість додавати нових користувачів та керувати їх ролями та доступом до функцій системи;

– безпека та захист даних: Застосунок має забезпечувати безпеку та конфіденційність даних. Всі дані повинні бути зашифровані та захищені від несанкціонованого доступу.

Загальна постановка задачі полягає у створенні функціональності, що задовольняє вищезазначені вимоги, забезпечуючи ефективне управління проектами в організації.

2 Проектування програмного забезпечення

2.1 Аналіз та вибір архітектури веб-застосунка

Архітектура веб-застосунка визначається на етапі його проектування та розробки і відіграє важливу роль у забезпеченні успішного функціонування системи. Вона визначає структуру та організацію компонентів застосунка, його взаємозв'язки та способи взаємодії між ними.

Правильний вибір архітектури дозволяє створити систему, яка працює ефективно та швидко. Вона дозволяє оптимізувати використання ресурсів, зменшити навантаження на сервер та забезпечити швидкий відгук системи на запити користувачів. Правильно спроектована архітектура також сприяє забезпеченню масштабованості системи, що дозволяє легко розширювати її функціональні можливості та оброблювати більші обсяги даних та трафіку.

Окрім того, правильно побудована архітектура допомагає зменшити ризики виникнення проблем під час експлуатації системи. Вона дозволяє покращити стабільність, надійність та безпеку застосунка шляхом правильного розподілу функціональності та застосування ефективних патернів проектування.

Процес аналізу та вибору архітектури є ключовим етапом у розробці програмного забезпечення і має бути детальним та комплексним. При його проведенні необхідно розглянути різні варіанти архітектур і оцінити їх переваги та недоліки в контексті конкретного проекту.

Врахування різних факторів є важливим кроком у процесі вибору оптимальної архітектури. Потрібно врахувати функціональні вимоги проекту, обсяги даних, очікувану кількість користувачів, технічні обмеження та бюджет проекту. Ці фактори впливають на вибір архітектурного рішення, яке найкраще відповідатиме потребам проекту.

Клієнт-серверна архітектура є одним з перших варіантів архітектур, який може бути успішно використаний при розробці веб-застосунка. В цій архітектурі клієнти взаємодіють з сервером, який забезпечує доступ до даних.

Цей варіант архітектури особливо підходить для систем зі складним функціоналом та великою кількістю користувачів. Сервер є центральним елементом, який обробляє запити від клієнтів та надає необхідні дані та ресурси. Це забезпечує стабільність та надійність системи, оскільки весь оброблювальний процес відбувається на сервері.

Клієнт-серверна архітектура має свої переваги. Вона дозволяє забезпечити централізоване управління даними та функціональністю, що спрощує розгортання та підтримку системи. Крім того, така архітектура може бути дуже ефективною для великих проектів, оскільки розподілення завдань між клієнтом та сервером дозволяє забезпечити оптимальне використання ресурсів.

Звичайно, клієнт-серверна архітектура також має свої недоліки. Наприклад, вона може потребувати встановлення та підтримки серверного обладнання та програмного забезпечення. Крім того, вона може бути менш гнучкою у порівнянні з іншими архітектурними варіантами, такими як мікросервісна архітектура. Однак, з правильним розподілом функцій між клієнтом та сервером, цей варіант архітектури може бути досить ефективним та надійним для веб-систем

Схематичне зображення роботи клієнт-серверної архітектури можна побачити на рисунку 2.1.

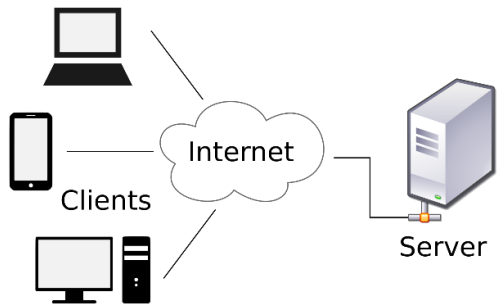


Рисунок 2.1 – Схематичне зображення роботи клієнт-серверної архітектури.

Другий розглянутий варіант архітектури - це модель клієнт-сервера зі статичним генеруванням контенту. В цій моделі сервер має здатність генерувати

HTML-сторінки, які потім передаються клієнтам. Зазвичай така архітектура використовується для веб-застосунків зі статичним контентом та невеликим обсягом даних.

Варто відзначити, що цей підхід може бути менш ефективним для систем зі складним функціоналом та динамічним контентом. У таких випадках серверу доведеться генерувати нові HTML-сторінки для кожного запиту, що може спричинити перевантаження сервера та зниження продуктивності системи. Однак, якщо веб-система не потребує складного функціоналу та динамічного контенту, то модель зі статичним генеруванням контенту може бути достатньо ефективною та легко масштабуватись.

Третій варіант архітектури - модель клієнт-сервер з динамічним генеруванням контенту - є одним з найбільш популярних варіантів для сучасних веб-систем. У цій моделі сервер генерує контент на основі запитів користувачів, що дозволяє забезпечити високу гнучкість та ефективність системи, оскільки клієнти отримують потрібну інформацію в режимі реального часу. Ця архітектура підходить для веб-систем зі складним функціоналом та великою кількістю користувачів, оскільки дозволяє забезпечити високу продуктивність та масштабованість системи. Однак, вона може потребувати більш високої продуктивності сервера та мережі для забезпечення ефективної роботи. Для забезпечення найкращих результатів використання такої архітектури необхідно враховувати особливості проекту та розроблювати відповідні стратегії кешування та оптимізації роботи системи. Крім того, необхідно мати належні знання та досвід у розробці застосунків з використанням такої архітектури для успішної реалізації проекту.

Четвертий варіант архітектури - це SPA (Single Page Application) архітектура. У цій архітектурі весь контент завантажується разом зі сторінкою при її першому завантаженні, а далі відбувається динамічна зміна контенту на сторінці без перезавантаження сторінки. Цей варіант підходить для веб-систем зі складним функціоналом та великою кількістю динамічного контенту, а також для

систем з великою кількістю користувачів. Однак, він може потребувати більш високої продуктивності браузера користувача та може бути менш ефективним для систем зі статичним контентом.

З урахуванням функціональних вимог та очікуваної кількості користувачів веб-застосунка було обрано оптимальний варіант архітектури, яка забезпечує найвищу ефективність та гнучкість системи. Відповідно до цього вибору, модель клієнт-сервер з динамічним генеруванням контенту є найкращим варіантом.

Застосування динамічної генерації контенту дозволить серверу генерувати вміст на основі запитів користувачів у режимі реального часу. Це забезпечить максимальну ефективність та точність обробки інформації, оскільки веб-застосунок зможе надавати дані та результати обчислень у відповідь на кожний запит користувача.

За допомогою динамічного генерування контенту сервер може адаптуватись до змінних потреб користувачів, надавати персоналізовані результати та взаємодіяти з ними у режимі реального часу. Це забезпечить високу гнучкість системи та задоволення потреб різних користувачів.

Враховуючи вищезазначені переваги, модель клієнт-сервера з динамічним генеруванням контенту є найкращим вибором архітектури для розробки веб-застосунка, забезпечуючи оптимальну ефективність та гнучкість системи.

Завдяки обраній архітектурі, веб-система зможе успішно опрацьовувати велику кількість користувачів та забезпечити їм швидкий та ефективний доступ до необхідної інформації. Проте, варто враховувати, що для досягнення високої продуктивності системи необхідно мати сервери з достатньою обчислювальною потужністю та мережеві з'єднання з високою швидкістю передачі даних.

Для забезпечення ефективної роботи веб-застосунка важливо мати сервери, які здатні швидко обробляти запити користувачів та генерувати контент у реальному часі. Також важливо мати надійне мережеве з'єднання з високою пропускною здатністю, щоб забезпечити швидку передачу даних між сервером і клієнтами. Тільки за наявності достатньо продуктивних серверів та швидкої

мережі можна гарантувати максимальну ефективність та продуктивність роботи веб-застосунка, що відповідає потребам користувачів та виконує всі вимоги, які до нього ставляться.

Враховуючи ці фактори, треба забезпечити належні технічні ресурси для веб-системи, щоб забезпечити ефективну роботу веб-системи і задовольнити потреби користувачів.

Далі була розроблена діаграма взаємодії, яка візуально представляє взаємодію між різними компонентами веб-системи для розподілу та обліку завдань і контролю робочого часу IT-компанії. Ця діаграма надає цінну інформацію про послідовність дій та обмін повідомленнями між різними модулями системи.

На діаграмі взаємодії можна побачити, які компоненти взаємодіють між собою та в якому порядку вони виконують свої функції. Вона дозволяє виявити ключові етапи обробки даних, передачі повідомлень та виконання операцій у системі.

На рисунку 2.2 зображено діаграму взаємодії.

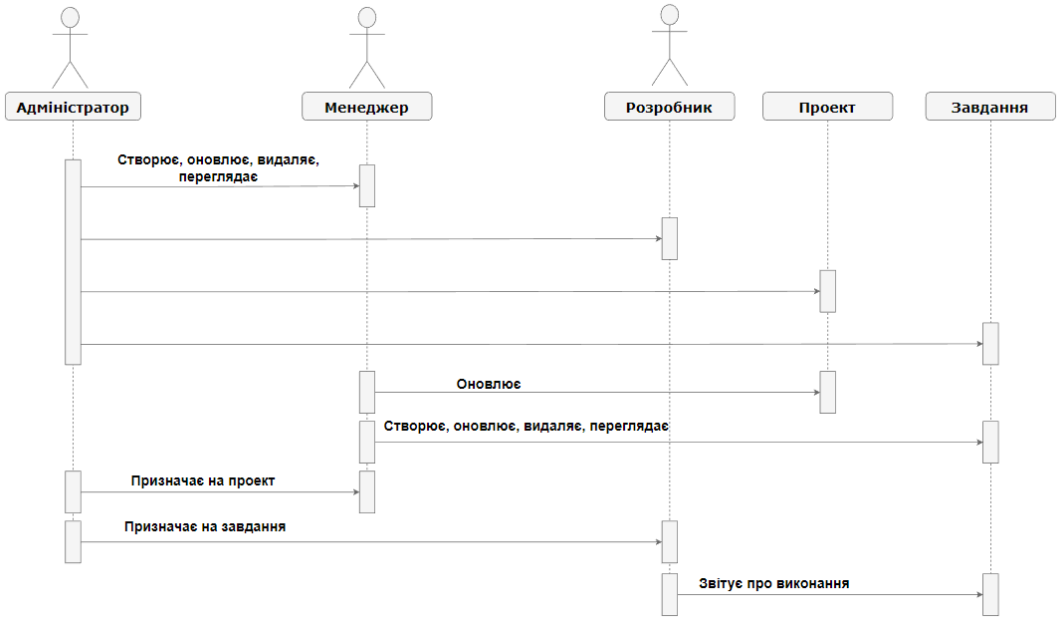


Рисунок 2.2 – Діаграма взаємодії

Ця діаграма стане корисним інструментом, оскільки вона допоможе краще зрозуміти, як компоненти системи співпрацюють між собою та як керуються даними та функціональністю. Вона також допоможе краще зрозуміти взаємодію між компонентами системи та виявити можливі проблеми або покращення.

2.2 Опис структури даних та моделі бази даних

Оскільки основним завданням даної веб-системи є ефективне ведення обліку робочого часу та контроль за виконанням завдань, значна увага повинна бути приділена розробці структури даних та моделі бази даних. Вона повинна бути добре продуманою та ретельно розробленою, щоб забезпечити оптимальну роботу системи.

В розділі про структуру даних та модель бази даних будуть розглянуті основні елементи, які відіграють важливу роль у веб-системі, спрямованій на розподіл завдань та облік робочого часу співробітників IT-компанії. При проектуванні моделі бази даних будуть визначені сутності, їхні атрибути та взаємозв'язки, які відображатимуть взаємодію між різними елементами системи.

Важливо врахувати, що модель бази даних має забезпечувати ефективне зберігання, оновлення та доступ до даних. Вона повинна бути гнучкою та масштабованою, щоб забезпечити оптимальну продуктивність системи навіть при зростанні обсягів даних та кількості користувачів.

У системі будуть зберігатися дані про користувачів, проєкти, завдання та робочий час. Кожен користувач матиме свій особистий кабінет, де будуть доступні список призначених йому завдань, інформація про його робочий час та результати виконання завдань. Кабінет дозволить користувачеві зручно керувати своїми завданнями та відстежувати їх стан.

Кожен проект у системі матиме власну сторінку, де буде відображена інформація про стан проекту. На цій сторінці буде доступний список учасників

проекту та завдань, призначених для виконання. Це дозволить всім учасникам проекту бути в курсі його поточного стану та розподілу завдань.

Кожне завдання в системі буде мати свій опис, в якому будуть вказані вимоги та очікувані результати. Також буде встановлений строк виконання завдання та призначені виконавці, які відповідають за його виконання. Це забезпечить чітку організацію робочого процесу та сприятиме ефективній комунікації між учасниками проекту.

Дані про користувачів, проекти, завдання та робочий час будуть зберігатися у базі даних системи. Це дозволить забезпечити структуроване та надійне зберігання інформації, швидкий доступ до неї та можливість здійснення різних запитів та аналізу даних для підтримки прийняття рішень.

Усі ці елементи структури даних та моделі бази даних сприятимуть ефективному розподілу завдань, контролю за робочим часом та забезпечать зручний інтерфейс для користувачів системи.

Структура даних буде представлена у вигляді набору таблиць у базі даних. База даних буде складатися з декількох таблиць, що будуть взаємодіяти між собою за допомогою зв'язків. Також буде використана нормалізація для забезпечення ефективності та структурованості бази даних.

Опис таблиць бази даних:

– користувачі (Users): ця таблиця містить інформацію про користувачів системи, таку як логін, ім'я, прізвище, електронну пошту, пароль, роль користувача та інше;

– клієнти (Clients): ця таблиця містить інформацію про клієнтів IT-компанії, таку як ім'я, контактну інформацію, деталі про проекти, з якими працює клієнт, та інше;

– проекти (Projects): ця таблиця містить інформацію про проекти, з якими працює IT-компанія, таку як назва проекту, опис проекту, статус проекту;

– завдання (Tasks): ця таблиця містить інформацію про завдання, які призначені для виконання співробітниками ІТ-компанії, таку як опис завдання, дата створення, дата закінчення, статус завдання та інше;

– повідомлення (Messages): ця таблиця містить інформацію про повідомлення, які користувачі можуть надсилати один одному в системі;

– ролі (Roles): ця таблиця містить інформацію про доступні ролі користувачів в системі, такі як адміністратор, менеджер, розробник тощо;

– історія входів (LoginHistory): ця таблиця містить інформацію про всі входи користувачів до системи, такі як дата та час входу, IP-адреса, браузер.

Для того, щоб уникнути повторення інформації та забезпечити цілісність даних, у таблицях бази даних будуть використовуватись зв'язки між таблицями. Зв'язки між таблицями визначають, які дані з однієї таблиці відносяться до певного запису в іншій таблиці.

У даній веб-системі було визначено зв'язки між таблицями, зокрема:

– таблиця Користувачі (Users) матиме зв'язки з таблицями Ролі (Roles) та Історія входів (LoginHistory). Кожен користувач повинен мати одну роль, яка забезпечує йому відповідність прав доступу. Таблиця Історія входів дозволяє відстежувати всі входи користувачів до системи;

– таблиця Клієнти (Clients) матиме зв'язок з таблицею Проекти (Projects). Кожен клієнт може мати декілька проектів;

– таблиця Проекти (Projects) матиме зв'язок з таблицею Завдання (Tasks). Кожен проект може мати декілька завдань;

– таблиця Завдання (Tasks) матиме зв'язки з таблицями Користувачі (Users). Кожне завдання має відповідального користувача, який відповідає за його виконання;

– таблиця Повідомлення (Messages) матиме зв'язки з таблицями Користувачі (Users). Кожне повідомлення має одержувача та відправника.

З метою поліпшення сприйняття та більш чіткого представлення структури бази даних, була розроблена ER-діаграма. Ця діаграма відображає сутності (такі як таблиці) та зв'язки між ними у базі даних веб-застосунка.

Завдяки ER-діаграмі можна визначити основні сутності, їхні атрибути та зв'язки, що існують між ними. Це дозволяє краще зрозуміти структуру бази даних та її логіку.

ER-діаграма допомагає забезпечити належну організацію та структуру даних у базі даних, що є важливим фактором для ефективної роботи та забезпечення надійності веб-застосунка. Вона допомагає зрозуміти взаємозв'язки між різними сутностями та спрощує розробку запитів та операцій з базою даних.

Розроблена ER-діаграма, зображена на рисунку 2.3.

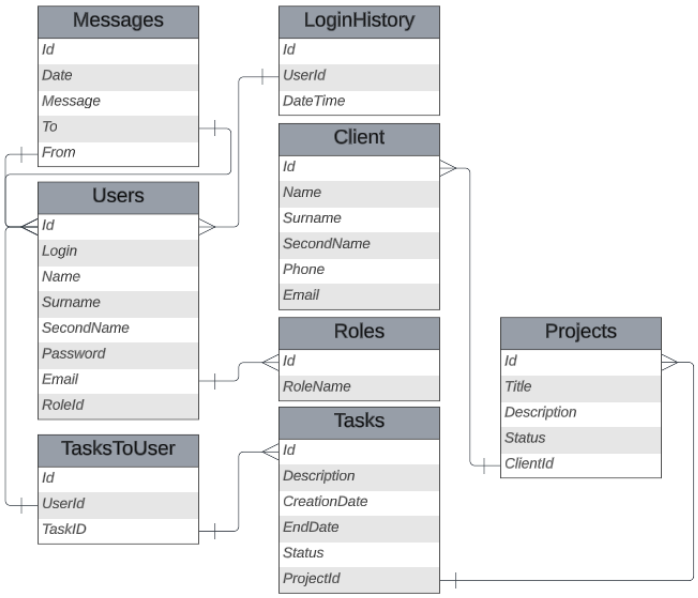


Рисунок 2.3 – ER-діаграма бази даних

Описана структура даних дозволяє ефективно організувати роботу з завданнями, контролювати робочий час та зберігати важливі дані про користувачів, проекти та інші елементи системи. Вона забезпечує належну організацію даних, зручність доступу та швидкість операцій з базою даних.

2.3 Проектування серверної частини веб-застосунка

У процесі проектування серверної частини веб-застосунка для розподілу та обліку завдань і контролю робочого часу IT-компанії, було вирішено скористатися технологією ASP.NET Core. Ця технологія визнана потужним інструментом для розробки веб-застосунків, здатним забезпечити високу продуктивність, масштабованість, безпеку та зручну розробку.

ASP.NET Core має багатий набір функціональних можливостей, що дозволяє ефективно створювати складні веб-застосунки. Вона пропонує різні модулі та інструменти для обробки запитів, маршрутизації, автентифікації та авторизації користувачів, що спрощує розробку та підтримку веб-застосунків. Багатопотоковий режим роботи ASP.NET Core дозволяє обробляти багато запитів одночасно, що забезпечує високу продуктивність та швидку відповідь на запити користувачів.

ASP.NET Core також надає можливість легко масштабувати веб-застосунок, що дозволяє йому відповідати зростаючим потребам користувачів. Безпека є ще однією важливою перевагою цієї технології, оскільки вона надає засоби для захисту веб-застосунка від різних загроз і атак.

Вибір ASP.NET Core в якості технології розробки веб-застосунка дозволить швидко та ефективно створити потужний та надійний веб-сервіс.

На попередньому етапі проектування серверної частини веб-застосунка для розподілу та обліку завдань і контролю робочого часу IT-компанії, було приділено особливу увагу розробці бази даних, яка є одним з головних компонентів серверної частини. Для полегшення роботи з базою даних та забезпечення її інтеграції зі стеком технологій ASP.NET Core, використовувалась технологія ORM (Object-Relational Mapping) - Entity Framework Core. Ця технологія дозволяє працювати з базою даних на вищому рівні абстракції, що спрощує роботу з даними та забезпечує більш зручну інтеграцію з серверною

частиною веб-застосунка. Також варто відзначити, що Entity Framework Core забезпечує безпеку та масштабованість бази даних.

У процесі вибору архітектури веб-застосунка для даного проекту була обрана модель, яка забезпечує поділ застосунка на окремі модулі з різними функціями та можливістю масштабування. Для цього було використано популярний паттерн проектування - Model-View-Controller (MVC). Цей паттерн дозволяє розділити застосунок на три компоненти: модель, представлення та контролер.

Модель відповідає за обробку даних та бізнес-логіку застосунка. Вона взаємодіє з базою даних та забезпечує доступ до необхідної інформації.

Представлення відображає дані користувачам та відповідає за їхнє візуальне подання. Воно включає в себе HTML-шаблони та стилізацію застосунка.

Контролер відповідає за обробку запитів користувачів та взаємодію з моделлю та представленням. Він приймає запити від користувачів, виконує відповідні дії та оновлює стан моделі і представлення.

Використання архітектури MVC дозволяє досягти розділення відповідальностей між компонентами, що сприяє покращенню модульності, зручності розробки та масштабованості веб-застосунка. Схематичне зображення роботи такої архітектури можна побачити на рисунку 2.4.

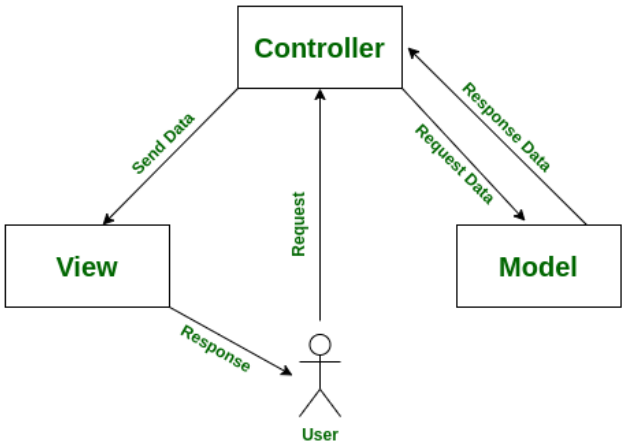


Рисунок 2.4 – Схематичне зображення роботи MVC

Модель (Model) відповідає за обробку та зберігання даних, включаючи роботу з базою даних. Представлення (View) відповідає за відображення даних та взаємодію з користувачем. Контролер (Controller) відповідає за обробку запитів користувача та керування роботою моделі та представлення.

Один з головних компонентів серверної частини веб-застосунка - це контролер, що відповідає за обробку запитів від клієнта та передачу відповідей. Контролер може бути реалізований у вигляді класу ASP.NET MVC, який містить декілька методів, що відповідають на різні запити.

Для забезпечення безпеки серверної частини веб-застосунка можна використовувати систему авторизації та аутентифікації. Наприклад, можна використовувати ASP.NET Identity, щоб дозволити користувачам створювати облікові записи, управляти ними та здійснювати авторизацію на сайті. Крім того, можна використовувати HTTPS для шифрування даних, що передаються між клієнтом та сервером.

2.4 Проектування клієнтської частини веб-застосунка

У проектуванні веб-застосунків клієнтська частина грає важливу роль, оскільки це та частина, яку будуть використовувати кінцеві користувачі. Клієнтська частина застосунку повинна бути зручною та простою у використанні, а також повинна мати ефективний та зрозумілий інтерфейс.

У веб-системі для розподілу та обліку завдань і контролю робочого часу IT-компанії, клієнтська частина буде розроблена з використанням HTML, CSS та JavaScript. Буде використано бібліотеку jQuery, яка є потужним інструментом для спрощення взаємодії з DOM структурою сторінки та виконання AJAX запитів до серверної частини застосунку. Це дозволить забезпечити зручну та ефективну роботу зі сторінкою, змінювати її вміст без перезавантаження сторінки та взаємодіяти з сервером в режимі реального часу.

Використання HTML, CSS та JavaScript дозволить створити привабливий та інтерактивний інтерфейс користувача. HTML забезпечуватиме структуру сторінки, CSS відповідатиме за її стилізацію, а JavaScript додасть динаміку та можливості взаємодії з користувачем. Комбінація цих технологій дозволить створити зручне та просте у використанні інтерфейсне рішення.

Однією з головних вимог до клієнтської частини застосунку є зручний та інтуїтивно зрозумілий інтерфейс користувача. Для цього під час проектування необхідно ретельно продумати структуру сторінок та їх зовнішній вигляд, а також розміщення елементів управління на сторінках.

Структура системи управління змістом має бути максимально зрозумілою та логічною. Для зручності користувачів рекомендується розробити зручнине меню навігації, яке буде містити всі необхідні пункти меню та дозволить користувачам швидко та легко знайти потрібну інформацію.

Окрім цього, важливим елементом клієнтської частини застосунку є логічна організація інформації на сторінках ресурсу. Користувач повинен легко знайти всю необхідну інформацію та мати змогу швидко перейти до інших сторінок, що стосуються даного функціоналу.

Для візуального представлення структури системи управління змістом та логічної організації інформації на сторінках можуть використовуватись різноманітні діаграми, блок-схеми та інші графічні елементи.

Важливо, щоб вони були з відповідними віджетами та елементами, які забезпечують зручну та логічну навігацію для користувача. Наприклад, меню повинне бути зручним у використанні та легко знаходити необхідні функції. Також важливо, щоб інтерфейс був адаптивним та пристосовувався до різних пристроїв, таких як комп'ютери, планшети та мобільні телефони.

2.5 Створення макета веб-застосунка та дизайн

Гарний дизайн у веб-системі для менеджменту проектів є дуже важливим, оскільки він може впливати на користувацький досвід та продуктивність роботи з системою. Якщо дизайн буде недбало розробленим, то користувач може мати труднощі з знаходженням необхідної інформації та виконанням завдань. Навпаки, гарний дизайн, та продумане розташування елементів зробить роботу з веб-системою більш зручною та ефективною.

Дизайн повинен бути привабливим та сприяти зручності взаємодії з застосунком, відображати ідентичність компанії та передавати інформацію зрозуміло та лаконічно.

Перед створенням макета було розглянуто потреби та функціонал кожної з ролей, які будуть користуватися веб-застосунком. На основі цього було розроблено окремі макети для кожної ролі: адміністратора, менеджера проекту та розробника. Макети були створені з урахуванням основних потреб та функціональних можливостей кожної ролі.

Основна мета такого Макету - це узгодити загальний дизайн та вигляд сторінок до початку розробки та імплементації функцій.

Простий шаблон дозволяє візуалізувати загальний вигляд застосунка та окремих його елементів, таких як меню, кнопки, форми та інші. Він також дозволяє визначити загальний стиль та колірну схему застосунка, що забезпечує більш єдинообразний та привабливий вигляд для користувачів.

Крім того, створення шаблону дозволяє зекономити час та зусилля на наступних етапах розробки, оскільки він дає змогу попередньо визначити всі елементи, що необхідні для веб-застосунка та їх розміщення на сторінках.

Макет головної сторінки адміністратора включає в себе список проектів, які можна переглядати та редагувати, а також кнопки для додавання нового проєкту. В лівій частині сторінки розміщено меню навігації, що дозволяє швидко перейти до інших розділів системи. На макеті було використано простий та

зрозумілий для користувача дизайн, з використанням лаконічних елементів та чіткого поділу на блоки. Макет був створений як шаблон який буде використовуватись для позиціонування та більш чіткого уявлення про росташування елементів сторінки. Шаблон головної сторінки адміністратора можна побачити на рисунку 2.5.

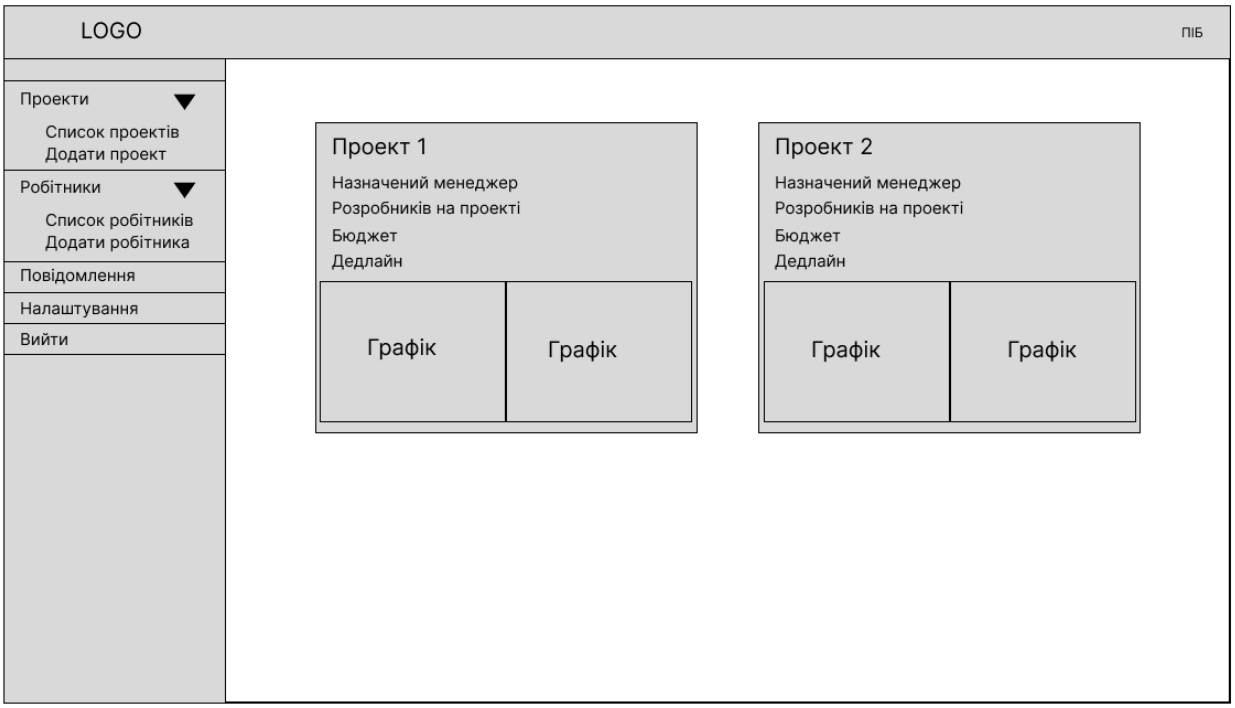


Рисунок 2.5 – Шаблон головної сторінки адміністратора

Макет головної сторінки менеджера проекту включає загальну інформацію про проєкт, за який він відповідає. На цій сторінці менеджер зможе отримати доступ до основних даних про проєкт, його стан та прогрес виконання. Макет дизайну сторінки враховує потреби менеджера проекту у зручному та швидкому доступі до інформації.

Головна сторінка менеджера проекту містить заголовок, що вказує на назву проєкту, а також загальний огляд стану проєкту. Інформація про прогрес виконання проєкту, кількість завдань, що виконані та не виконані, а також відсоток завершеності проєкту, відображаються на цій сторінці. Крім того, менеджер може переглянути список учасників проєкту та їхні ролі.

Дизайн сторінки орієнтований на простоту та зрозумілість інформації. Використання відповідних кольорів, шрифтів та іконок допомагає виділити основну інформацію та полегшує сприйняття даних. Шаблон зображено на рисунку 2.6.

Рисунок 2.6 – Шаблон головної сторінки менеджера

На макеті головної сторінки розробника відображається перелік завдань, призначених для виконання. Цей перелік містить докладну інформацію про кожне завдання, включаючи назву, опис та термін виконання. Кожен елемент списку надає користувачеві зручну переглядову форму для швидкого огляду важливих деталей кожного завдання.

Ця функціональність дозволяє розробникам легко отримувати доступ до переліку своїх завдань та отримувати необхідну інформацію для їх виконання. Завдяки цьому, розробники можуть ефективно організувати свою роботу, визначати пріоритети та керувати часом для досягнення поставлених цілей. Шаблон зображено на рисунку 2.7.

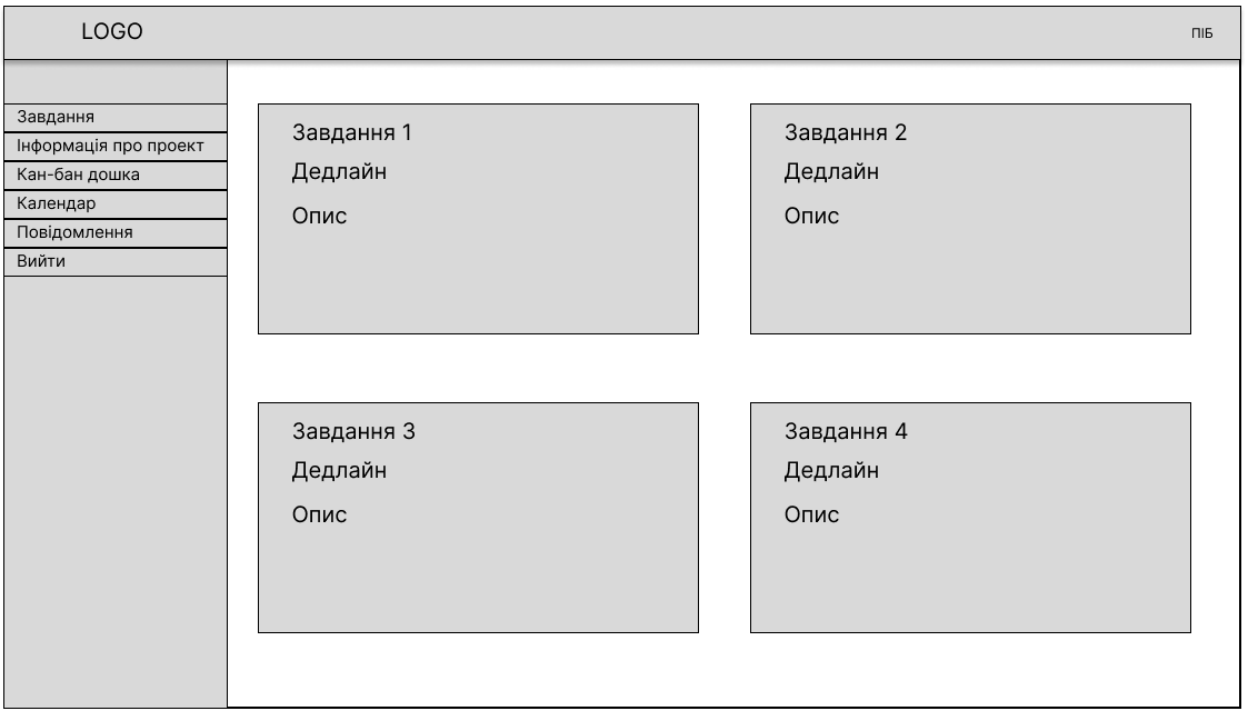


Рисунок 2.7 – Шаблон головної сторінки розробника

2.6 Аналіз та вибір технологій і методів реалізації веб-застосунка

Веб-система буде реалізована за допомогою таких технологій, як ASP.NET Core, HTML, CSS та JavaScript. ASP.NET Core є відкритою платформою, спеціально розробленою для створення веб-застосунків, що надає розробникам можливість творити високоякісні веб-застосунки на різних платформах. Вона пропонує численні переваги, включаючи високу швидкість роботи, високий рівень безпеки, легкість розгортання та масштабованість.

ASP.NET Core забезпечує ефективну обробку запитів, що дозволяє веб-застосунку працювати швидко та ефективно, що є важливим аспектом для задоволення потреб користувачів. Безпека є однією з важливих характеристик ASP.NET Core, і вона надає розробникам набір інструментів для захисту веб-застосунків від потенційних загроз. Крім того, фреймворк забезпечує легкість

розгортання веб-застосунків, що дозволяє швидко виводити їх в експлуатацію. Завдяки вбудованим можливостям масштабування, ASP.NET Core дозволяє веб-системі зростати разом з розширенням обсягу роботи і кількості користувачів.

HTML та CSS є фундаментальними складовими будь-якого веб-застосунка і використовуються для створення структури та зовнішнього вигляду веб-сторінок. Вони надають можливість розміщувати та оформлювати різноманітний вміст, включаючи текст, зображення, форми та інші елементи, що сприяють інтерактивності та зручності використання веб-застосунка.

HTML та CSS мають простий синтаксис, що спрощує їх використання для розробки веб-застосунків різної складності. Вони є стандартними технологіями для веб-розробки і є підтримуваними всіма сучасними веб-браузерами, що забезпечує їхню універсальність та сумісність з різними пристроями, платформами та браузерами.

JavaScript є однією з найпопулярніших мов програмування, яка широко використовується розробниками для створення веб-застосунків. Він має широкий функціонал і дозволяє розробникам створювати динамічні та інтерактивні веб-сторінки, що поліпшують взаємодію користувача з застосунком.

JavaScript забезпечує можливості для маніпуляції вмістом веб-сторінок, зміни стилів, обробки подій, валідації даних та взаємодії з сервером. Ця мова програмування дозволяє створювати функціональність, що реагує на дії користувача, наприклад, валідувати форми перед відправкою, оновлювати вміст сторінки без перезавантаження і відправляти асинхронні запити на сервер.

Окрім вбудованих можливостей JavaScript, існує також широкий вибір відкритих бібліотек і фреймворків, що розширюють можливості мови. Наприклад, бібліотеки React, Angular і Vue дозволяють розробникам будувати складні веб-застосунки з використанням компонентного підходу та повторного використання коду.

Також у майбутньому веб-застосунку передбачається використання різних JavaScript фреймворків, спеціально призначених для малювання графіків та

візуалізації даних. Ці фреймворки надають потужні інструменти для створення привабливих та інтерактивних графіків, що допомагають відображати інформацію в зрозумілій та ефективній формі.

Фреймворки, такі як D3.js, Chart.js, або Plotly.js, надають широкі можливості для створення різних типів графіків, діаграм, інтерактивних візуалізацій даних та географічних карт. Вони дозволяють зручно працювати з даними, налаштовувати зовнішній вигляд графіків та забезпечувати можливості взаємодії з ними, такі як масштабування, перетягування, анімація тощо.

2.6 Висновки розділу проектування програмного забезпечення

У розділі проектування програмного забезпечення був проведений аналіз різних архітектур веб-застосунків, зокрема клієнт-серверної зі статичною генерацією контенту, динамічною генерацією та single page. В результаті проведеного аналізу було обрано клієнт-серверну архітектуру з динамічною генерацією сторінок та обґрунтовано цей вибір.

Динамічна генерація веб-сторінок дозволяє формувати контент на сервері відповідно до запитів користувача і передавати його на клієнт. Це забезпечує більш гнучкий та динамічний підхід до розробки, оскільки сервер може динамічно формувати сторінки залежно від стану системи, даних користувача та виконувати різноманітні операції.

У розділі опис структури даних та моделі бази даних було проведено детальне дослідження та розробку структури даних, необхідних для веб-застосунка. Це включало визначення таблиць, зв'язків та атрибутів, необхідних для зберігання даних та забезпечення їх цілісності.

Була створена модель бази даних у формі ER-діаграми, що дозволяє візуально представити структуру бази даних та зв'язки між сутностями. В результаті проведеного аналізу було визначено оптимальну структуру даних для

потреб веб-застосунка, а також встановлено необхідні зв'язки та атрибути для кожної таблиці бази даних.

Під час проектування серверної частини веб-застосунка було обрано архітектурний патерн MVC (Model-View-Controller). Цей патерн дозволяє ефективно розділити відповідальності між компонентами системи, забезпечуючи модульність, зручність розробки та масштабованість.

Під час проектування клієнтської частини веб-застосунка було звернуто увагу на створення зручного та естетичного інтерфейсу користувача. Дизайн та макет веб-застосунка були розроблені з урахуванням потреб користувачів.

Аналіз та вибір технологій і методів реалізації веб-застосунка був проведений з огляду на досягнення поставлених цілей проекту. Були вибрані технології і методи, які найкраще відповідають вимогам проекту та забезпечують ефективну реалізацію функціональності.

3 Програмна реалізація

3.1 Розробка бази даних

Початковим кроком в розробці бази даних веб-застосунку було визначення сутностей або таблиць, які необхідно було зберігати для ефективного функціонування системи. За допомогою Entity Framework, потужного інструменту для роботи з базами даних, було створено моделі даних, що точно відображали структуру цих сутностей та їх взаємозв'язки.

Під час визначення структури бази даних, було використано різні типи відношень між таблицями, зокрема один-до-одного, один-до-багатьох та багато-до-багатьох. Ці відношення дозволили встановити зв'язки між сутностями та визначити, як вони будуть взаємодіяти між собою.

Один з ключових аспектів проектування бази даних було забезпечення цілісності даних. Було використано відповідні обмеження та правила для забезпечення правильного збереження та маніпуляції даними в межах бази даних. Наприклад, були встановлені обмеження на значення полів, контроль унікальності або обов'язковість заповнення певних полів.

Використання Entity Framework у сполученні з правильним визначенням сутностей та відношень між ними дозволило створити стійку та ефективну базу даних для веб-застосунку. Завдяки цьому, система зберігання даних відповідала вимогам проекту та забезпечувала надійність та швидкодію при роботі.

Після визначення структури бази даних, було здійснено процес міграції, що дозволило автоматично створити необхідні таблиці та налаштувати структуру бази даних за допомогою міграційних скриптів. Цей підхід спрощує управління структурою бази даних та дозволяє легко вносити зміни в майбутньому, необхідні для розширення або покращення функціональності.

Під час міграції, було використано інструменти Entity Framework для створення міграційних скриптів, які містили необхідні команди для створення таблиць, відношень та інших структурних елементів бази даних. Після цього,

було виконано міграцію, що запустила виконання цих скриптів та створила фактичну базу даних з необхідною структурою.

Міграційні скрипти також дозволяють контролювати зміни в базі даних і вносити зміни безпечним та контрольованим способом. Наприклад, якщо виникає потреба додати нову сутність або змінити вже існуючу, можна створити відповідний міграційний скрипт, який буде автоматично застосований до бази даних при запуску процесу міграції.

Такий підхід спрощує управління базою даних та забезпечує її гнучкість для внесення змін у майбутньому.

Індексація поля в базі даних дозволяє створити спеціальну структуру даних, яка прискорює пошук та фільтрацію записів за цим полем. Ідентифіковано ті поля, за якими найчастіше відбувається пошук або фільтрація даних, створено відповідні індекси для цих полів. Це дозволило значно зменшити час виконання запитів, оскільки система може швидко локалізувати та доступатися до відповідних записів за значенням індексованого поля.

Після створення індексів, було перевірено та оптимізовано запити, що використовуються у застосунку. Проаналізовано структуру запитів та розглядали можливості використання індексів для покращення продуктивності. При необхідності, можна ввести зміни до запитів, щоб вони ефективно використовували індекси та максимально скорочували час виконання.

Використання індексів та оптимізація запитів в базі даних дозволили досягти покращення продуктивності та ефективності роботи з даними. Швидкий доступ до інформації та скорочення часу виконання запитів сприяє покращенню продуктивності та користувацького досвіду веб-застосунку.

Під час розробки бази даних для веб-застосунку, було враховано потребу валідації та обмеження даних. Було встановлено правила перевірки даних, щоб гарантувати їх коректність та цілісність перед збереженням у базі даних. Це було особливо важливо для забезпечення правильності збереженя даних.

Під час визначення правил перевірки даних, було враховано специфіку кожного поля та його припустимі значення. Налаштовано перевірки на типи даних, щоб переконатися, що введені значення відповідають очікуваному формату та валідним критеріям.

Крім того, встановлено обмеження щодо допустимих значень, довжини полів та зв'язків між сутностями. Наприклад, для числових полів були встановлені мінімальні та максимальні значення, щоб обмежити діапазон введених даних. Для текстових полів були встановлені обмеження довжини, щоб уникнути перевищення розміру поля та забезпечити оптимальну продуктивність бази даних.

Застосування правил валідації та обмежень даних гарантує, що лише коректні та допустимі дані будуть збережені в базі даних. Це сприяє якості та надійності інформації, що зберігається, та допомагає уникнути можливих проблем, пов'язаних з некоректними

Всі вищезазначені кроки, дозволили їм успішно реалізувати базу даних для веб-застосунку. Ця база даних забезпечує ефективне зберігання, доступ та маніпуляцію даними. Використання Entity Framework виявилося ключовим фактором, що спростив процес роботи з базою даних та забезпечив зручну розробку та підтримку системи зберігання даних.

Entity Framework надав потужний інструментарій для взаємодії з базою даних, адаптований до вимог проекту. Використання моделей даних спростило процес створення таблиць та відображення зв'язків між ними. Можливість автоматичної генерації таблиць та структури бази даних за допомогою міграційних скриптів забезпечила зручну маніпуляцію базою даних та легкість у внесенні змін.

3.2 Розробка програмних модулів

Процес розробки програмних модулів для застосунка, що ґрунтується на технології ASP.Net Core MVC, був здійснений відповідно до принципів цієї архітектури. Зокрема, була використана можливість розділення застосунка на окремі компоненти - модель, представлення та контролер. Цей підхід дозволяє виконувати конкретні функції в межах кожного модуля та призначати їм відповідальність у системі.

Кожен програмний модуль мав свою власну функціональність та виконував певні завдання у контексті застосунка. Модуль моделі відповідав за управління даними та взаємодію з базою даних. Він забезпечував доступ до даних, їх збереження та обробку. Модуль представлення відповідав за відображення даних та взаємодію з користувачем через веб-інтерфейс. Він відповідав за створення та відображення сторінок, форм та інших елементів інтерфейсу. Контролери були відповідальні за обробку запитів користувача, виконання відповідних дій та взаємодію з іншими модулями.

Розробка програмних модулів передбачала створення інтерфейсів та методів, які забезпечували взаємодію між модулями та передачу необхідних даних. Кожен модуль був проектований з урахуванням його функціональних обов'язків та взаємозв'язку з іншими модулями, що дозволило досягти високого рівня модульності та перевикористання коду.

Було використано API та функціонал ASP.Net Core MVC для реалізації програмних модулів. Використано доступні класи та методи, щоб забезпечити необхідну функціональність та взаємодію між модулями. Крім того, було враховувано принципи доброї практики програмування, такі як розділення відповідальностей, забезпечення чистоти коду.

У процесі розробки програмних модулів було створено моделі даних, які відображали структуру та зв'язки між різними сутностями. Ці моделі використовувалися для доступу до бази даних та проведення операцій

збереження та маніпуляції даними. Розробка моделей проводилась з урахуванням специфічних потреб застосунку та вимог проекту.

Розробка моделей даних включала в себе визначення структури таблиць та їхніх атрибутів, а також встановлення зв'язків між різними сутностями. Було створено класи, які відображали окремі таблиці бази даних, а також використовували анотації та конфігураційні файли для визначення правил і зв'язків. При розробці моделей даних були враховані особливості домену застосунку, його логіки та вимог до даних.

Створені моделі даних дозволили зручно та ефективно взаємодіяти з базою даних, забезпечуючи доступ до неї та забезпечення цілісності даних. Вони надавали зручні методи та властивості для отримання, зміни та збереження даних, а також відображення взаємозв'язків між різними таблицями.

Завдяки ретельній розробці моделей даних, було забезпечено стабільну та ефективну роботу програмних модулів, що дозволило застосунку успішно функціонувати та задовольняти вимоги проекту.

При розробці програмних модулів була створена модель користувача, яка відображає структуру та характеристики користувачів у системі. Нижче на рисунку 3.1 представлено фрагмент коду моделі користувача.

```
public class ApplicationUser : IdentityUser
{
    [PersonalData]
    [Column(TypeName = "nvarchar(100)")]
    public string Name { get; set; }

    [PersonalData]
    [Column(TypeName = "nvarchar(100)")]
    public string Surname { get; set; }

    [PersonalData]
    [Column(TypeName = "nvarchar(100)")]
    public string SecondName { get; set; }
    public virtual ICollection<LoginHistory> LoginHistories { get; set; }

    public virtual ICollection<Task> Tasks { get; set; }
}
```

Рисунок 3.1 – Фрагмент коду моделі користувача.

Нижче на рисунку 3.2 представлений фрагмент коду моделі клієнта, який відображає структуру та характеристики цієї моделі.

```
public class Client
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(50)]
    public string Name { get; set; }

    [MaxLength(255)]
    public string ContactInfo { get; set; }

    public virtual ICollection<Project> Projects { get; set; }
}
```

Рисунок 3.2 – Фрагмент коду моделі клієнта.

Нижче на рисунку 3.3 представлений фрагмент коду моделі проекту.

```
public class Project
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(50)]
    public string Name { get; set; }

    [MaxLength(255)]
    public string Description { get; set; }

    [Required]
    public int ClientId { get; set; }

    public virtual Client { get; set; }

    public virtual ICollection<Task> Tasks { get; set; }
}
```

Рисунок 3.3 – Фрагмент коду моделі проекту.

Представлені фрагменти коду моделей, демонструють реалізацію конкретної частини системи. Розгляд цих фрагментів коду надає детальнішу інформацію про структуру та функціональні можливості системи.

Ці фрагменти коду демонструють реалізацію конкретної частини системи та надають детальну інформацію про її структуру та функціональні можливості. Вони допомагають краще зрозуміти, як взаємодіють різні компоненти системи та як вони виконують свої завдання. Аналіз цих фрагментів коду дозволяє здобути глибше розуміння системи та її потенціалу.

Нижче на рисунку 3.4 представлений фрагмент коду моделі завдання.

```
public class Task
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(255)]
    public string Description { get; set; }

    [Required]
    public DateTime CreatedAt { get; set; }

    public DateTime? Deadline { get; set; }

    public virtual ICollection<ApplicationUser> AssignedUser { get; set; }

    [Required]
    public int ProjectId { get; set; }

    public virtual Project { get; set; }
}
```

Рисунок 3.3 – Фрагмент коду моделі завдання.

Нижче на рисунку 3.4 представлений фрагмент коду моделі повідомлення.

```
public class Message
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(255)]
    public string Content { get; set; }

    [Required]
    public DateTime SentAt { get; set; }

    public virtual ApplicationUser Sender { get; set; }

    public virtual ApplicationUser Reciever { get; set; }
}
```

Рисунок 3.4 – Фрагмент коду моделі повідомлення.

У процесі розробки програмних модулів було розроблено представлення, які відповідали за візуальне представлення даних користувачам. Ці представлення включали в себе шаблони сторінок, де були визначені компоненти та елементи управління, а також механізми відображення результатів запитів користувача. Програмні модулі використовували розширення ASP.Net Core MVC, яке спрощувало роботу з представленнями та забезпечувало їх взаємодію з контролерами.

Одним з ключових аспектів розробки було використання Identity - фреймворку для автентифікації та авторизації користувачів. З його допомогою було забезпечено реєстрацію користувачів, управління їхніми обліковими записами та визначення рівнів доступу до функціональності застосунку. Identity надавав готові компоненти, які можна було легко інтегрувати у програмні модулі та представлення, що значно спрощувало розробку системи автентифікації та авторизації.

Завдяки розробці представлень та використанню bootstrap, програмні модулі надавали зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

У результаті розробки представлень були створені всі необхідні сторінки, включаючи головні сторінки для всіх ролей, які відповідали вимогам проекту. Було використано попередньо розроблені макети, щоб забезпечити єдність в дизайні та вигляді застосунка. Результатом розробки представлень було створення зручного та привабливого інтерфейсу, який дозволяв користувачам легко навігуватись по сторінках, виконувати необхідні дії та отримувати необхідну інформацію. Головну сторінку адміністратора зображено на рисунку 3.5

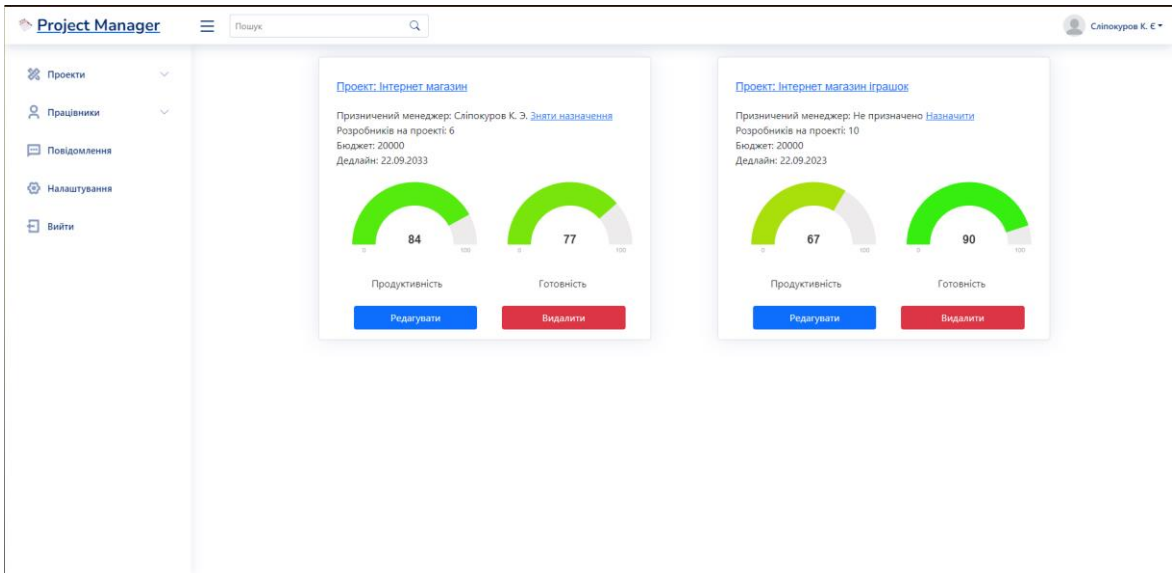


Рисунок 3.5 – Головна сторінка адміністратора

Головну сторінку менеджера зображено на рисунку 3.6

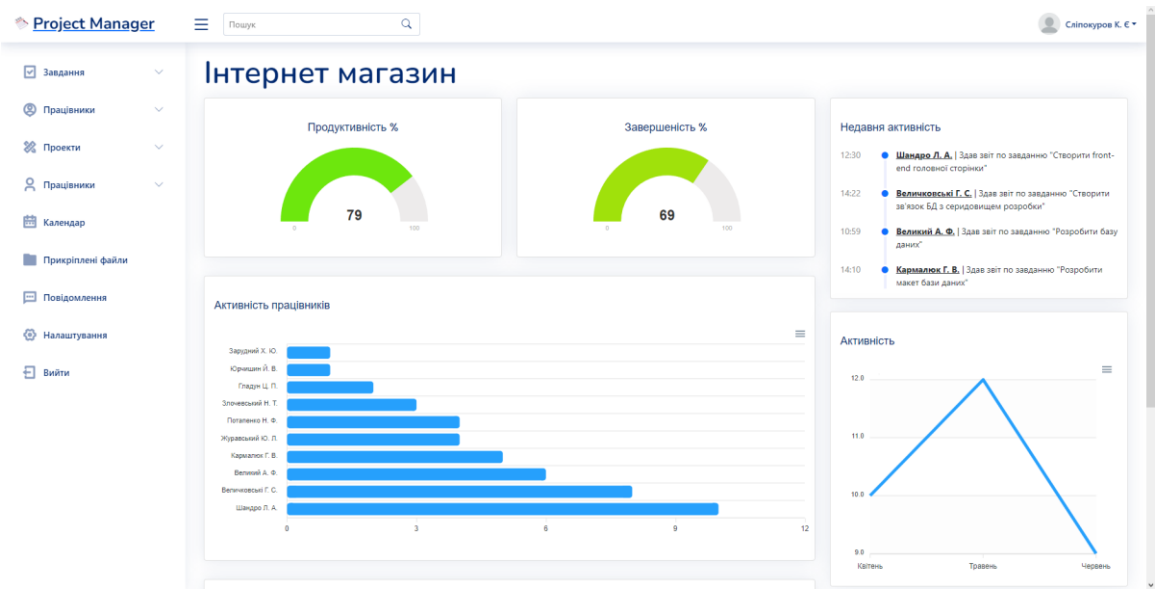


Рисунок 3.6 – Головна сторінка менеджера

Головну сторінку розробника зображено на рисунку 3.7

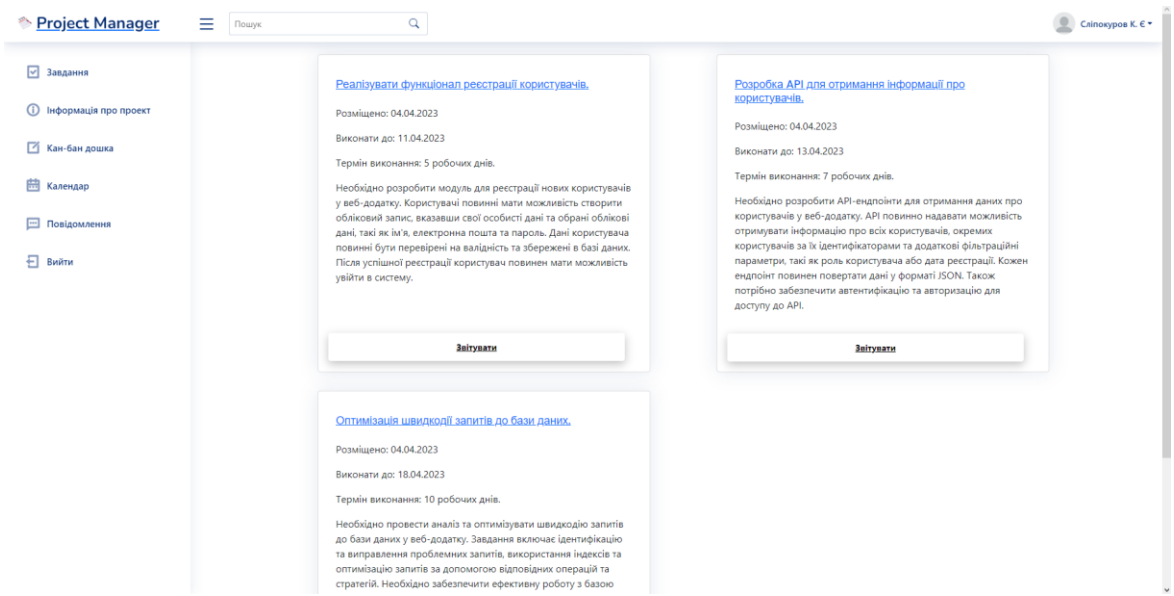


Рисунок 3.7 – Головна сторінка менеджера

Контролери відігравали важливу роль у розробці програмних модулів, відповідаючи за обробку запитів користувача та керування взаємодією між

моделями та представленнями. Вони приймають запити від користувача і виконували необхідні дії для обробки цих запитів.

Контролери відповідальні за обробку різних подій та дій користувача, таких як натискання кнопок, введення даних, навігація між сторінками тощо. Вони забезпечували передачу цих дій до моделей, які відповідали за обробку даних та виконання необхідних операцій. Контролери також оновлювали представлення, щоб відображати зміни, які сталися після виконання дій користувача або обробки даних.

Нижче наведено приклади коду, який було написано на вище описаному етапі. На рисунку 3.8 зображено код для додавання нового проекту.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>Create([Bind("Id,Name,Description,Budget,DeadLine")] Project project)
{
    if(ModelState.IsValid)
    {
        _context.Add(project);
        await _context.SaveChangesAsync();
        return RedirectToAction("/Projects/List");
    }
    return View(project);
}
```

Рисунок 3.8 – Код для створення нового проекту.

Код який відповідає за авторизацію зображено на рисунку 3.9.

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(model.Login, model.Password, model.RememberMe, false);
        if (result.Succeeded)
        {
            return RedirectToLocal("/Home/Index");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Помилка при вході");
            return View(model);
        }
    }
    return View(model);
}
```

Рисунок 3.9 – Код авторизації користувача.

На рисунку 3.10 зображено який відповідає за створення нового завдання.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Description,CreatedAt,Deadline")] Data.Task task)
{
    if (ModelState.IsValid)
    {
        Data.Task taskData = task;
        taskData.CreatedAt = DateTime.Now;
        _context.Add(taskData);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["ProjectId"] = new SelectList(_context.Projects, "Id", "Name", task.ProjectId);
    return View(task);
}
```

Рисунок 3.10 – Код для створення нового завдання.

Під час розробки програмних модулів використовувалася функціональність Identity, яка забезпечувала автентифікацію та авторизацію користувачів. Identity включав в себе компоненти, які були інтегровані в контролери для перевірки прав доступу та виконання інших пов'язаних з автентифікацією дій.

Таким чином, контролери взаємодіяли з моделями та представленнями, обробляючи запити користувача, передаючи дані до моделей для обробки та оновлюючи представлення відповідно до результатів операцій. Використання функціональності Identity дозволяло контролерам забезпечувати безпеку та управляти доступом користувачів до функціональності застосунка.

3.4 Технічні характеристики веб-застосунка

Застосунок сумісний з різними операційними системами, такими як Windows Server і Linux. Це дозволить вибрати оптимальну операційну систему залежно від вимог та вподобань. Для обробки запитів та надання веб-сторінок користувачам може використовуватися веб-сервер, такий як Internet Information Services (IIS) або Apache. Ці веб-сервери забезпечували стабільну та безперебійну роботу застосунка, здатну витримати великий потік запитів.

Ураховуючи вимоги до серверної частини, розроблення застосунка належним чином враховувало апаратне та програмне забезпечення. Під час вибору компонентів і налаштування інфраструктури, враховувалися потреби та масштаб проекту, щоб забезпечити ефективну та безперебійну роботу застосунка.

Щодо клієнтської частини застосунка, вимоги до комп'ютера користувача є меншими порівняно з серверною частиною. Застосунок був розроблений таким чином, щоб бути сумісним з різними операційними системами, включаючи Windows, macOS і Linux. Це дозволить користувачам вибирати платформу, яка найбільше відповідає їх потребам та вподобанням.

Для оптимального використання функціональності застосунка, рекомендується використовувати сучасні браузери, такі як Google Chrome, Mozilla Firefox або Microsoft Edge, оновлені до останніх версій. Це дозволить користувачам насолоджуватися підтримкою новітніх веб-технологій та забезпечувати оптимальну швидкість та зручність взаємодії з застосунком.

Застосування сумісності з різними операційними системами та сучасними браузерами максимально розповсюджує доступ до застосунка та забезпечує зручну роботу для широкого кола користувачів. Ураховуючи ці вимоги, розроблена клієнтська частина застосунка забезпечує сумісність із різними платформами та гарантує зручний та безперебійний досвід користувачів.

Це означає, що користувачі можуть використовувати застосунок на будь-якому комп'ютері або пристрої з операційною системою Windows, macOS або Linux. Ця гнучкість дає змогу користувачам вибрати платформу, яка найкраще відповідає їхнім потребам та уподобанням.

Крім того, використання сучасних браузерів, таких як Google Chrome, Mozilla Firefox або Microsoft Edge, в оновлених версіях, забезпечує оптимальну швидкість та зручність взаємодії з застосунком. Користувачі мають можливість насолоджуватися усіма функціональними можливостями застосунка та отримувати швидкі й безперебійні результати.

3.5 Тестування

У процесі ретельного огляду було розглянуто широкий спектр методів тестування, що включали модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування. Кожен із цих методів має свої унікальні перевагами та особливості, що вимагає об'єктивного відбору при виборі методів для проекту.

Модульне тестування, насамперед, може слугувати для перевірки ізольованих компонентів застосунка. Цей метод дозволяє провести докладний аналіз функціональності кожного модуля та переконатися в їх правильному функціонуванні. Інтеграційне тестування, у свою чергу, може зосереджуватися на перевірці взаємодії між різними компонентами та модулями застосунка.

Системне тестування, включає комплексну перевірку всього застосунка в цілому. Цей метод дозволяє переконатися, що всі компоненти застосунка працюють спільно і відповідають вимогам функціональності та продуктивності.

При виборі цих методів тестування були враховані їхні переваги та особливості, а також вимоги та характеристики розробленого веб-застосунка. Комбінація цих методів дозволила забезпечити широкий охоплення тестування та виявлення можливих проблем на всіх рівнях застосунка. В результаті цього обґрунтованого підходу була досягнута висока якість, надійність та функціональність розробленого веб-застосунка.

Після ретельного огляду було проведено детальний аналіз кожного методу тестування в контексті розробленого веб-застосунка. Модульне тестування використовувалось з метою перевірки окремих компонентів та функцій застосунка на відповідність специфікаціям та очікуваному результату. Цей підхід дозволив виявити потенційні помилки та проблеми в роботі окремих частин застосунка та вчасно їх виправити.

Щодо інтеграційного тестування, воно було використано для перевірки взаємодії між різними компонентами та модулями застосунка. Особлива увага

зверталася на взаємодію між моделями, представленнями та контролерами. Цей тип тестування дозволив виявити можливі проблеми, що можуть виникнути при інтеграції різних компонентів, та забезпечити правильну роботу застосунка.

Системне тестування було проведено для перевірки роботи всього застосунка як єдиного цілого. В рамках цього тестування перевірялися не лише функціональні аспекти, але й продуктивність та безпека застосунка. Різні сценарії використання були враховані для визначення реакції застосунка на різні умови та навантаження. Це дозволило виявити можливі проблеми та покращити ефективність та надійність застосунка.

Приймальне тестування було проведено з метою перевірки відповідності застосунка вимогам та очікуванням користувачів. Під час цього тестування користувачі, що не мали прямого зв'язку з розробкою, оцінювали функціональність, зручність використання та задоволеність від роботи з застосунком. Їх відгуки та пропозиції були враховані для подальшого вдосконалення застосунка та задоволення їх потреб.

Враховуючи характеристики та особливості розробленого веб-застосунка, обрані методи тестування були найбільш підходящими для забезпечення якості та надійності продукту. Кожен метод тестування мав свою роль у виявленні помилок та проблем на різних етапах розробки, дозволяючи забезпечити стабільну та функціональну роботу застосунка під час його використання.

Застосування комбінації різних методів забезпечило широкий спектр перевірок, що розпочиналися з тестування окремих компонентів та функцій, а закінчувалися інтеграційними та системними перевірками всього застосунка.

Крім вищезгаданих методів тестування, вибір включав використання різноманітних засобів, які значно полегшили та прискорили процес тестування. Один з таких засобів - автоматизоване тестування, що дозволило автоматизувати виконання тестових сценаріїв та перевірку результатів. Це знизило залежність від ручного виконання тестів і забезпечило більшу точність та однорідність у виконанні тестових процедур.

Для генерації тестових наборів даних були використані спеціальні інструменти, що дозволили швидко створювати різноманітні сценарії з реалістичними даними. Це дозволило випробувати застосунок у різних умовах та перевірити його поведінку на різних наборах даних, забезпечуючи більш глибоке тестування та виявлення потенційних проблем.

Для моніторингу продуктивності були використані спеціальні засоби, що дозволяли відстежувати роботу застосунка в реальному часі. Це дозволило виявити можливі проблеми з продуктивністю, швидко виявляти буттєві помилки та забезпечувати оптимальну продуктивність.

Засоби для виявлення та виправлення помилок були використані для швидкого виявлення, діагностики та усунення дефектів. Це включало в себе використання систем відстеження помилок, журналів подій та інших інструментів для аналізу та виправлення проблем. Такі засоби допомогли ефективно виявляти та виправляти помилки, забезпечуючи стабільність та надійність застосунка.

У процесі розробки веб-застосунка, використовувались тестові сценарії, що включали в себе широкий спектр тестових випадків. Ці сценарії були розроблені з метою перевірки різних аспектів функціонування застосунка.

Кожен тестовий сценарій був створений з урахуванням специфікацій та вимог до функціональності застосунка. Вони включали в себе вхідні дані, запуск відповідних функцій та перевірку очікуваного результату. Тестові сценарії дозволяли впевнитися в правильній роботі окремих компонентів та їх взаємодії, а також виявляти можливі помилки та недоліки.

Використовувались ці тестові сценарії під час розробки, вони запускались на регулярній основі для перевірки якості та коректності роботи застосунка. Виявлені помилки та недоліки були фіксовані, а тестові сценарії оновлювались залежно від змін у функціональності та вимогах.

Використання таких тестових сценаріїв значно полегшило процес тестування та допомогло виявити потенційні проблеми раніше, що сприяло

покращенню якості та надійності застосунка. Крім того, вони допомагали забезпечити сумісність різних компонентів та модулів, що було важливою передумовою для стабільної та безперебійної роботи застосунка.

Кожен unit-тест був ретельно спроектований з метою перевірки конкретної частини функціональності веб-застосунка. У цей процес входило передача вхідних даних до відповідних функцій та виклик необхідних методів, а також перевірка отриманого результату з очікуваним. Якщо отриманий результат не відповідав очікуваному, це свідчило про наявність помилки або недоліку в коді застосунка.

У кожному unit-тесті проводилися докладні перевірки, які допомагали виявляти різноманітні проблеми. Це включало перевірку правильності обробки різних вхідних сценаріїв та варіантів виконання функцій, а також перевірку коректності повернутих значень та стану системи після виконання певних дій. Крім того, проводилися перевірки на обробку виняткових ситуацій, забезпечуючи надійність та стійкість застосунка навіть у складних умовах.

Процес unit-тестування дозволяв ідентифікувати помилки та недоліки в ранніх стадіях розробки, що сприяло швидшому виправленню проблем та поліпшенню загальної якості застосунка. Також він підвищував розуміння роботи окремих компонентів та їх взаємодії, що сприяло подальшій оптимізації та вдосконаленню кодової бази.

Застосування unit-тестів у процесі розробки веб-застосунка було ключовим фактором у забезпеченні високої якості, стабільності та надійності продукту. Цей підхід допомагав виявляти помилки, забезпечувати коректну роботу компонентів та функцій, а також забезпечувати відповідну реакцію на різні сценарії використання.

Використання unit-тестів у розробці веб-застосунка було важливою практикою, яка допомагала виявляти помилки та недоліки в ранніх стадіях розробки. Це мав великий позитивний вплив на якість та надійність програмного

забезпечення, оскільки дозволяло швидко виявляти та згодом виправляти проблеми.

Завдяки unit-тестам, можна було перевіряти різні частини застосунка і переконуватися в їх коректній роботі. Це було особливо корисним на початкових етапах розробки, коли недоліки можуть бути виявлені та виправлені швидше та ефективніше.

Однією з головних переваг використання unit-тестів є зменшення ризиків виникнення непередбачених проблем під час розгортання та використання застосунка в реальних умовах. Це дозволяє уникнути ситуацій, коли недоліки стають очевидними або навіть критичними після введення в експлуатацію.

Тестові сценарії та їх результати стали об'єктивними доказами правильності роботи окремих компонентів, що сприяло збільшенню довіри до коду та полегшувало подальші модифікації й розширення.

У загальному контексті, використання unit-тестів було важливою практикою, яка допомагала забезпечити високу якість програмного забезпечення, зменшити ризики та поліпшити ефективність процесу розробки.

На рисунку 3.11 зображено фрагмент коду unit-тесту для тестування створення видалення та редагування проектів.

```
[TestClass]
public class UnitTestForProjectsController
{
    [TestMethod]
    public void TestMethodForCreation()
    {
        var project = new Project()
        {
            Name = "test",
            Description = "test",
            Budget = 10,
            DeadLine = DateTime.Now
        };
        DBContext _context = new DBContext();

        var controller = new ProjectsController(_context);
        var result = controller.Create(project) as ViewResult;
        Assert.AreEqual("List", result.ViewName);
    }

    [TestMethod]
    public void TestMethodForUpdate()
    {
        var project = new Project()
        {
            Name = "test",
            Description = "test",
            Budget = 10,
            DeadLine = DateTime.Now
        };
        DBContext _context = new DBContext();

        var controller = new ProjectsController(_context);
        var result = controller.Update(1, project) as ViewResult;
        Assert.AreEqual("List", result.ViewName);
    }

    [TestMethod]
    public void TestMethodForDelete()
    {
        DBContext _context = new DBContext();

        var controller = new ProjectsController(_context);
        var result = controller.Delete(1) as ViewResult;
        Assert.AreEqual("List", result.ViewName);
    }
}
```

Рисунок 3.11 – Фрагмент коду unit-тесту для тестування.

І на рисунку 3.12 зображено результат усіх unit-тестів які були створені.

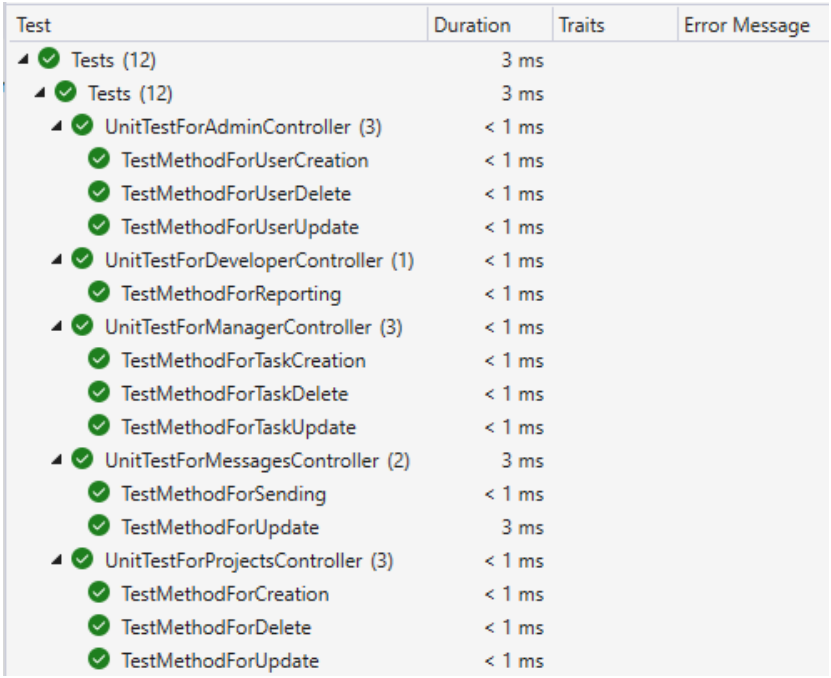


Рисунок 3.12 – Результати тестів

В результаті проведення тестування веб-застосунка було досягнуто кілька важливих висновків.

По-перше, використання різних методів тестування, таких як unit-тести, автоматизоване тестування, генерація тестових наборів даних та моніторинг продуктивності, дозволило виявити та виправити багато помилок та недоліків у веб-застосунку. Це сприяло забезпеченню його стабільності, надійності та високої якості.

По-друге, використання unit-тестів дозволило перевірити окремі компоненти та функції застосунка, забезпечуючи їх правильну роботу та виявлення можливих помилок. Це знизило ризик появи непередбачених проблем та сприяло ранньому виявленню та виправленню помилок у коді.

По-третє, використання автоматизованих засобів тестування та моніторингу продуктивності дозволило знизити зусилля та час, витрачений на тестування застосунку. Це дозволило швидше виявляти, а потім виправляти помилки, а також забезпечити оптимальну продуктивність та відповідність вимогам користувачів.

Узагальнюючи, тестування веб-застосунка було важливою складовою частиною його розробки. Використання різних методів тестування та засобів допомогло забезпечити високу якість, стабільність та надійність застосунка. Тестування сприяло виявленню та виправленню помилок у ранніх стадіях розробки, що дало змогу покращити його функціональність та задоволення потреб користувачів.

3.5 Аналіз результатів тестування

Після завершення процесу тестування веб-застосунка, було здійснено ретельний аналіз отриманих результатів з метою оцінки його функціональності, продуктивності, стабільності та безпеки. Цей аналіз був спрямований на ретельне дослідження різних аспектів застосунка, зокрема його можливостей, ефективності та безпечності.

Оцінка функціональності включала перевірку відповідності реалізованих функцій вимогам, визначеним у Технічному Завданні (ТЗ). Було здійснено перевірку, чи працюють всі функції застосунка так, як очікується, та чи задовольняють вони потреби користувачів.

Аналіз продуктивності застосунка передбачав вимірювання різних метрик, таких як час відгуку, завантаження ресурсів та швидкодія окремих операцій. Метою було переконатися, що застосунок працює швидко та ефективно, не має зайвих затримок та може витримувати навантаження.

При аналізі стабільності застосунка перевірялося, чи виникають помилки, крахи або несправності під час його використання. Була звернута увага на потенційні проблеми, що можуть впливати на стабільність застосунка, і розроблені відповідні заходи для їх виправлення.

Аналіз безпеки застосунка зосереджувався на перевірці потенційних вразливостей та забезпеченні адекватного рівня захисту. Було перевірено, чи

існують можливості несанкціонованого доступу, а також чи виконуються необхідні заходи для захисту конфіденційності та цілісності даних.

В процесі ручного тестування було проведено послідовне перевіряння різних функціональних аспектів застосунка згідно з визначеними тестовими сценаріями. Під час цього процесу виявлено ряд недоліків, таких як некоректна обробка вхідних даних або непередбачені ситуації, що призводили до некоректної роботи застосунка.

Автоматизовані тести, які були розроблені та виконані, допомогли здійснити швидку та повторювану перевірку різних функціональних модулів застосунка. Це дозволило виявити помилки, які можуть залишитися непоміченими під час ручного тестування, а також покращити швидкість і ефективність процесу тестування.

Для більш глибокого аналізу були використані спеціалізовані інструменти тестування, які дозволили автоматично перевірити певні аспекти застосунка, такі як безпекові проблеми, завантаження ресурсів або швидкодія. Ці інструменти надали інформацію про продуктивність та стабільність застосунка.

Всі виявлені помилки, недоліки та проблеми були ретельно задокументовані з вказівкою конкретного сценарію, умови виникнення та очікуваного результату. Це забезпечило систематизацію знайдених проблем та створило підґрунтя для їх подальшого вирішення.

Завдяки проведенню аналізу продуктивності, можна бути впевненим у високій продуктивності застосунка та його здатності ефективно виконувати поставлені завдання. Крім того, замовник має гарантію, що веб-застосунок здатний працювати з великою кількістю користувачів без втрати продуктивності, що підвищує його довіру та задоволення від використання програмного продукту.

Важливим аспектом проведеного аналізу була оцінка стабільності застосунка під час його використання. В процесі аналізу активно перевірялося,

чи виникають помилки, крахи або несправності, які можуть вплинути на його нормальну роботу.

Після виявлення потенційних вразливостей були прийняті відповідні заходи для забезпечення безпеки застосунка. Застосовувалися різні техніки, такі як шифрування даних, використання безпечних протоколів зв'язку, контроль доступу та перевірка даних на валідність. Були виконані необхідні зміни в коді та конфігурації застосунка для усунення виявлених вразливостей.

Для вирішення виявлених помилок та проблем було необхідно розробити та впровадити покращення в застосунок. Ці покращення могли включати оптимізацію коду, виправлення програмних помилок, впровадження додаткових заходів безпеки або поліпшення інтерфейсу користувача. Процес розробки та впровадження покращень проводився з метою забезпечення високої якості та надійності застосунка, а також задоволення потреб користувачів.

В цілому, аналіз результатів тестування веб-застосунка виявив недоліки та проблеми, які потребували уваги та вирішення. Впровадження покращень та усунення помилок дозволили забезпечити високу якість та надійність застосунка, що в свою чергу сприяло поліпшенню задоволення користувачів та підвищенню репутації компанії, яка стоїть за розробкою застосунка.

Завдяки аналізу результатів тестування, було забезпечено відповідність веб-застосунка визначеним вимогам. Усі функції та можливості, які були передбачені у Технічному Завданні (ТЗ), були перевірені та відповідали встановленим критеріям. Замовник мав впевненість, що його вимоги були виконані належним чином.

3.6 Висновки розділу програмна реалізація

Розробка бази даних була проведена відповідно до вимог проекту. При проектуванні були створені необхідні таблиці і встановлені взаємозв'язки між ними, що дозволяє ефективно зберігати та керувати даними проектів, завдань та

користувачів. База даних була заповнена тестовими даними для перевірки функціональності та правильності роботи системи. Під час розробки було враховано необхідну функціональність, забезпечуючи зручну роботу з даними і можливість виконувати необхідні операції, такі як створення, редагування та видалення проектів, завдань та інших об'єктів.

Контролери, моделі та представлення були реалізовані відповідно до вимог проекту. Була успішно реалізована необхідна функціональність системи, включаючи створення, редагування та видалення проектів, завдань, а також призначення проектних менеджерів на проекти та інші операції. Реалізовані компоненти програмного забезпечення забезпечують коректну та надійну роботу системи, дозволяючи користувачам здійснювати потрібні дії та взаємодіяти з даними.

Було забезпечено сумісність з різними системами, а також встановлені необхідні технічні характеристики сервера для оптимальної роботи застосунка.

Тестування веб-застосунка було проведено для перевірки його функціональності та надійності. В результаті тестування було виявлено та виправлено деякі помилки та проблеми. Застосунок пройшов випробування з різних сценаріїв використання та показав задовільні результати.

Аналіз результатів тестування підтвердив відповідність веб-застосунка вимогам та очікуванням. Функціональність та надійність системи були перевірені, і вона готова до подальшої експлуатації та використання.

ВИСНОВКИ

Завдяки веб-системі, компанія здатна здійснювати точний та прозорий розподіл завдань між працівниками, встановлювати приоритети та контролювати їх виконання. Це сприяє підвищенню продуктивності та зменшенню можливих перекриттів у завданнях. Крім того, система забезпечує точний облік робочого часу, дозволяючи компанії краще контролювати ресурси та витрати.

Власна веб-система також дозволяє компанії пристосувати її до своїх потреб і вимог. Розробка системи була здійснена з урахуванням специфіки діяльності компанії, її процесів та вимог до функціоналу. Це дозволило створити інструмент, який відповідає унікальним потребам компанії та сприяє її оптимальному функціонуванню.

Під час розробки було проведено детальний аналіз специфіки діяльності компанії. Детально вивчено основні процеси та потреби компанії у розробці програмних продуктів для замовників. В результаті, було отримано глибоке розуміння вимог та внутрішнього функціоналу, які є критичними для успішної роботи компанії.

З метою забезпечення високої придатності та зручного користування веб-системою, був проведений аналіз вимог до інтерфейсу та функціоналу серед працівників компанії.

Враховуючи аналіз специфіки діяльності ІТ-компанії, вимог до інтерфейсу та функціоналу серед працівників, а також узгодження з керівництвом, була створена веб-система, яка належним чином відповідає потребам та вимогам компанії. Результатом цього процесу є розробка ефективного та функціонального інструменту, який сприяє покращенню робочих процесів та контролю робочого часу в ІТ-компанії.

Розроблений веб-сервіс відповідає всім поставленим вимогам та ефективно виконує свої функції. Це стало можливим завдяки використанню передових моделей та алгоритмів, які забезпечують оптимальну роботу системи. Процес

валідації та верифікації підтвердив, що веб-сервіс працює бездоганно та відповідає вимогам до якості і надійності.

Після проведення детального аналізу результатів тестування, можна зробити висновок, що розроблена веб-система відповідає всім вимогам, що були встановлені у технічному завданні. Під час тестування було підтверджено, що система функціонує стабільно, не виникає помилок або відмов у роботі. Крім того, веб-система забезпечує необхідний рівень безпеки, захищаючи дані користувачів від несанкціонованого доступу. Вхідні та вихідні дані коректно оброблюються та відповідають заданим форматам, що гарантує правильну та ефективну роботу веб-застосунка. Ці висновки підтверджують високу якість та надійність розробленої веб-системи.

Отже, можна стверджувати, що мета кваліфікаційної роботи, яка полягала у розробці веб-системи для розподілу та обліку завдань і контролю робочого часу ІТ-компанії, була досягнута. Веб-система успішно розроблена, пройшла валідацію, верифікацію та тестування, підтвердивши свою відповідність встановленим вимогам. Вона забезпечує ефективний розподіл завдань, контроль робочого часу і облік виконаної роботи. Результати аналізу показали, що веб-система функціонує стабільно, безпечно та коректно обробляє дані. Таким чином, розроблена веб-система відповідає потребам ІТ-компанії та може використовуватись для покращення управління завданнями та контролю робочого процесу. Результати кваліфікаційної роботи підтверджують її успішне завершення і відповідність поставленим цілям та завданням.

ПЕРЕЛІК ДЖЕРЕЛ

1. ТОП 5 інструментів управління проектами у 2022 [Електронний ресурс] – Режим доступу: https://worksection.com/ua/blog/5-project-management-tools.html (дата звернення – 11.02.2023). – Назва з екрану.
2. 5 інструментів для проджект менеджерів [Електронний ресурс] – Режим доступу: https://happymonday.ua/5-instrumentiv-dlja-prodzhekt-menedzhera (дата звернення – 11.02.2023). – Назва з екрану.
3. 25 найкращих інструментів управління проектами у 2021 році [Електронний ресурс] – Режим доступу: https://uk.myservername.com/25-best-project-management-tools-2021 (дата звернення – 11.02.2023). – Назва з екрану.
4. Проектний менеджер (управління проектами) [Електронний ресурс] – Режим доступу: https://pererislyanska-gromada.gov.ua/proektnij-menedzhment-upravlinnya-proektami-1532185544/ (дата звернення – 12.02.2023). – Назва з екрану.
5. Виконання оцінки проекту на основі LOC- і FP- метрик [Електронний ресурс] – Режим доступу: http://um.co.ua/11/11-1/11-19333.html (дата звернення – 15.02.2023). – Назва з екрану.
6. Розмірно-орієнтовані метрики [Електронний ресурс] – Режим доступу: https://studfile.net/preview/8403848/page:2/ (дата звернення – 15.02.2023). – Назва з екрану.
7. Оцінка проекту на основі LOC- і FP- метрик [Електронний ресурс] – Режим доступу: https://studfile.net/preview/5200675/page:15/ (дата звернення – 15.02.2023). – Назва з екрану.
8. Функціонально-орієнтовані метрики [Електронний ресурс] – Режим доступу: http://www.ni.biz.ua/6/6_13/6_130000_funktsionalno-orientirovannie-metriki.html (дата звернення – 15.02.2023). – Назва з екрану.

9. Як обрати архітектуру для веб застосунку [Електронний ресурс] – Режим доступу: https://blog.ithillel.ua/articles/web-application-architecture (дата звернення – 09.03.2023). – Назва з екрану.

10. Архітектура программного забезпечення [Електронний ресурс] – Режим доступу: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення – 09.03.2023). – Назва з екрану.

11. Клієнт–серверна архітектура [Електронний ресурс] – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення – 20.03.2023) – Назва з екрану.

12. Клієнт–серверна архітектура [Електронний ресурс] – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення – 20.03.2023) – Назва з екрану.

13. Клієнт–серверна архітектура та ролі серверів [Електронний ресурс] – Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229 (дата звернення – 20.03.2023) – Назва з екрану.

14. Архітектура клієнт-сервер [Електронний ресурс] – Режим доступу: http://inter.ptngu.com/kompyuterni-merezhi/arhitektura-kliyent-server (дата звернення – 20.03.2023) – Назва з екрану.

15. Найважливіші архітектурні шаблони, які необхідно знати [Електронний ресурс] – Режим доступу: https://devzone.org.ua/post/nayvazhlivishi-arkhitekturni-shabloni-yakineobkhidno-znati (дата звернення - 23.03.2023) – Назва з екрану.

16. Welcome to Flask – Flask Documentation (2.2.x) [Електронний ресурс] – Режим доступу: https://flask.palletsprojects.com/en/2.2.x/ (дата звернення – 27.03.2023). – Назва з екрану.

17. Pehlivanov V. ASP.NET Core vs ASP.NET MVC: Which .NET Framework is better for You? / V. Pehlivanov [Електронний ресурс] – Режим доступу: https://www.resolutesoftware.com/blog/asp-net-mvc-vs-asp-net-core/ (дата звернення – 27.03.2023). – Назва з екрану.

18. Розробка інтерфейсів. Проектування графічного інтерфейсу користувача [Електронний ресурс] Режим доступу: https://uaeu.top/digital-online/rozrobka-interfejsiv-proektuvannya-grafichnogo-interfejsu-koristuvacha.html (дата звернення – 08.04.2023). – Назва з екрану.

19. Основні принципи створення інтерфейсу [Електронний ресурс] – Режим доступу: http://um.co.ua/8/8-10/8-109690.html (дата звернення – 05.04.2023). – Назва з екрану.

20. Overview of ASP.NET Core MVC [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk- ua/aspnet/core/mvc/overview?view=aspnetcore-7.0 (дата звернення – 17.04.2023). – Назва з екрану.

21. SP.NET Core Documentation [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0 (дата звернення – 18.04.2023). – Назва з екрану.

22. Ajax Frameworks [Електронний ресурс] – Режим доступу: https://www.xul.fr/ajax-frameworks.php (дата звернення – 18.04.2023). – Назва з екрану.

23. AJAX Introduction [Електронний ресурс] – Режим доступу: https://www.w3schools.com/xml/ajax_intro.asp (дата звернення – 19.04.2023). – Назва з екрану.

24. AJAX Framework [Електронний ресурс] – Режим доступу: https://wintercms.com/docs/ajax/introduction (дата звернення – 19.04.2023). – Назва з екрану.

25. Довідник по HTML тегам [Електронний ресурс] – Режим доступу: https://css.in.ua/html/tags (дата звернення – 20.04.2023). – Назва з екрану.

26. HTML підручник [Електронний ресурс] – Режим доступу: https://w3schoolsua.github.io/html/index.html#gsc.tab=0 (дата звернення – 20.04.2023). – Назва з екрану.

27. CSS підручник [Електронний ресурс] – Режим доступу: https://w3schoolsua.github.io/css/index.html#gsc.tab=0 (дата звернення – 20.04.2023). – Назва з екрану.

28. Javascript documentation [Електронний ресурс] – Режим доступу: https://devdocs.io/javascript/ (дата звернення – 20.04.2023). – Назва з екрану.

29.

30. Javascript documentation [Електронний ресурс] – Режим доступу: https://devdocs.io/javascript/ (дата звернення – 20.04.2023). – Назва з екрану.

31. JavaScript reference [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення – 20.04.2023). – Назва з екрану.

32. The Modern JavaScript Tutorial [Електронний ресурс] – Режим доступу: https://javascript.info (дата звернення – 20.04.2023). – Назва з екрану.

33. JavaScript and HTML DOM Reference [Електронний ресурс] – Режим доступу: https://www.w3schools.com/jsrEF/default.asp (дата звернення – 20.04.2023). – Назва з екрану.

34. Murugan M. Custom User Management in ASP.NET Core MVC with Identity / M. Murugan [Електронний ресурс] – Режим доступу: https://codewithmukesh.com/blog/user-management-in-aspnet-core-mvc/ (дата звернення – 21.04.2023). – Назва з екрану.

35. Use AJAX to Deliver Dynamic Updates [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/nerddinner/use-ajax-to-deliver-dynamic-updates (дата звернення – 25.04.2023). – Назва з екрану.

36. Unit Testing in ASP.NET Core MVC [Електронний ресурс] – Режим доступу: https://code-maze.com/unit-testing-in-asp-net-core-mvc/ (дата звернення – 25.04.2023). – Назва з екрану.

37. Entity Framework, With .Net Core MVC, Code-First [Електронний ресурс] – Режим доступу: https://www.c-sharpcorner.com/article/entity-framework-with-net-core-mvc-4-code-first/ (дата звернення – 23.04.2023). – Назва з екрану.

38. Огляд видів тестування [Електронний ресурс] – Режим доступу: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення – 02.05.2023). – Назва з екрану.

39. Що таке тестування програмного забезпечення [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/ (дата звернення – 02.05.2023). – Назва з екрану.

40. Види тестування програмного забезпечення [Електронний ресурс] – Режим доступу: https://sqa.lviv.ua/yaki-ye-typy-testuvannya (дата звернення – 02.05.2023). – Назва з екрану.

41. Функціональне тестування [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/ (дата звернення – 02.05.2023). – Назва з екрану.

42. Що таке функціональне тестування? Типи, приклади, контрольний список та впровадження [Електронний ресурс] – Режим доступу: https://www.zaptest.com/uk/що-таке-функціональне-тестування-тип (дата звернення – 02.05.2023). – Назва з екрану.

43. Що таке функціональне тестування? Типи, приклади, контрольний список та впровадження [Електронний ресурс] – Режим доступу: https://www.zaptest.com/uk/що-таке-функціональне-тестування-тип (дата звернення – 03.05.2023). – Назва з екрану.

44. Що таке функціональне тестування? Типи, приклади, контрольний список та впровадження [Електронний ресурс] – Режим доступу: https://www.zaptest.com/uk/що-таке-функціональне-тестування-тип (дата звернення – 03.05.2023). – Назва з екрану.

45. Нефункціональне тестування [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/nefunktsionalne-testuvannya/ (дата звернення – 03.05.2023). – Назва з екрану.

46. Структурне тестування [Електронний ресурс] – Режим доступу: https://www.wikiwand.com/uk/Структурне_тестування (дата звернення – 03.05.2023). – Назва з екрану.

ДОДАТОК А

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

А.1 Код контролера AdminController

```
using System;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using YourProject.Models;

namespace ProjectManager.Controllers
{
    public class AdminController : Controller
    {
        private readonly DbContext _dbContext;

        public AdminController(DbContext dbContext)
        {
            _dbContext = dbContext;
        }

        // Отримати список проектів
        public IActionResult Projects()
        {
            var projects = _dbContext.Projects.ToList();
            return View(projects);
        }

        // Створити новий проект
        [HttpGet]
        public IActionResult CreateProject()
        {
            return View();
        }

        [HttpPost]
```

```
public IActionResult CreateProject(Project project)
{
    if (ModelState.IsValid)
    {
        _dbContext.Projects.Add(project);
        _dbContext.SaveChanges();
        return RedirectToAction("Projects");
    }
    return View(project);
}

// Редагувати проект
[HttpGet]
public IActionResult EditProject(int id)
{
    var project = _dbContext.Projects.Find(id);
    if (project == null)
    {
        return NotFound();
    }
    return View(project);
}

[HttpPost]
public IActionResult EditProject(Project project)
{
    if (ModelState.IsValid)
    {
        _dbContext.Entry(project).State = EntityState.Modified;
        _dbContext.SaveChanges();
        return RedirectToAction("Projects");
    }
    return View(project);
}

// Видалити проект
[HttpPost]
public IActionResult DeleteProject(int id)
{
    var project = _dbContext.Projects.Find(id);
```

```
    if (project == null)
    {
        return NotFound();
    }
    _dbContext.Projects.Remove(project);
    _dbContext.SaveChanges();
    return RedirectToAction("Projects");
}

// Призначити PM на проект
[HttpGet]
public IActionResult AssignPM(int projectId)
{
    var project = _dbContext.Projects.Find(projectId);
    if (project == null)
    {
        return NotFound();
    }
    var employees = _dbContext.Employees.ToList();
    ViewBag.ProjectId = projectId;
    ViewBag.Employees = employees;
    return View();
}

[HttpPost]
public IActionResult AssignPM(ProjectManagerAssignment assignment)
{
    if (ModelState.IsValid)
    {
        _dbContext.ProjectManagerAssignments.Add(assignment);
        _dbContext.SaveChanges();
        return RedirectToAction("Projects");
    }
    var employees = _dbContext.Employees.ToList();
    ViewBag.ProjectId = assignment.ProjectId;
    ViewBag.Employees = employees;
    return View(assignment);
}

// Додати працівника
```

```
        [HttpGet]
        public IActionResult AddEmployee()
        {
            return View();
        }

        [HttpPost]
        public IActionResult AddEmployee(Employee employee)
        {
            if (ModelState.IsValid)
            {
                _dbContext.Employees.Add(employee);
                _dbContext.SaveChanges();
                return RedirectToAction("Employees");
            }
            return View(employee);
        }

        // Видалити працівника
        [HttpPost]
        public IActionResult DeleteEmployee(int id)
        {
            var employee = _dbContext.Employees.Find(id);
            if (employee == null)
            {
                return NotFound();
            }
            _dbContext.Employees.Remove(employee);
            _dbContext.SaveChanges();
            return RedirectToAction("Employees");
        }
    }
}
```

А.2 Код контролера ManagerController

```
using System;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.EntityFrameworkCore;
using YourProject.Models;

namespace ProjectManager.Controllers
{
    public class ProjectManagerController : Controller
    {
        private readonly DbContext _dbContext;

        public ProjectManagerController(DbContext dbContext)
        {
            _dbContext = dbContext;
        }

        // Редагувати проект
        [HttpGet]
        public IActionResult EditProject(int id)
        {
            var project = _dbContext.Projects.Find(id);
            if (project == null)
            {
                return NotFound();
            }
            return View(project);
        }

        [HttpPost]
        public IActionResult EditProject(Project project)
        {
            if (ModelState.IsValid)
            {
                _dbContext.Entry(project).State = EntityState.Modified;
                _dbContext.SaveChanges();
                return RedirectToAction("Projects");
            }
            return View(project);
        }

        // Додати завдання
        [HttpGet]
```

```
        public IActionResult AddTask(int projectId)
        {
            var project = _dbContext.Projects.Find(projectId);
            if (project == null)
            {
                return NotFound();
            }
            var developers = _dbContext.Developers.ToList();
            ViewBag.ProjectId = projectId;
            ViewBag.Developers = developers;
            return View();
        }

        [HttpPost]
        public IActionResult AddTask(Task task)
        {
            if (ModelState.IsValid)
            {
                _dbContext.Tasks.Add(task);
                _dbContext.SaveChanges();
                return RedirectToAction("ProjectTasks", new { projectId =
task.ProjectId });
            }
            var developers = _dbContext.Developers.ToList();
            ViewBag.ProjectId = task.ProjectId;
            ViewBag.Developers = developers;
            return View(task);
        }

        // Редагувати завдання
        [HttpGet]
        public IActionResult EditTask(int id)
        {
            var task = _dbContext.Tasks.Find(id);
            if (task == null)
            {
                return NotFound();
            }
            var developers = _dbContext.Developers.ToList();
            ViewBag.Developers = developers;
```

```
            return View(task);
        }

        [HttpPost]
        public IActionResult EditTask(Task task)
        {
            if (ModelState.IsValid)
            {
                _dbContext.Entry(task).State = EntityState.Modified;
                _dbContext.SaveChanges();
                return RedirectToAction("ProjectTasks", new { projectId =
task.ProjectId });
            }
            var developers = _dbContext.Developers.ToList();
            ViewBag.Developers = developers;
            return View(task);
        }

        // Видалити завдання
        [HttpPost]
        public IActionResult DeleteTask(int id)
        {
            var task = _dbContext.Tasks.Find(id);
            if (task == null)
            {
                return NotFound();
            }
            _dbContext.Tasks.Remove(task);
            _dbContext.SaveChanges();
            return RedirectToAction("ProjectTasks", new { projectId =
task.ProjectId });
        }

        // Призначити розробника на завдання
        [HttpGet]
        public IActionResult AssignDeveloper(int taskId)
        {
            var task = _dbContext.Tasks.Find(taskId);
            if (task == null)
            {
```

```
                    return NotFound();
                }
                var developers = _dbContext.Developers.ToList();
                ViewBag.TaskId = taskId;
                ViewBag.Developers = developers;
                return View();
            }

            [HttpPost]
            public IActionResult AssignDeveloper(TaskAssignment assignment)
            {
                if (ModelState.IsValid)
                {
                    _dbContext.TaskAssignments.Add(assignment);
                    _dbContext.SaveChanges();
                    return RedirectToAction("ProjectTasks", new { projectId =
assignment.Task.ProjectId });
                }
                var developers = _dbContext.Developers.ToList();
                ViewBag.TaskId = assignment.TaskId;
                ViewBag.Developers = developers;
                return View(assignment);
            }
        }
    }
```

А.2 Код контролера DeveloperController

```
using System;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using YourProject.Models;

namespace YourProject.Controllers
{
    public class DeveloperController : Controller
    {
        private readonly YourDbContext _dbContext;
```

```
        public DeveloperController(YourDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        // Головна сторінка розробника
        public IActionResult Index()
        {
            var tasks = _dbContext.Tasks
                .Where(t => t.DeveloperId == GetCurrentDeveloperId())
                .ToList();
            return View(tasks);
        }

        // Звіт про виконання завдання
        [HttpGet]
        public IActionResult ReportTask(int id)
        {
            var task = _dbContext.Tasks.Find(id);
            if (task == null)
            {
                return NotFound();
            }
            return View(task);
        }

        [HttpPost]
        public IActionResult ReportTask(Task task)
        {
            if (ModelState.IsValid)
            {
                _dbContext.Entry(task).State = EntityState.Modified;
                _dbContext.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(task);
        }
}
```

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Рисунок Б.1 – Слайд 1

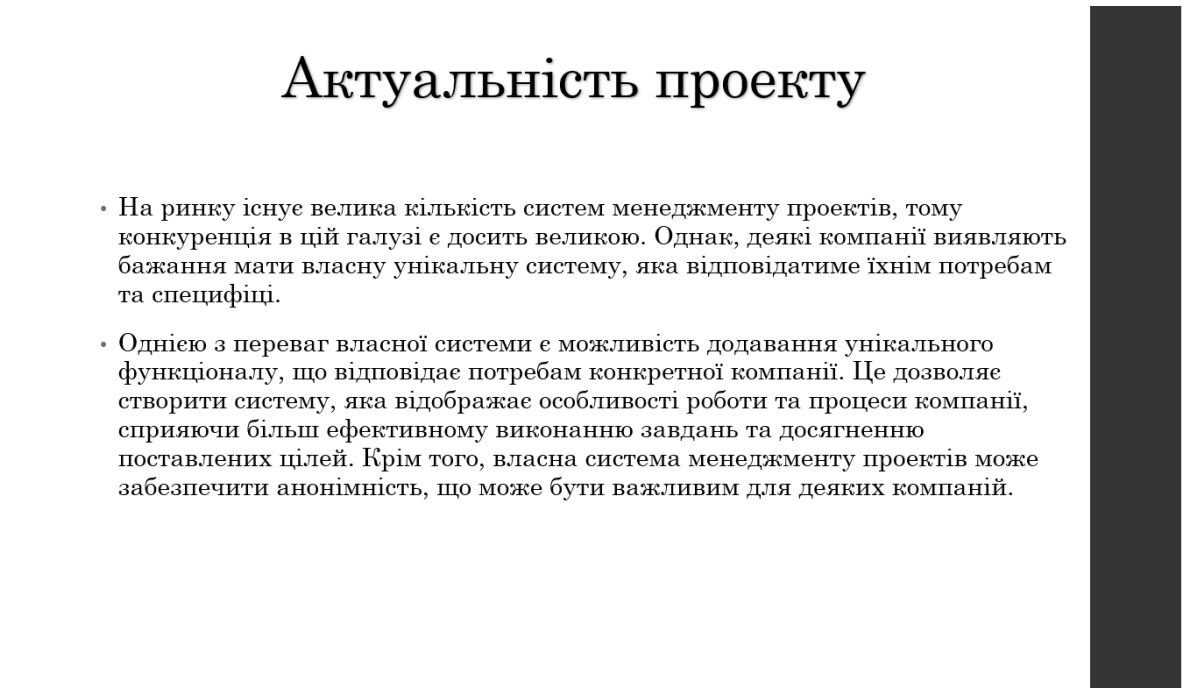


Рисунок Б.2 – Слайд 2

Мета

- Метою кваліфікаційнійної роботи є розробка веб-сервісу, спрямованого на розподіл та облік завдань і робочого часу для IT компанії.

Рисунок Б.3 – Слайд 3

Завдання

- Головним завданням проекту є створення функціонального та ефективного інструменту, який допоможе компанії в управлінні завданнями, контролі часу та забезпечить оптимальний розподіл ресурсів.
- Проведення аналізу вимог до інтерфейсу
- Розробка моделей та алгоритмів майбутнього програмного продукту.
- Виконання програмної реалізації програмного продукту
- Виконання тестових робіт.

Рисунок Б.4 – Слайд 4

Дослідження предметної області

- На початковому етапі було проведено детальний аналіз взаємодії між робітниками IT-компанії. З метою ефективного розподілу завдань і контролю робочого часу, було визначено три ролі, кожна з яких мала свій унікальний функціонал.

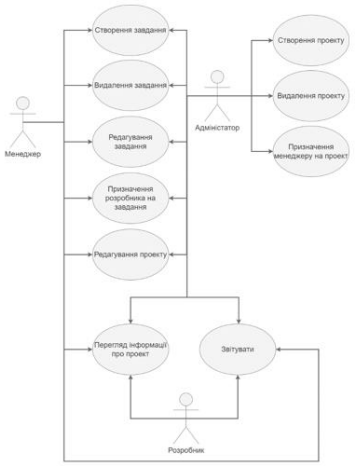


Рисунок Б.5 – Слайд 5

Аналіз вимог до програмного забезпечення

- Ролі: Адміністратор, PM (проектний менеджер), розробник;
- Адміністратор повинен мати повний доступ до функцій системи, включаючи створення, редагування та видалення проектів, призначення PM на ці проекти, а також інші функції;
- PM має повний контроль над проектом, включаючи редагування проекту, додавання, редагування та видалення завдань, а також призначення розробників на ці завдання;
- Розробники повинні мати можливість звітувати про виконання їх поставлених завдань;
- Система повинна підтримувати можливість звітування декількома файлами;
- Слід розробити чат для спілкування між PM та розробником;
- Розглянути можливість того, що PM може виконувати роботу розробника;
- Інтерфейс системи повинен мати приємний та зручний для користувачів вигляд;
- Головна сторінка для адміністратора має містити перелік всіх проектів, імена PM, які призначені на проекти, та дату дедлайну для кожного проекту;
- Головна сторінка PM повинна містити перелік завдань, імена розробників, які призначені на ці завдання, та дату, до якої ці завдання потрібно завершити;
- Головна сторінка розробника повинна містити опис поточного завдання;

Рисунок Б.6 – Слайд 6

Аналіз економічних показників

- Під час аналізу економічних показників була отримана приблизна вартість за допомогою FP-метрик, а також проаналізована окупність розроблюваного рішення.
- У результаті було отримано вартість у 430 умовних одиниць, проте під час аналізу окупність сума бралась у доларах. В результаті середня окупність проекту становить п'ять з половиною місяців. А у гіршому випадку, при використанні готового рішення Worksection, буде вісім місяців.

Коефіцієнт регулювання складності		Оцінка
№	Назва	
F1	Передачі даних	2
F2	Розподілена обробка даних	3
F3	Продуктивність	0
F4	Поширеність використовуваної конфігурації	4
F5	Швидкість транзакцій	5
F6	Оперативне введення даних	4
F7	Ефективність роботи кінцевого користувача	4
F8	Оперативне відновлення	3
F9	Складність обробки	3
F10	Повторна використовуваність	3
F11	Легкість інсталяції	1
F12	Легкість експлуатації	4
F13	Різноманітні умови розміщення	0
F14	Простота змін	4
Сумма		36

Рисунок Б.7 – Слайд 7

Аналіз та вибір архітектури

- Під час аналізу архітектури було розглянуто низку різних архітектур, таких як клієнт-серверна архітектура, SPA (Single-Page Application), Microservice.
- Із них було обрано клієнт-серверну архітектуру. Ця архітектура підходить для веб-систем зі складним функціоналом та великою кількістю користувачів, оскільки дозволяє забезпечити високу продуктивність та масштабованість системи.
- Завдяки обраній архітектурі, веб-система зможе успішно опрацьовувати велику кількість користувачів та забезпечити їм швидкий та ефективний доступ до необхідної інформації.

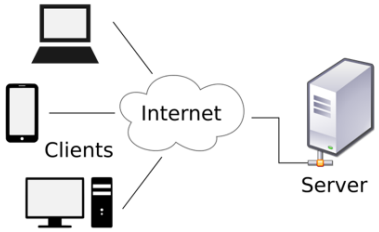


Рисунок Б.8 – Слайд 8

Опис структури даних та моделі бази даних

- На даному етапі було проведено детальний опис та розробка структури даних веб-системи для розподілу та обліку завдань і контролю робочого часу IT-компанії. Визначено необхідні таблиці, поля і зв'язки між ними для ефективного зберігання та організації даних.
- З метою наглядності та кращого розуміння структури даних, була створена діаграма бази даних. Ця діаграма у вигляді графічного зображення відображає всі таблиці, поля та зв'язки між ними.

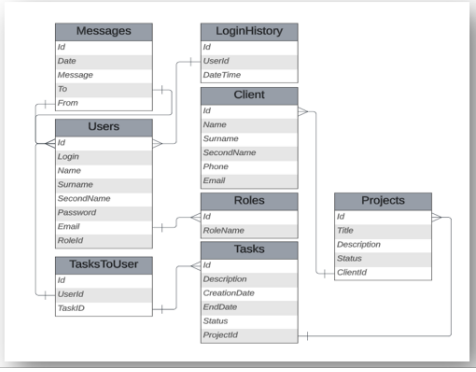


Рисунок Б.9 – Слайд 9

Створення макета веб-додатка та дизайн

- На даному етапі було розписано що потрібно зображати на головних сторінках кожної ролі, та створені прості макети головних сторінок, для узгодження позиції кожного елементу.

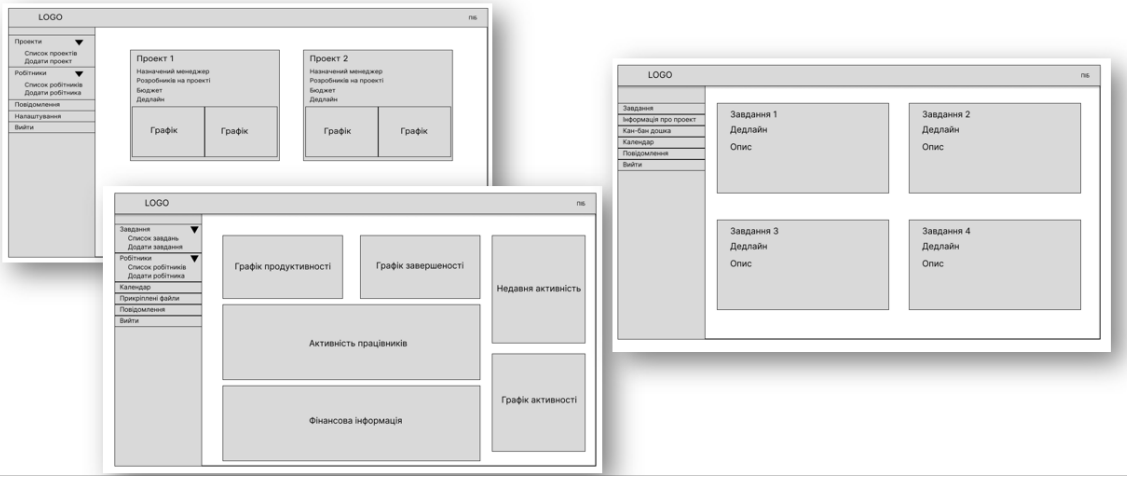


Рисунок Б.10 – Слайд 10

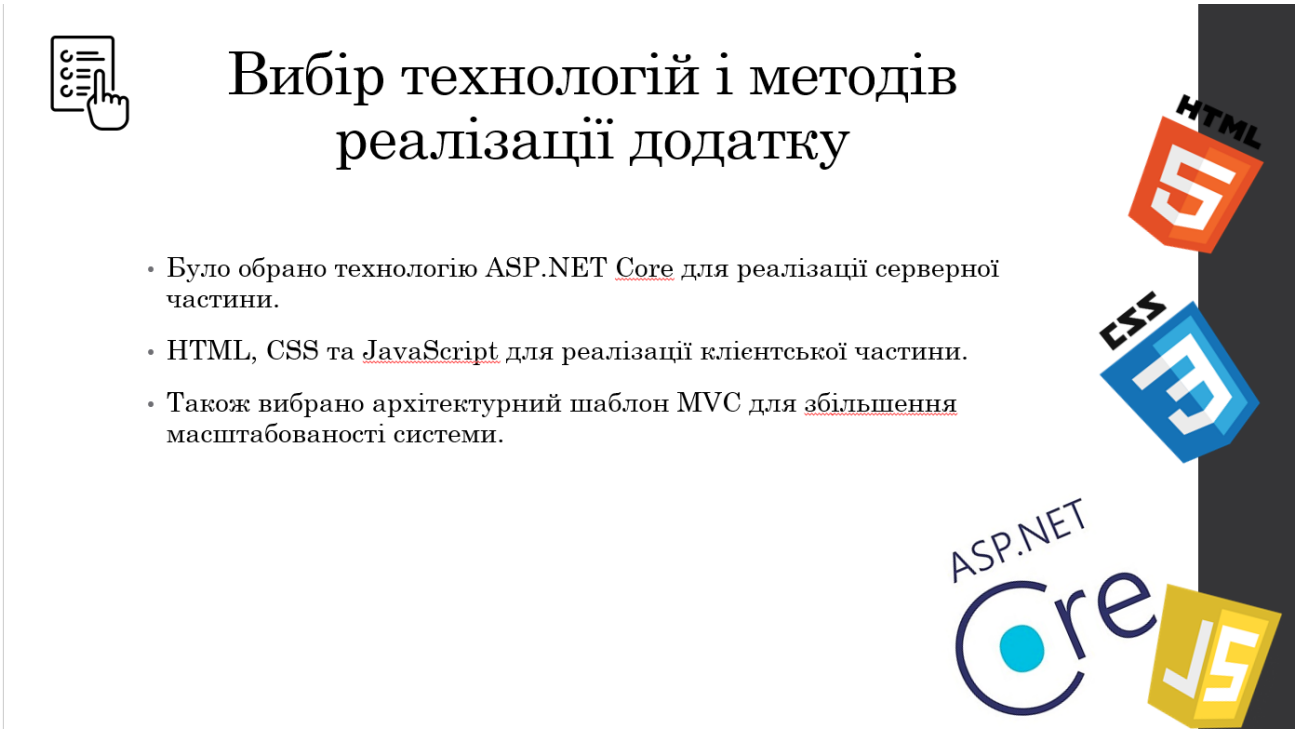


Рисунок Б.11 – Слайд 11

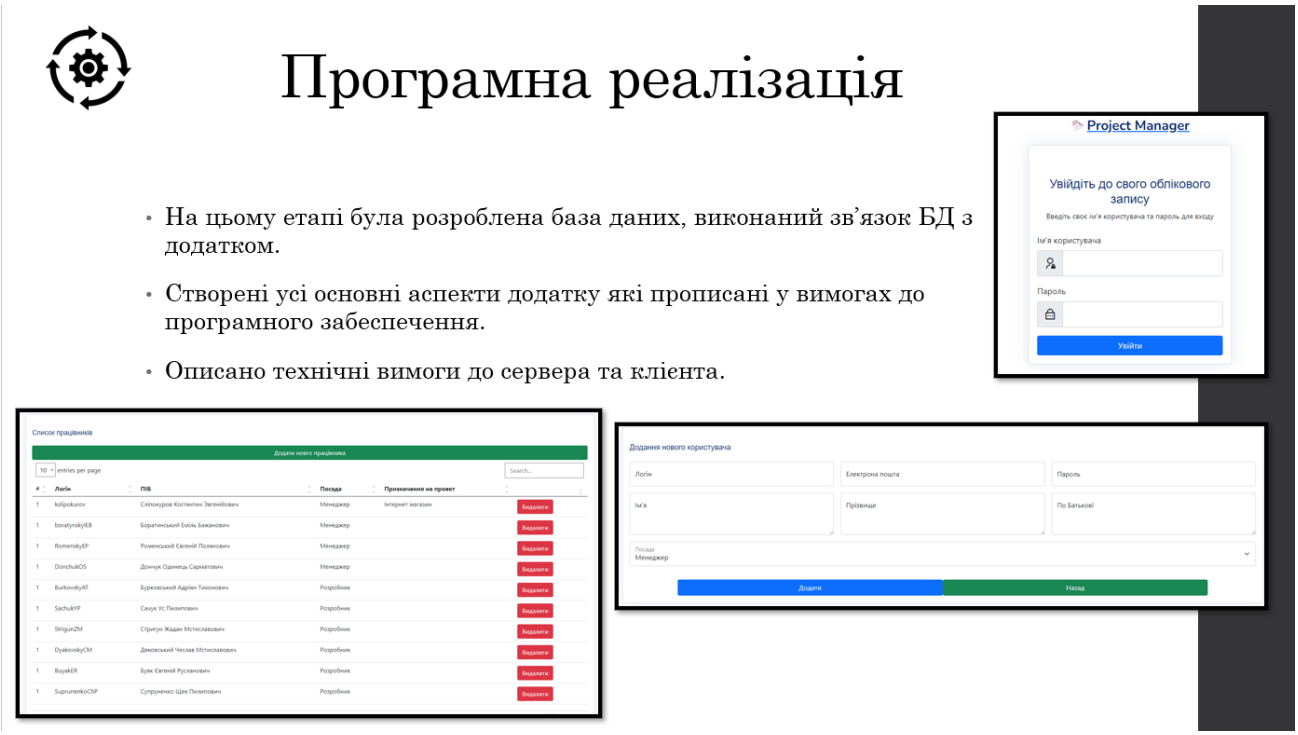


Рисунок Б.12 – Слайд 12

Програмна реалізація

Головна сторінка адміністратора

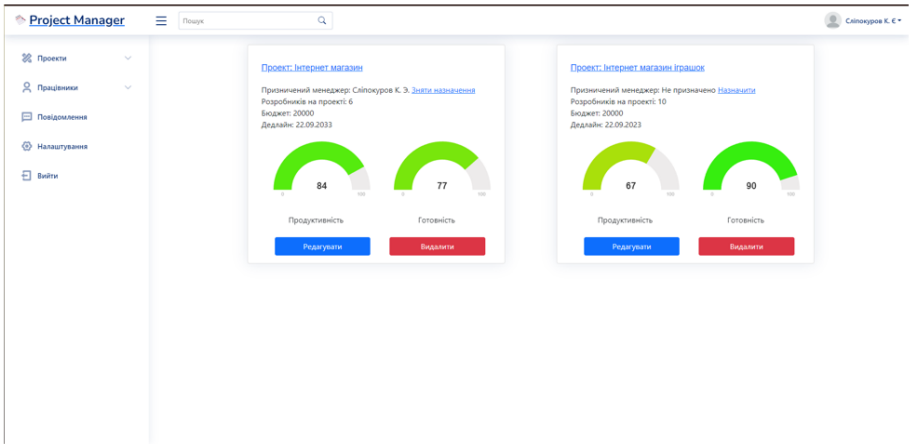


Рисунок Б.13 – Слайд 13

Програмна реалізація

Головна сторінка менеджера

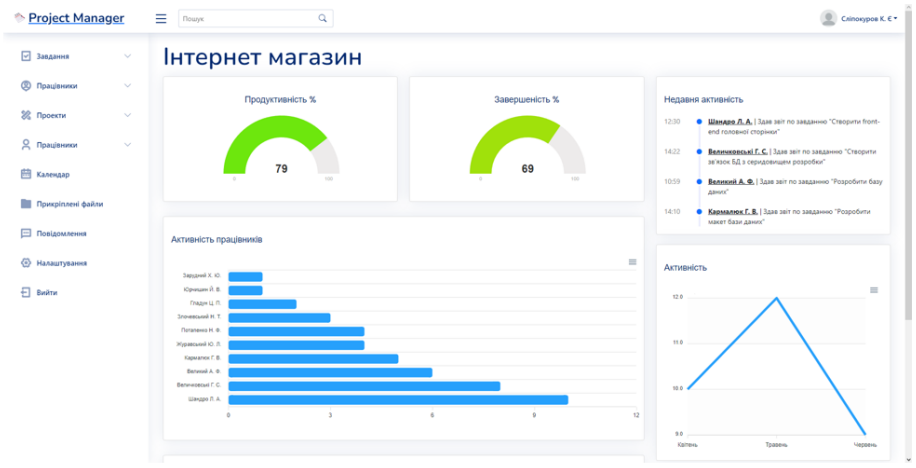


Рисунок Б.14 – Слайд 14

Програмна реалізація

Головна сторінка розробника

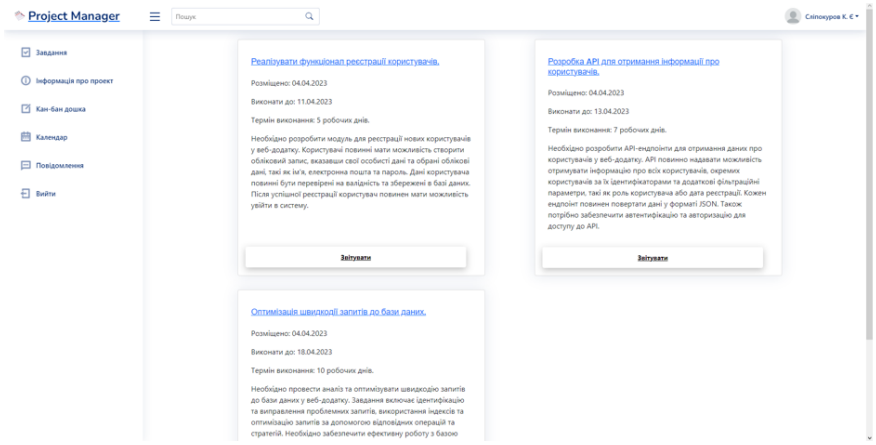


Рисунок Б.15 – Слайд 15

Тестування

- На етапах тестування були проведені ручні тестування системи, перевірка валідації, та інше.
- Зокрема протягом всієї розробки додатку використовувались unit-тести, результати яких теж описано у цьому розділі.
- Далі було проаналізовано всі тести та внесено необхідні корективи до коду.

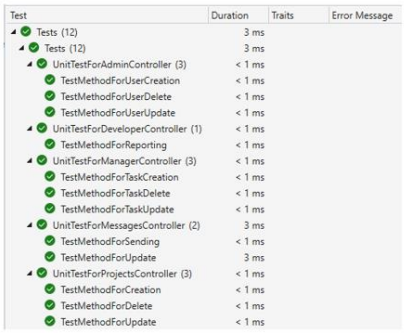


Рисунок Б.16 – Слайд 16

Висновки

- Розроблений веб-сервіс відповідає всім поставленим вимогам та ефективно виконує свої функції. Це стало можливим завдяки використанню передових моделей та алгоритмів, які забезпечують оптимальну роботу системи. Процес валідації та верифікації підтвердив, що веб-сервіс працює бездоганно та відповідає вимогам до якості і надійності.
- Отже, можна стверджувати, що мета кваліфікаційної роботи, яка полягала у розробці веб-системи для розподілу та обліку завдань і контролю робочого часу IT-компанії, була досягнута. Веб-система успішно розроблена, пройшла валідацію, верифікацію та тестування, підтвердивши свою відповідність встановленим вимогам.

Рисунок Б.17 – Слайд 17

Дякую за увагу

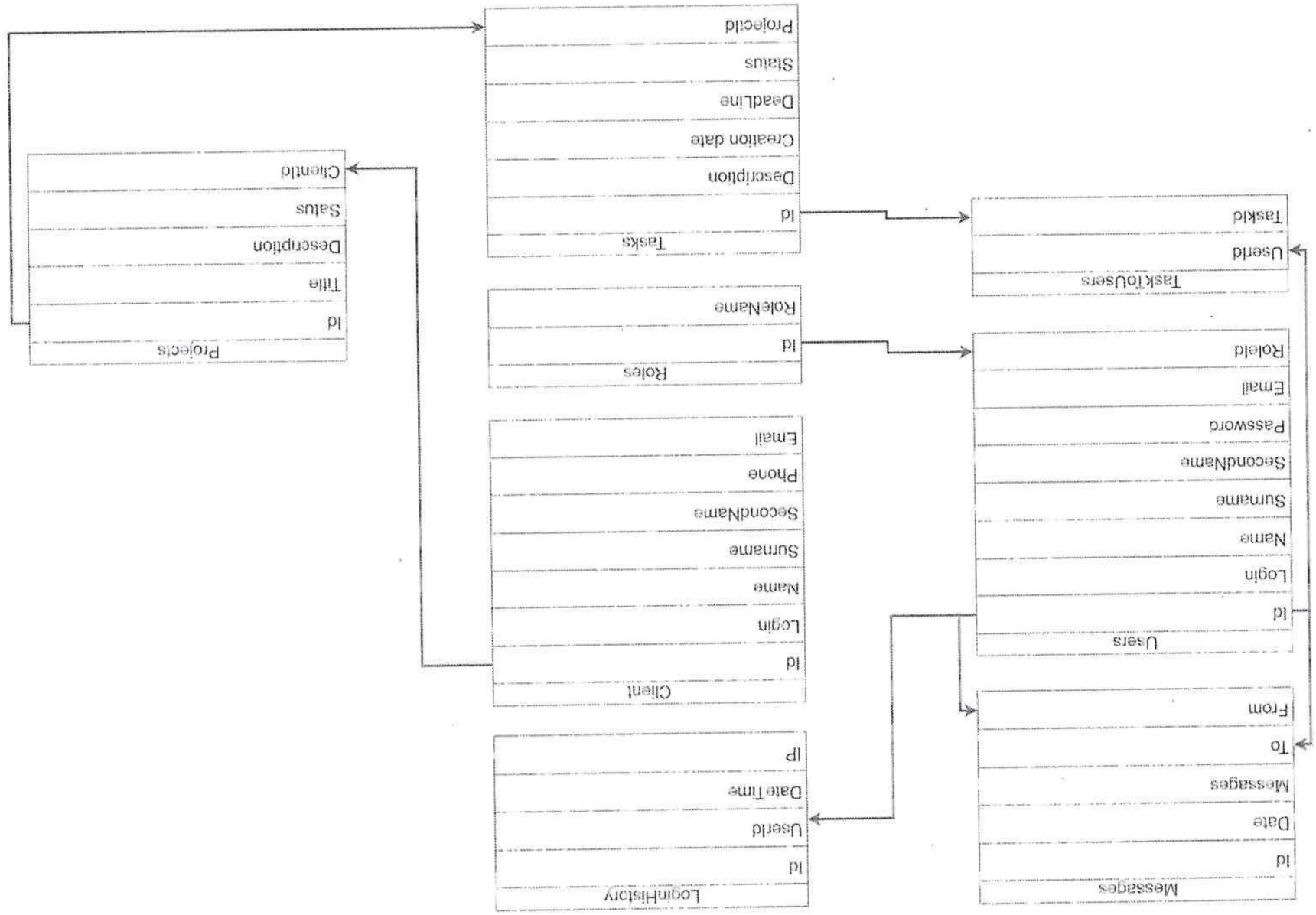
Доповідь закінчено

Рисунок Б.18 – Слайд 18

ДОДАТОК В

(обов’язковий)

ГРАФІЧНА ЧАСТИНА



КвРПЗ. 190141.01.18.Е8

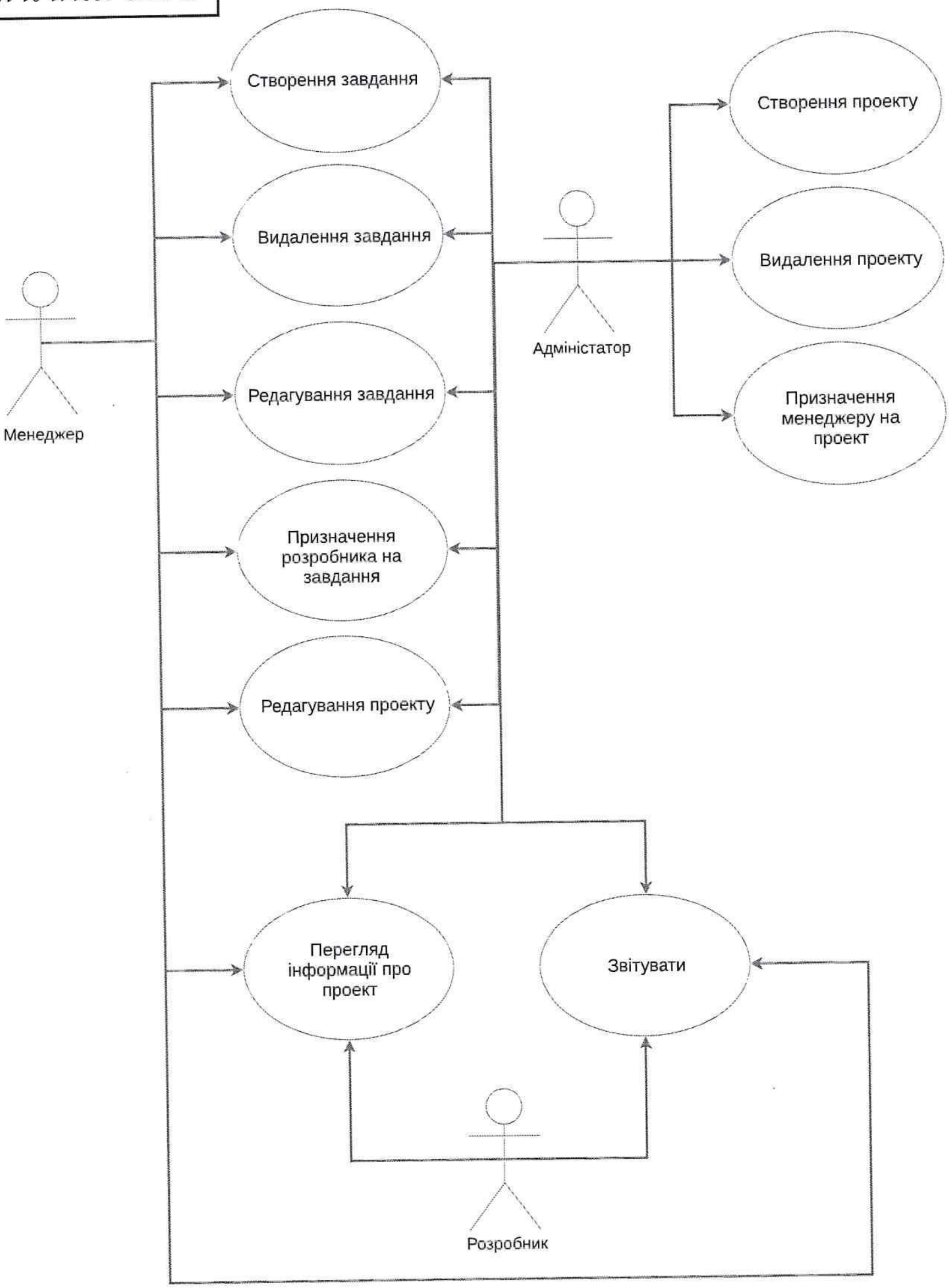
Изм.	Арк.	№ докум.	Підпис	Дата
Розробив		Сліпокуров К. Є.		5.06
Керівник		Гудим І. В.		5.06
Консульт.				
Н. Контр.				
Зав. каф.		Бездітко Л.Г.		6.06

ER-діаграма бази даних

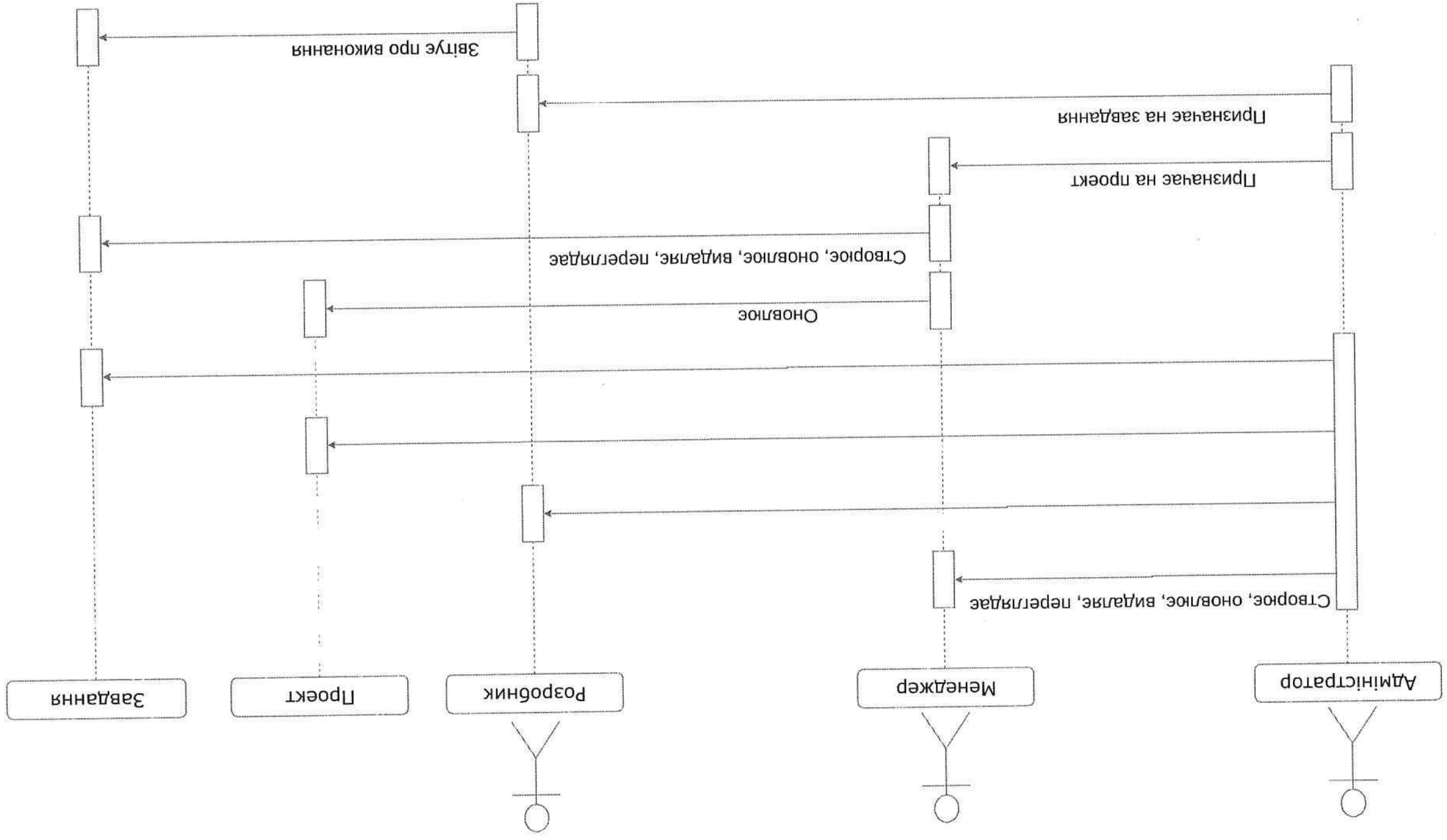
Літера | Маса | Аркуш 1 | Аркушів 3

ХНУ, ПЗ-19-1

КвРПІЗ. 190141.01.18.Е8



Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 190141.01.18.Е8		
Розробив		Співокуров К. Є		3.06	UML-діаграма варіантів використання	Літера	Маса	Масштаб
Керівник		Гурман І. В.		3.06				
Консульт.						Аркуш 2	Аркушів 3	
Н. Контр.		Форкун Ю.В.		05.06		ХНУ, ІПЗ-19-1		
Зав. каф.		Бедратюк Л.П.		6.06				



Зм.	Арк.	№ докум.	Підпис	Дата	КвРПЗ. 190141.01.18.Е8		
Розробив		Сліпокуров К. Є.		5.06	UML-діаграма варіантів взаємодії	Літера	Маса
Керівник		Гурман І. В.		3.06			
Консульт.						Аркуш 3	Аркушів 3
Н. Контр.		Форкун Ю.В.		5.06			
Зав. каф.		Бедратюк Л.Г.		6.06			

КвРПЗ. 190141.01.18.Е8

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Сліпокуров К. Є.

Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05.02.2023

дата

підпис

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015403889

Дата перевірки:
03.06.2023 06:28:08 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2023 06:45:25 EEST

ID користувача:
100005589

Назва документа: **Сліпокуров Антиплагіат**

Кількість сторінок: **72** Кількість слів: **14035** Кількість символів: **111047** Розмір файлу: **1.09 MB** ID файлу: **1015067593**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.98%
Схожість

Найбільша схожість: **2.57%** з джерелом з Бібліотеки (ID файлу: **1015040217**)

4.74% Джерела з Інтернету	613	Сторінка 74
4.07% Джерела з Бібліотеки	143	Сторінка 78

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	21
Підозріле форматування	14 сторінок

Sat Jun 03 05:26:36 EEST 2023, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114633 Назва: БКР Веб-система для розподілу та обліку завдань і контролю робочого часу IT-компанії Додано в БД: 2023-06-03 Автора: Сліпокуров К.Є. Керівники: Гурман І.В. к.т.н. доц. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	96130	766	1163 (1%)	22 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Сліпокуров Костянтин Євгенійович

Тема Веб-система для розподілу та обліку завдань і контролю робочого часу ІТ-компанії

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 62

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було проведено дослідження предметної області та визначено функціональні та нефункціональні вимоги. Також було визначено вартість проекту та проаналізовано окупність проекту шляхом порівняння його вартості з вартістю інших рішень на ринку. Під час проектування програмного забезпечення було проведено аналіз та вибір архітектури веб-застосунка, описано структуру даних та модель бази даних, а також розроблено серверну та клієнтську частини веб-застосунка. Створено макет веб-застосунка та розроблено дизайн. Також розроблена база даних та програмні модулі. Було проведене тестування програмного забезпечення з метою підтвердження його коректної роботи та готовності до експлуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі кваліфікаційної роботи була доведена актуальність теми, визначена мета та завдання дипломного проектування, що свідчить про усвідомлення актуальності проблем і потреб в обраній предметній області. У першому розділі був проведений аналіз предметної області, було визначено вартість проекту та проаналізовано окупність проекту, визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі був проведений аналіз сучасних архітектур, розглянуті їх переваги і недоліки та прийнято рішення обрати MVC архітектуру та модель клієнт-сервер. У третьому розділі представлено всі залежності для написання коду, виконано практичну розробку програмних модулів і описано їх особливості. Також у третьому розділі було проведено модульне тестування системи у відповідності до функціональних вимог.

4. Позитивні сторони роботи В роботі проведений ретельний аналіз предметної області, відомих рішень та вимог до програмного забезпечення, в тому числі було розраховано вартість проекту за допомогою FP-метрик та визначено окупність проекту. Робота включає модульне тестування системи, яке було проведено відповідно до функціональних вимог.

5. Негативні сторони роботи Недостатня деталізація використовуваних методів та технологій, що може ускладнити розуміння процесу реалізації і обґрунтування вибору конкретних рішень. Не представлено алгоритм, за яким розраховуються швидкість виконання проекту та поточні витрати.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано на високому рівні. Робота містить діаграми та рисунки, які ілюструють структуру системи, архітектурне рішення та основні компоненти програмного забезпечення. Графічні матеріали чіткі та відображають необхідну інформацію, що сприяє кращому розумінню роботи та її ключових аспектів. Пояснювальна записка оформлена відповідно до вимог чинних стандартів, є добре структурованою, кожен розділ містить необхідні підрозділи. В пояснювальній записці представлено всю необхідну інформацію про роботу, аргументовано викладено проведені дослідження та прийняті рішення.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки викладений структуровано, послідовно, чітко та просто. Пояснювальна записка дозволяє легко зрозуміти весь викладений матеріал у межах тематики проектування.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ доцент каф. комп'ютерної інженерії та інформаційних систем Бобровський К. О.

"05" 06 2023 р. (підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Веб-система для розподілу та обліку завдань і контролю робочого часу IT-компанії»

Автор: Сліпокуров Костянтин Євгенійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Гурман Іван Васильвич, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 6,98% і адресується до 756 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 6.06.23

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Іван ГУРМАН