Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ ЗАДАЧАМИ
WEB APPLICATION FOR MANAGING TASKS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-130
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ШТЕПА С.С.
(ПРІЗВИЩЕ та ініціали)
Керівник ШИТІКОВА О.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент ТЕРЕЩЕНКО Е.В.
(ПРІЗВИЩЕ та ініціали)

2024

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ШТЕПИ Сергія Сергійовича
(ПРІЗВИЩЕ, ім’я, по батькові)
1. Тема проєкту (роботи) Вебзастосунок для управління задачами. Web Application for Managing Tasks

керівник проєкту (роботи) к.т.н., ШИТІКОВА Олена Вікторівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168
2. Строк подання студентом проєкту (роботи) 10 червня 2024 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Обрання засобів та середовища розробки. 3. Проєктування вебзастосунку. 4. Розробка вебзастосунку. 5. Експлуатація та тестування вебзастосунку.

5. Перелік графічного матеріалу (із точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ШИТІКОВА О.В., доцент кафедри ПЗ		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач кафедри ПЗ		

7. Дата видачі завдання « 24 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програмного забезпечення.	2 тиждень	Розділ 3
5	Розробка програмного забезпечення.	3-4 тижні	Розділ 4
6	Тестування програмного забезпечення.	5 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Сергій ШТЕПА
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Олена ШИТІКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 124 с., 38 табл., 44 рис., 3 дод., 30 джерел.

ВЕБЗАСТОСУНОК, УПРАВЛІННЯ ЧАСОМ, ОРГАНІЗАЦІЯ, ЗАДАЧІ, НОТАТКИ.

Об’єкт дослідження – процес управління задачами.

Предмет дослідження – вебзастосунок для управління задачами.

Мета роботи – розробка вебзастосунку, який надасть користувачам зручні інструменти для керування задачами.

Матеріали, методи та технічні засоби: мови програмування C# та JavaScript, персональний комп’ютер з процесором Intel Core i3-8100F під управлінням операційної системи Windows 10.

Результати. Розроблено функціональний вебзастосунок, який забезпечить користувачам зручні інструменти для планування, організації та виконання задач, а також збереження нотаток.

Висновки. Розроблений вебзастосунок забезпечить користувачам зручні інструменти для планування, організації та виконання задач, а також збереження нотаток.

Галузь використання – широкі практичні застосування у різних сферах діяльності, починаючи від особистого використання до використання в бізнесі та освіті.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 124 pages, 38 tables, 44 figures, 3 appendixes, 30 sources.

WEB APPLICATION, TIME MANAGMENT, ORGANIZATION, TASKS, NOTES.

Object of study is a process of managing tasks.

Subject of study is a web application for managing tasks.

The purpose of the work is to develop a web application that will provide users with convenient tools for managing tasks.

Materials, methods, and tools: programming languages C# and JavaScript, personal computer with Intel Core i3-8100F processor running Windows 10 operating system.

Results. A functional web application has been developed, which provides users with convenient tools for planning, organizing, and executing tasks, as well as for saving notes.

Conclusions. The developed web application provides users with convenient tools for planning, organizing, and executing tasks, as well as for saving notes.

The field of use has wide practical applications in various fields of activity, ranging from personal use to use in business and education.

ЗМІСТ

С.

Перелік скорочень та умовних познак 8

Вступ 9

1 Аналіз предметної області 11

1.1 Організація робочого та особистого часу 11

1.2 Огляд існуючих аналогів застосунків планування та управління задачами 12

1.3 Постановка завдання 17

1.4 Висновки за розділом 1 18

2 Обрання засобів та середовища розробки 20

2.1 Вибір мови програмування 20

2.2 Огляд особливостей середовища розробки 22

2.3 Системи управління базами даних 24

2.4 Висновки за розділом 2 25

3 Проєктування вебзастосунку 27

3.1 Опис основних вимог до програми 27

3.2 Вимоги до технічних і програмних засобів 29

3.3 Розробка архітектури програми 30

3.4 Розробка користувацького інтерфейсу 45

3.5 Висновки за розділом 3 55

4 Розробка вебзастосунку 56

4.1 Алгоритми та UML-діаграми послідовності 56

4.2 Опис модулів розроблюваної системи 60

4.3 Висновки за розділом 4 77

5 Експлуатація та тестування вебзастосунку 78

5.1 Опис роботи з програмою 78

5.2 Інструкція з використання 81

5.3 Тестування 81

5.4 Висновки за розділом 5 88

Висновки 89
Перелік джерел посиланя 90
Додаток А Технічне завдання 93
А.1 Вступ 94
А.2 Підстави для розробки 94
А.3 Призначення розробки 94
А.4 Вимоги до програми чи програмному виробу 97
А.5 Вимоги до програмної документації 101
А.6 Техніко-економічні показники 102
А.7 Стадії та етапи розробки 102
А.8 Порядок контролю та приймання 102
Додаток Б Текст програми 103
Б.1 Клас IsOwnerAuthorizationHandler 104
Б.2 Клас EmailSender 106
Б.3 Клас AccountController 107
Додаток В Слайди презентації 117

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE – Integrated Development Environment

UML – Unified Modeling Language

MVC – Model-View-Controller

HTTP – Hypertext Transfer Protocol

СУБД – система управління базами даних

ВСТУП

Сучасний світ характеризується швидким темпом життя та постійним потоком інформації, що вимагає ефективних засобів для організації робочого та особистого часу. У зв'язку з цим виникає актуальна проблема керування задачами та нотатками. Незважаючи на існуючі програмні засоби для цього, багато користувачів зустрічаються з проблемами ефективного планування та організації своєї діяльності.

На сьогоднішній день багато провідних наукових установ та організацій вже звернули увагу на цю проблему та розробили власні інструменти для керування задачами та нотатками. Однак існує необхідність в подальшому дослідженні та розробці більш ефективних та зручних засобів для вирішення цієї проблеми.

Об’єктом дослідження є процес управління задачами.

Предметом дослідження є вебзастосунок для управління задачами.

Метою цієї роботи є розробка вебзастосунку, який надасть користувачам зручні інструменти для керування задачами. Важливість цього дослідження полягає в його потенційній здатності покращити продуктивність та ефективність роботи користувачів у різних сферах їхньої діяльності.

Ця робота ставить перед собою наступні завдання:

– проаналізувати існуючі підходи до керування задачами та нотатками;

– розробити архітектуру та функціональні можливості вебзастосунку для керування задачами та нотатками;

– реалізувати розроблену систему та провести її тестування.

Результатом цієї роботи буде створення функціонального вебзастосунку, який забезпечить користувачам зручні інструменти для планування, організації та виконання задач, а також збереження нотаток. Такий застосунок може мати широкі практичні застосування у різних сферах

діяльності, починаючи від особистого використання до використання в бізнесі та освіті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі проводиться докладний аналіз предметної області, що стосується керування задачами та нотатками. Він включає огляд існуючих програмних рішень, їхніх переваг і недоліків, а також формальну постановку завдань роботи.

1.1 Організація робочого та особистого часу

Керування задачами та нотатками є одним з ключових елементів ефективного організаційного та особистого управління часом.

Тайм-менеджмент (від анг. time management) – це систематичний підхід до організації та використання свого часу з метою досягнення максимальної продуктивності та поставлених цілей. Основна ідея полягає в тому, щоб ефективно розподіляти час між різними задачами, проєктами та активностями, забезпечуючи баланс між роботою, особистим життям і відпочинком [1].

В сучасному світі, де багато задач можна виконати за допомогою технологій, правильний тайм-менеджмент допомагає оптимізувати використання цих інструментів та зменшити час, витрачений на рутинні роботи [1].

Існує п’ять ключових принципів успішного тайм-менеджменту:

– встановлення пріоритетів – спрямування своєї уваги в першу чергу на важливі речі;

– планування – збереження організованості і зосередженості, уникнення зайвого стресу;

– поділ задач і делегування – розбиття великих задач на менші керовані кроки;

– зосередженість і уникнення відволікань – вимкнення сповіщень та фокус на задачах;

– стале вдосконалення та аналіз – оцінювання методів та підходів до управління часом, пошук способів оптимізувати процес [1].

Керування задачами – це процес планування, організації та контролю за виконанням різних задач або обов'язків. Включає у себе призначення пріоритетів, визначення строків та відстеження прогресу.

Керування нотатками – це короткі записи або відзначення, які служать для збереження інформації або нагадування про певні пункти.

Об'єктом дослідження є процес управління задачами. Предметом дослідження є функціональний та зручний інструмент у вигляді вебзастосунку, який надає користувачам можливість ефективно планувати, виконувати та відстежувати задачі, а також зберігати та організовувати нотатки.

1.2 Огляд існуючих аналогів застосунків планування та управління задачами

1.2.1 Програмний продукт Evernote

Evernote це популярний вебзастосунок для управліннями задачами та нотатками. Від має багато додаткових можливостей:

– система міток, що дозволяє сортувати об’єкти що мають якісь спільні риси;

– система блокнотів, що дозволяє об’єднати об’єкти для більш зручного логічного доступу до них;

– головна сторінка відображає нещодавні об’єкти [2].

Також Evernote має можливість встановлення мобільної версії застосунку для операційних систем Android та IOS [2].

Приклад створеної нотатки з задачами представлено на рисунку 1.1. Рисунок демонструє нотатку що окрім тексту містить в собі задачі. Окреме управління задачам доступне тільки у платній версії застосунку.

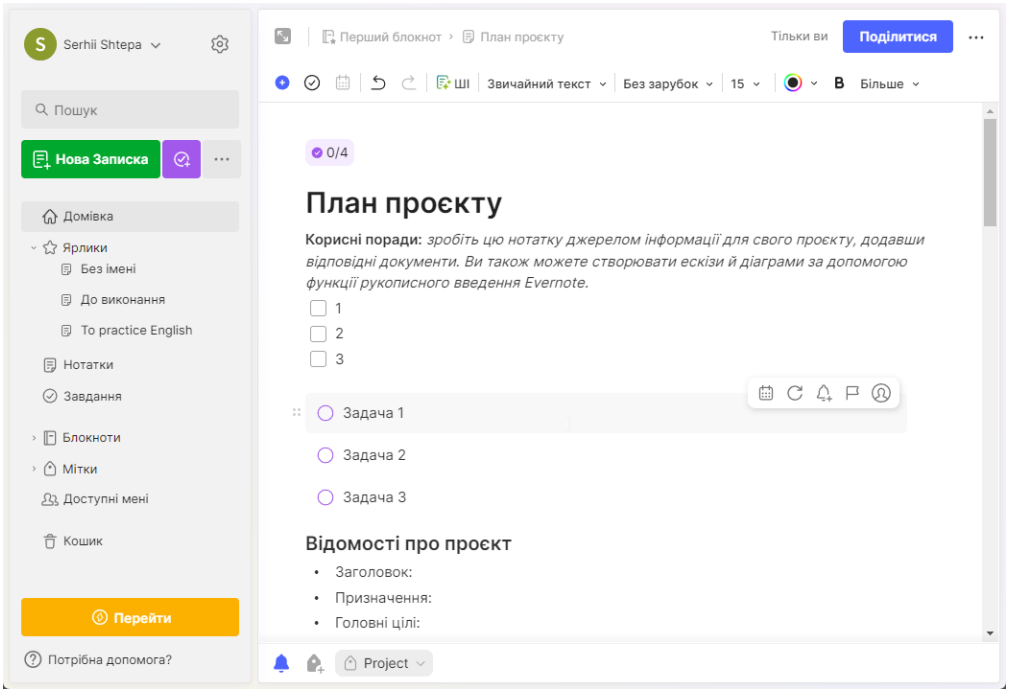

Рисунок 1.1 – Приклад створеної нотатки Evernote

1.2.2 Програмний продукт Trello

Trello – це популярний вебзастосунок для управління проєктами. Він має багато додаткових можливостей:

– система канбан-дошок;

– система перегляду задач у різних поданнях;

– система плагінів;

– система міток, що дозволяє сортувати об’єкти що мають якісь спільні риси;

– система робочих просторів, що дозволяє об’єднати канбан-дошки для більш зручного логічного доступу до них;

– можливість створення повторюваних задач;

– система кошику [3].

Також Trello має можливість встановлення мобільної версії застосунку для операційних систем Android та IOS [3].

Приклад створеної канбан-дошки з задачами представлено на рисунку 1.2.

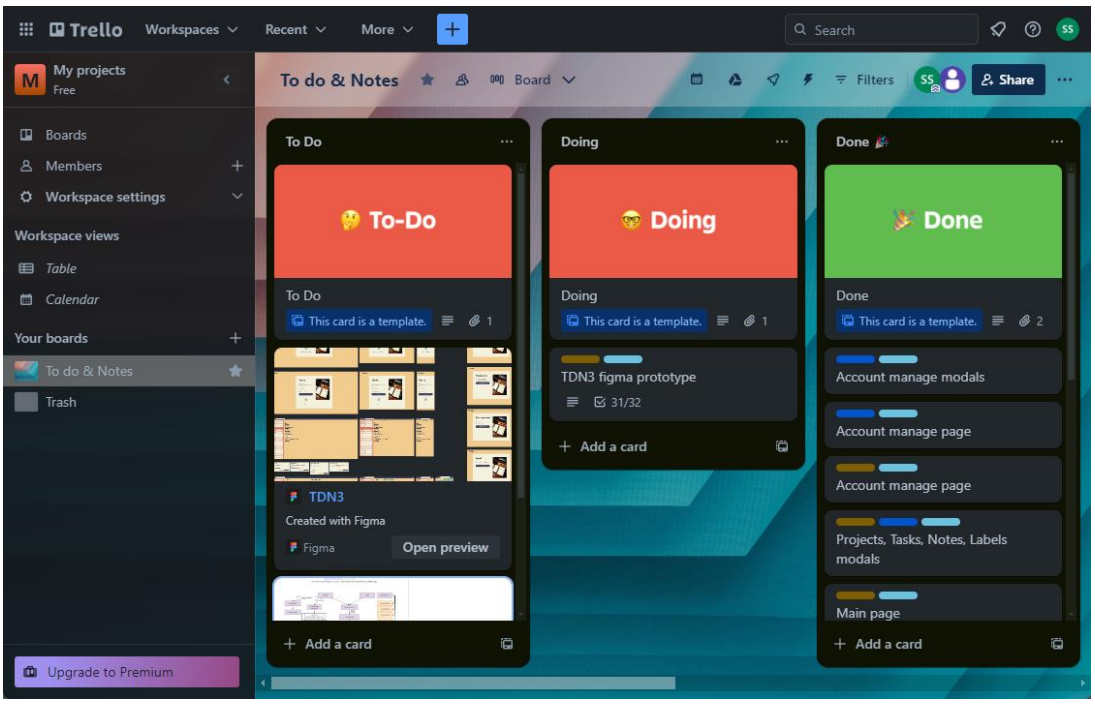

Рисунок 1.2 – Приклад створеної канбан-дошки Trello

1.2.3 Програмний продукт Microsoft To Do

Microsoft To Do – це популярний вебзастосунок для управліннями задачами.

Він має кілька додаткових можливостей:

– система списків, що дозволяє об’єднати задачі для більш зручного логічного доступу до них;

– система перегляду задач у різних поданнях;

– можливість створення повторюваних задач [4].

Також Microsoft To Do має можливість встановлення мобільної версії застосунку для операційних систем Android та IOS [4].

Приклад створеного списку з задачами представлено на рисунку 1.3. На мою думку у цьому застосунку не вистачає системи міток.

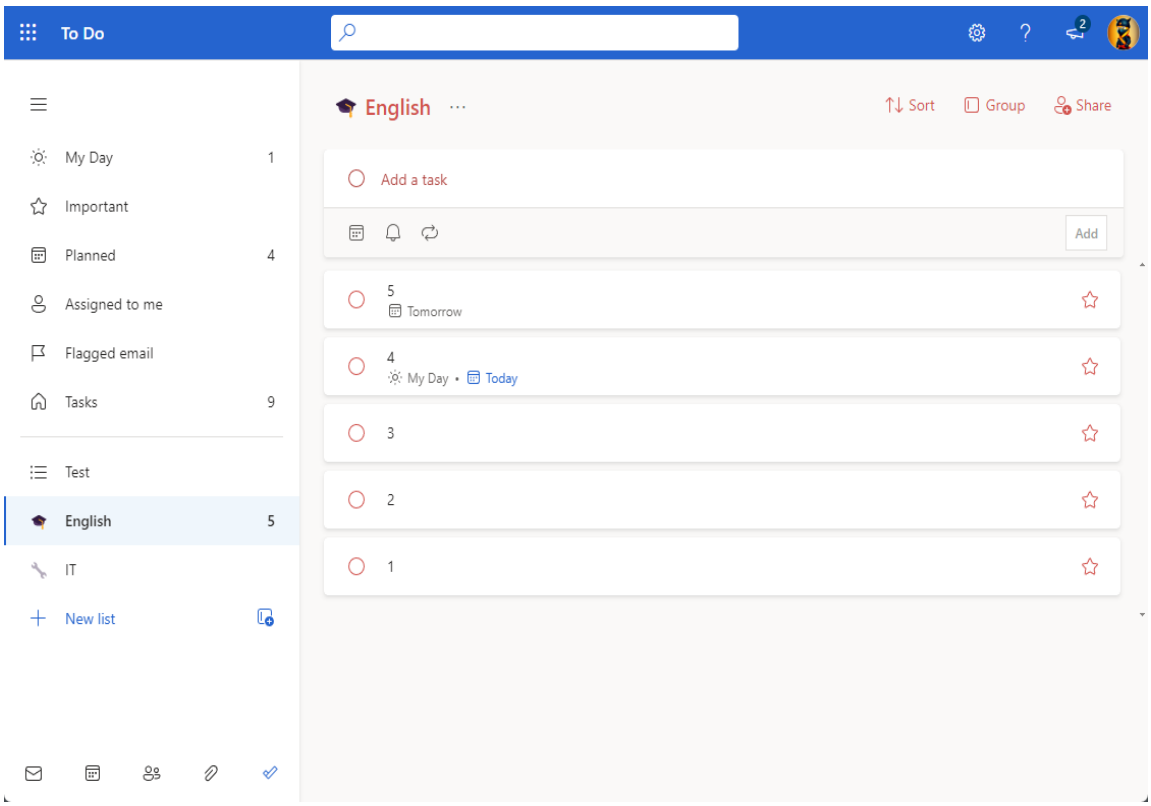

Рисунок 1.3 – Приклад створеного списку Microsoft To Do

1.2.4 Програмний продукт Google Keep

Google Keep – це популярний вебзастосунок для управліннями нотатками.

Він має кілька додаткових можливостей:

– система міток, що дозволяє сортувати об’єкти що мають якісь спільні риси;

– система перегляду задач у різних поданнях;

– система кошику [5].

Також Google Keep має можливість встановлення мобільної версії застосунку для операційних систем Android та IOS [5].

Приклад подання створених нотаток за міткою представлено на рисунку 1.4. На мою думку у цьому застосунку не вистачає більш розвиненої системи задач та проєктів.

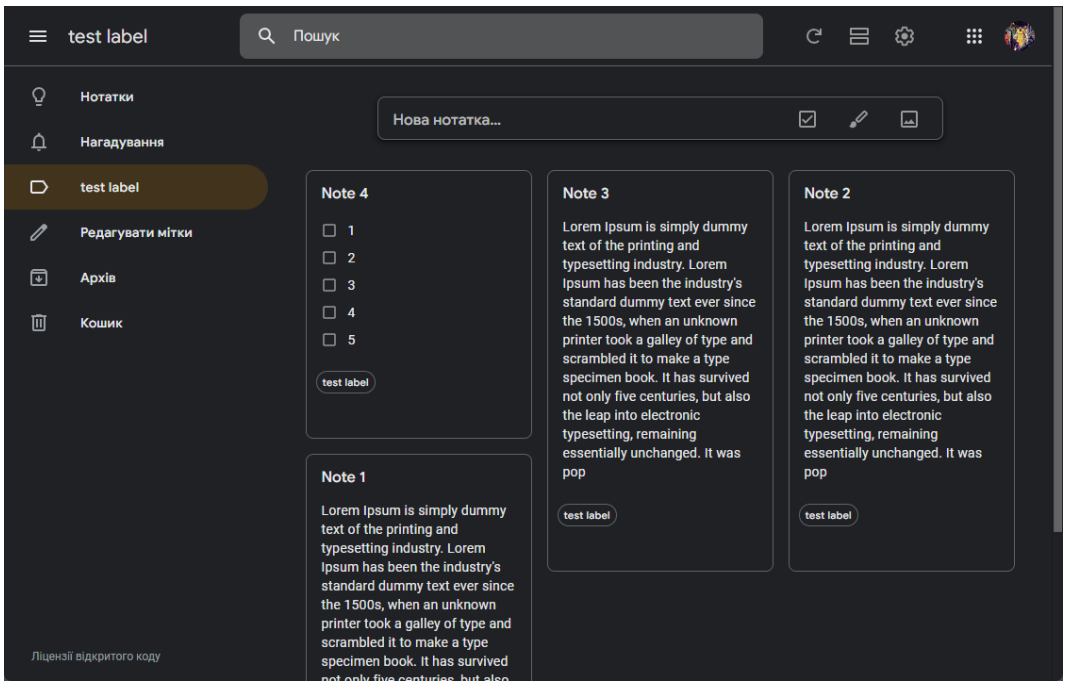

Рисунок 1.4 – Приклад подання створених нотаток Google Keep

1.2.5 Програмний продукт Todoist

Todoist це популярний вебзастосунок для управліннями задачами.

Він має кілька додаткових можливостей:

– система проєктів, що дозволяє об’єднати задачі для більш зручного логічного доступу до них;

– система перегляду задач у різних поданнях;

– можливість створення повторюваних задач;

– система міток, що дозволяє сортувати об’єкти що мають якісь спільні риси;

– система канбан-дошок;

– система плагінів [6].

Також Todoist має можливість встановлення мобільної версії застосунку для операційних систем Android та IOS [6].

Приклад створеного проєкту зі списком задач представлено на рисунку 1.5.

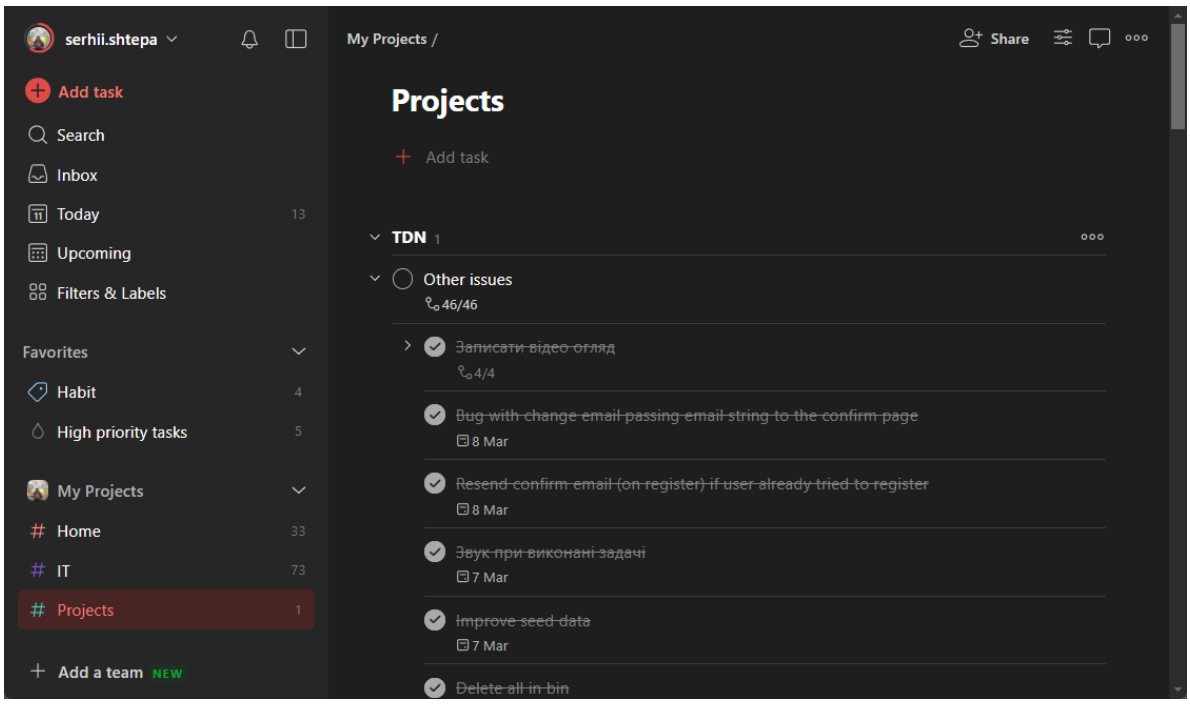

Рисунок 1.5 – Приклад створеного проєкту зі списком задач Todoist

1.3 Постановка завдання

Вище були згадані найбільш популярні вебзастосунки сучасності, які надають можливості керування задачами та нотатками. Проте, крім очевидних переваг, кожен з них має свої недоліки. Наприклад, складний інтерфейс може ускладнювати процес користування. Також, багато користувачів не використовують більшу частину наданого функціоналу, що може призводити до зайвого завантаження інтерфейсу та заплутаності користувача.

В таблиці 1.1 проведено порівняння вже існуючих програмних продуктів та програми, яка буде розроблена в ході виконання роботи.

Таблиця 1.1 — Порівняння програмних продуктів

	Evernote	Trello	Microsoft To Do	Google Keep	Todoist	TDN3
Робота online	+	+	+	+	+	+
Кросплатформенність	+	+	+	+	+	+

Продовження таблиці 1.1

	Evernote	Trello	Microsoft To Do	Google Keep	Todoist	Розроблюваний вебзастосунок
Задачі	+	+	+	-	+	+
Нотатки	+	+	-	+	-	+
Система проєктів	+	+	+	-	+	+
Система міток	+	+	-	+	+	+
Простий інтерфейс	-	-	+	+	+	+

Після проведення аналізу існуючих аналогів програм для створення задач та нотаток стало очевидним, що більшість з них характеризується або складністю інтерфейсу, або недостатньою функціональністю. Тому головне завдання роботи полягає у пошуку оптимального рішення, яке би забезпечило максимальну зручність інтерфейсу та необхідну функціональність.

Метою цієї роботи є розробка вебзастосунку, який надасть користувачам зручні інструменти для керування задачами.

Основними задачами роботи є:

– проаналізувати існуючі підходи до керування задачами та нотатками;

– розробити архітектуру та функціональні можливості вебзастосунку для керування задачами та нотатками;

– реалізувати розроблену систему та провести її тестування.

1.4 Висновки за розділом 1

В першому розділі було проведено аналіз предметної області. Було здійснено огляд і порівняння існуючих програмних рішень, виявлено як

переваги так і недоліки. Після чого було сформовано тему та завдання роботи.

2 ОБРАННЯ ЗАСОБІВ ТА СЕРЕДОВИЩА РОЗРОБКИ

2.1 Вибір мови програмування

Обираючи мову програмування для розробки, важливо враховувати низку факторів, таких як продуктивність розробника, ефективність програми, доступність інструментів, екосистема, безпека, масштабованість та інші. Вибір мови програмування для розробки як на бекенді, так і на фронтенді, є критично важливим етапом у процесі створення програмного забезпечення.

Бекенд – це внутрішня, прихована від користувача начинка сайту або вебпрограми. Іншими словами, це частина сервісу, яка працює на віддаленому сервері, а не у браузері чи персональному комп’ютері. Бекенд обробляє і постачає користувачеві дані, які потім відображає фронтенд: так називають інтерфейс користувача, видиму частину сайту або програми, яка працює на вашому пристрої [7].

Для розробки серверної частини вебзастосунку постає потреба у виборі бекенд мови програмування. Існує кілька популярних бекенд мов програмування, таких як C# або Java [8]. Розглянемо переваги та недоліки низки популярних бекенд мов програмування (таблиця 2.1).

Таблиця 2.1 – Переваги та недоліки бекенд мов програмування [9]

Мова програмування	Переваги	Недоліки
C#	Продуктивність, зручний синтаксис, екосистема, велика кількість доступних бібліотек та фреймворків, інтеграція та різні доповнення у Visual Studio	Складність платформи .NET Core для новачків через велику кількість доступних варіантів та інструментів

Продовження таблиці 2.1

Мова програмування	Переваги	Недоліки
Java	Широке використання, висока продуктивність, багатоплатформність.	Складність синтаксису, може бути складною для вивчення.
JavaScript	Універсальність, простота вивчення, велика спільнота	Не така продуктивна, порівнюючи з C#, може бути складною для масштабування.
Python	Простота вивчення, читабельність коду, велика кількість бібліотек	Не така продуктивна, як C#, може бути складною для розробки складних вебзастосунків.

Отже, обираючи C# для розробки бекенду, можна бути впевненим у високій продуктивності, ефективності та масштабованості прокту, а також у великій підтримці та безпеці програми. C# використовує платформу .NET, яка відома своєю ефективністю та швидкодією. CLR (Common Language Runtime) оптимізує виконання коду, що дозволяє створювати швидкодіючі додатки. C# добре підходить для будь-яких проєктів, від невеликих вебсайтів до великих корпоративних систем. Її об'єктно-орієнтований підхід і можливості абстрагування дозволяють легко масштабувати і модифікувати програми з ростом вимог. Для розробки вебзастосунків за допомогою мови C# використовується популярний фреймворк ASP.NET Core [10].

Для розробки клієнтської частини вебзастосунку постає потреба у виборі фронтенд мови програмування. Існує кілька популярних фронтенд мов програмування, таких як JavaScript або TypeScript [8]. Розглянемо основні особливості двох найпопулярніших фронтенд мов програмування (таблиця 2.2).

Таблиця 2.2 – Переваги та недоліки фронтенд мов програмування [9]

Аспект	JavaScript	TypeScript
Типи даних	Динамічно типізована мова	Статично типізована мова
Самостійність	Основна мова веброзробки	JavaScript з статичною типізацією

Мова JavaScript є більш поширеною з більшою кількістю документації. Також вибір цієї мови надає можливість використовувати бібліотеку jQuery. Використання jQuery надає наступні переваги:

– спрощення роботи з DOM;

– кросбраузерність;

– спрощення роботи з AJAX-запитами [11].

З огляду на все вище сказане, було прийнято рішення для фронтенду використовувати мову програмування JavaScript з бібліотекою jQuery.

2.2 Огляд особливостей середовища розробки

Вибір правильного інтегрованого середовища розробки (IDE) має багато сприяє таким факторам:

– підвищення продуктивності: IDE автоматизують багато завдань, що звільняє час для більш творчої роботи;

– покращення якості коду: IDE мають функції перевірки коду, які допомагають виявляти та виправляти помилки;

– зручність використання: IDE зазвичай мають зручний інтерфейс, що полегшує навігацію по коду та пошук необхідних функцій;

– співпраця: IDE можуть використовуватися кількома розробниками для спільної роботи над проєктом.

На сьогоднішній день існує кілька популярних IDE для роботи з мовою програмування C# та JavaScript. Ось деякі з них та їх особливості (таблиця 2.3).

Таблиця 2.3 – Особливості IDE [12]

IDE	Особливості	Платформи	Вартість
Visual Studio	Повнофункціональне середовище з багатою функціональністю та розширеннями	Windows, macOS, Linux	Безкоштовна (Community), платні версії: Professional, Enterprise
Visual Studio Code	Легкий та масштабований текстовий редактор з великою кількістю розширень	Windows, macOS, Linux	Безкоштовна
Rider	Інтелектуальні інструменти для розробки, розроблене компанією JetBrains	Windows, macOS, Linux	Безкоштовна пробна версія, платні версії для комерційного використання

Розглянемо ці IDE більш детально. Проаналізуємо їх переваги та недоліки (таблиця 2.4).

Таблиця 2.4 – Переваги та недоліки IDE [12]

IDE	Переваги	Недоліки
Visual Studio	Широкий вибір інструментів та розширень, велика спільнота користувачів	Великий обсяг ресурсів системи, Вимагає багато місця на диску

Продовження таблиці 2.4

IDE	Переваги	Недоліки
Visual Studio Code	Підтримує багато мов програмування, відкритий вихідний код	Обмежені можливості порівняно з повнофункціональними IDE, потребує додаткових розширень для роботи з C#
Rider	Висока продуктивність, підтримка різних мов програмуванн	Висока вартість, потребує ресурсів системи

Visual Studio є оптимальним вибором з огляду на вартість та наявність вбудованих інструментів розробки.

2.3 Системи управління базами даних

Система управління базами даних (СУБД) відіграє ключову роль у будь-якому проєкті програмного забезпечення, незалежно від його масштабів та призначення. Обрання відповідної СУБД є критичним кроком для успішної розробки та ефективного управління базами даних. Обрання відповідної СУБД визначається рядом факторів, серед яких важливість забезпечення надійності та стабільності роботи, потреби в широкому функціоналі, спрощення адміністрування та розробки, вартість ліцензії та підтримки, а також інтеграція з іншими компонентами програмного забезпечення. Розглянемо переваги та недоліки низки популярних СУБД (таблиця 2.5).

Таблиця 2.5 – Переваги та недоліки СУБД [13]

СУБД	Переваги	Недоліки
Microsoft SQL Server	Надійність та стабільність, широкі функціональні можливості, інтеграція з .NET Core, Azure	Вартість для великих підприємств, складність налаштування
Oracle Database	Широкий набір функцій	Вартість, складна в управлінні
MySQL	Безкоштовна, відносно проста у використані	Менш потужна, надійність, менше функцій в порівнянні зі складнішими СУБД
PostgreSQL	Надійність, широкий набір функцій	Складність в конфігурації та управлінні, менше функцій в порівнянні зі складнішими СУБД

З огляду на вибрану мову програмування – C#, середовище розробки Visual Studio, логічним вибором СУБД буде Microsoft SQL Server. Це дозволить виконувати розробку у одній екосистемі від компанії Microsoft. Саме інтеграції одних технологій в інші нівелюють недоліки у складному налаштуванні СУБД.

2.4 Висновки за розділом 2

У другому розділі було обрано мову програмування для серверної частини вебзастосунку – C#, для клієнтської – JavaScript з бібліотекою jQuery. Також базуючись на мовах програмування було обрано середовище розробки – Visual Studio. З огляду на екосистему що вибудовується (компанії

Microsoft), було прийнято рішення використовувати MS SQL Server у ролі СУБД.

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Опис основних вимог до програми

Опис основних вимог є одним з найважливіших етапів у процесі розробки будь-якого програмного забезпечення, який допомагає забезпечити успішну та ефективну реалізацію програмного забезпечення, забезпечуючи високу якість та відповідність вимогам користувачів та бізнесу.

Розглянемо перелік нефункціональних вимог:

– назва вебзастосунку – «To Do & Notes 3»

– мова вебзастосунку – англійська;

– задачі та нотатки повинні розміщуватися окремо у двох стовпчиках.

Розглянемо перелік функціональних вимог:

а) автентифікація користувача;

1) реєстрація користувача у системі;

2) вхід користувача у систему;

– вхід через електрону адресу;

– вхід за допомогою Google акаунту;

3) вихід користувача із системи;

б) авторизація;

в) управління акаунтом:

1) зміна імені користувача;

2) зміна електронної адреси користувача;

3) встановлення паролю;

4) змінення паролю;

5) під’єднання Google акаунту;

6) від’єднання Google акаунту;

7) видалення акаунту;

г) управління проєктами:

1) проєкт за замовчуванням;

2) створення проєкту;

3) перейменування проєкту;

4) створення копії проєкту;

5) переміщення проєкту до кошику;

6) відновити проєкт з кошику;

7) видалення проєкту назавжди;

ґ) управління мітками:

1) створення мітки;

2) перейменування мітки;

3) видалення мітки назавжди;

д) управління задачами:

1) створення задачі;

2) редагування задачі;

3) створення копії задачі;

4) переміщення задачі до кошику;

5) відновити задачу з кошику;

6) видалення задачі назавжди;

е) управління нотатками:

1) створення нотатки;

2) редагування нотатки;

3) створення копії нотатки;

4) переміщення нотатки до кошику;

5) відновити нотатку з кошику;

6) видалення нотатки назавжди;

є) кошик:

1) очистити весь кошик;

2) видалення окремого проєкту/задачі/нотатки;

2) відновлення окремого проєкту/задачі/нотатки;

ж) фільтрування:

1) за поданням;

2) за проєктом;

3) за міткою;

4) за пошуковим рядком;

з) сортування:

1) за спаданням дати;

2) за зростанням дати;

3) за статусом задач;

– показувати виконані задачі;

– приховати виконані задачі.

3.2 Вимоги до технічних і програмних засобів

Вимоги до технічних та програмних засобів визначають мінімальні характеристики, які повинні мати клієнтські пристрої для коректної роботи вебзастосунку. Цей пункт деталізує необхідні вимоги до обладнання та програмного забезпечення, які впливають на продуктивність, функціональність та ефективність вебзастосунку.

Важливо враховувати, що правильно підібрані вимоги дозволяють забезпечити оптимальну роботу застосунку на різних пристроях і підтримувати зручний досвід користувача.

Вимоги до технічних засобів:

– процесор Intel Core i3-8100F з тактовою частотою, ГГц – 3,6 або еквівалентний;

– оперативна пам'ять: DDR3 з об’ємом мінімум 4 Гб;

– засоби виводу інформації: монітор з роздільною здатністю не менше 1280x800 пікселів;

– дисковий простір: мінімум 1 ГБ вільного місця на жорсткому диску.

Вимоги до програмного забезпечення:

– веббраузер: останні версії Google Chrome, Mozilla Firefox, Microsoft Edge, Safari або аналогічних;

– інтернет-з'єднання: Стабільне з'єднання з Інтернетом для завантаження та використання вебзастосунку;

– підтримка JavaScript: увімкнена підтримка веббраузером для виконання скриптів JavaScript;

– операційна система: Windows 10 або вище, macOS 10.13 або вище, Linux (Ubuntu, Fedora тощо).

У технічному завдані (додаток А) наведено більш докладний опис усіх вимог.

3.3 Розробка архітектури програми

Розробка архітектури програми є важливим етапом у процесі створення вебзастосунку. Архітектура програми визначає структуру, компоненти та взаємодію між ними, що визначає робочі процеси, ефективність та масштабованість застосунку.

У цьому пункті буде розглянуто процес розробки архітектури програми, визначено ключові компоненти та їх взаємозв'язки, а також обговорено стратегії та підходи до створення ефективної та масштабованої архітектури для нашого вебзастосунку.

UML (Unified Modeling Language) діаграми дуже популярні та зручні діаграми для проєктування, саме вони будуть в подальшому використані при проєктуванні вебзастосунку.

3.3.1 Вибір шаблону проєктування

Вибір шаблону проєктування є одним з ключових етапів у процесі розробки програмного забезпечення, оскільки це визначає загальну структуру та організацію коду. Шаблони проєктування є важливим

інструментом для забезпечення ефективності, розширюваності та обслуговуваності програмного забезпечення.

У цьому розділі буде розглянуто різні шаблони проєктування, їх призначення та основні принципи застосування. Також буде обговорено, який шаблон краще відповідає потребам вебзастосунку та визначено стратегії його впровадження.

Найбільш універсальними та високорівневими є архітектурні шаблони проєктування. Їх можна застосувати практично в будь-якій мові програмування. На відміну від інших видів шаблонів їх можна використовувати для проєктування архітектури програми у цілому [14].

Завдяки вибору мови програмування C# та фреймворку ASP.NET Core для побудови архітектури вебзастосунку ми можемо обрати один з наступних архітектурних підходів або фреймворків:

– ASP.NET Core Blazor;
– ASP.NET Core MVC;
– ASP.NET Core Razor Pages;
– ASP.NET Core Single Page Applications (SPA) з фронтенд JavaScript фреймворком [15].

Blazor — це вебфреймворк .NET, який підтримує як рендеринг на стороні сервера, так і на стороні клієнта в одній моделі програмування [16].

MVC (Model-View-Controller) – це архітектурний шаблон. Шаблон MVC розділяє додаток на три основні групи компонентів: моделі, представлення та контролери. Запити користувачів направляються контролеру. Контролер відповідає за роботу з моделлю для виконання дій користувача або отримання результатів запитів. Контролер вибирає представлення для відображення користувачеві і надає йому будь-які необхідні йому дані моделі [15].

Razor Pages - це модель на основі сторінок для побудови користувацького інтерфейсу на стороні сервера. Інтерфейс сторінки користувача динамічно відображається на сервері для генерації HTML-

сторінки та CSS у відповідь на запит браузера. Сторінка надходить до клієнта, готова до показу. Підтримка Razor Pages побудована на ASP.NET Core MVC [15].

Порівняємо доступні архітектурні підходи до побудови вебзастосунку (таблиця 3.1):

Таблиця 3.1 – Порівняння архітектурних підходів [15]

Архітектурний підхід	Переваги	Недоліки
Blazor	Можливість будування спільної логіки як для коду на сервері так і для коду на клієнті одною мовою програмування	Менше можливостей для логіки на клієнті порівняно з JavaScript
MVC	Реалізує популярний архітектурний шаблон, чітке розмежування відповідальностей, максимальна гнучкість серверної логіки	Тільки серверна логіка
Razor Pages	Шаблон схожий на MVC, чітке розмежування відповідальностей між сторінками з моделями Razor Pages	Тільки серверна логіка
SPA з фронтенд JavaScript фреймворком	Можливість будування спільної логіки як для коду на сервері так і для коду на клієнті	Потреба у більшої кількості знань фреймворків

Порівнявши доступні варіанти архітектурних підходів та з огляду на вибрані мови програмування (C#, JavaScript) було прийнято рішення використовувати популярний архітектурний підхід що дозволяє чітко розділити обов’язки серверної логіки та має максимальну гнучкість – MVC. Саме цей архітектурний підхід реалізований у фреймворку ASP.NET Core MVC. За клієнтську частину буде відповідати попередньо вибрана мова програмування – JavaScript.

3.3.2 UML-діаграма розгортання

Програмне забезпечення має наступні елементи найвищого рівня:

– вебсервер;
– сервер бази даних;
– клієнт.

Вебсервер – це сервер, що приймає HTTP (Hypertext Transfer Protocol) запити від клієнтів, зазвичай веббраузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними [17].

Сервер баз даних виконує обслуговування та управління базою даних та відповідає за цілісність та збереження даних, а також забезпечує операції введення-виведення при доступі клієнта до інформації [18].

Клієнт – апаратний або програмний компонент обчислювальної системи, який надсилає запити серверу [19].

Взаємодію цих елементів ПО найвищого рівня за допомогою діаграми розгортання продемонстровано на рисунку 3.1. Також продемонстровано елементи більш низького рівня – артефакти, що використовуються елементами вищого рівня. Відповідно: вебсервер має вебзастосунок, сервер баз даних – базу даних, клієнтський браузер – HTML.

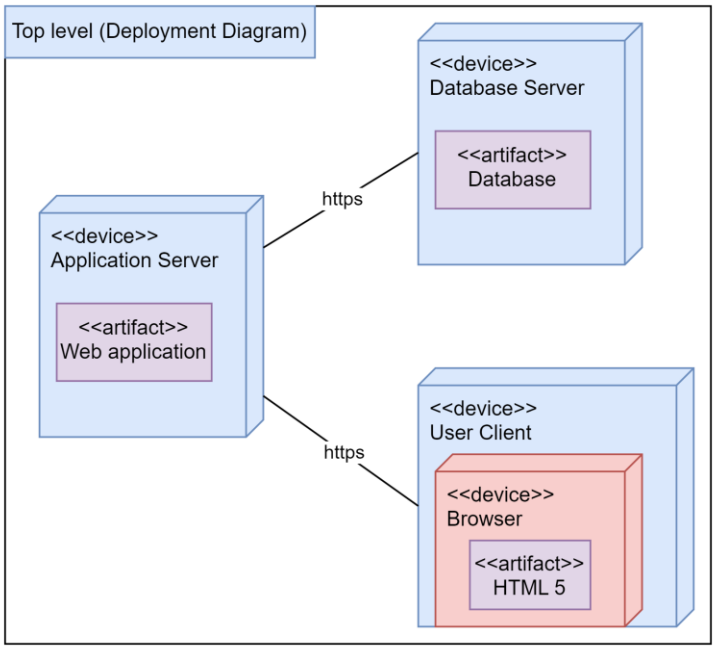

Рисунок 3.1 – Взаємодія елементів ПО найвищого рівня

3.3.3 UML-діаграми компонентів

Взаємодію основних складових компонентів вебзастосунку за допомогою діаграми компонентів продемонстровано на рисунку 3.2. До списку цих компонентів входять:

– MVC units;

– Services;

– Web application configuration.

Компонент «MVC units» абстрагує всі структурні елементи архітектури MVC розглянутої у пункті 3.1.1. Взаємодію цих елементів продемонстровано на рисунку 3.3.

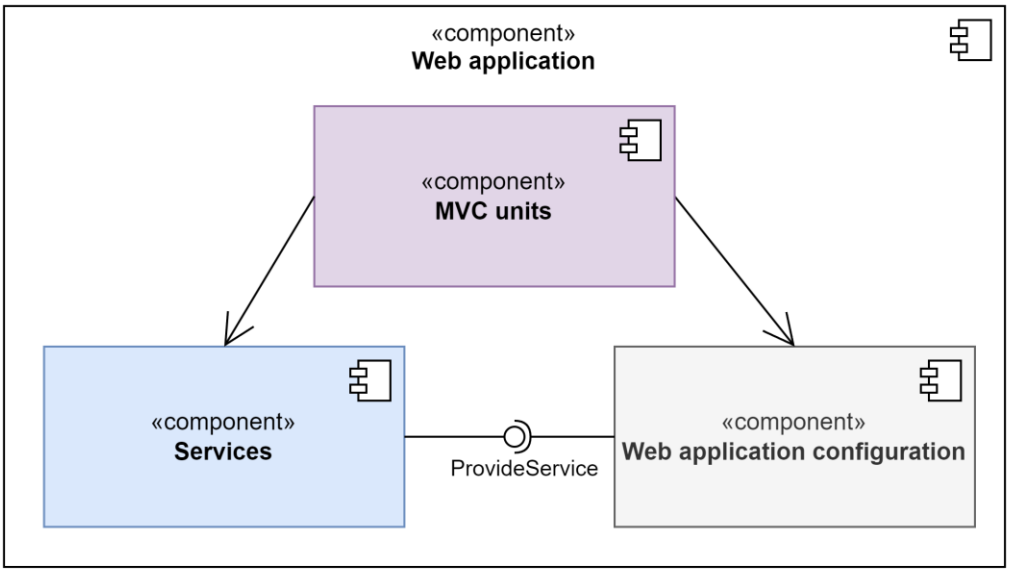

Рисунок 3.2 – Взаємодія основних складових компонентів вебзастосунку

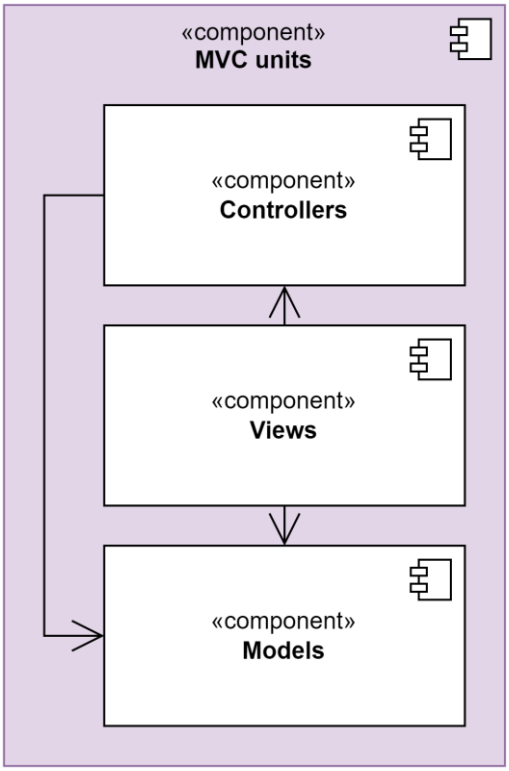

Рисунок 3.3 – Компонент «MVC units»

Компонент «Web application configuration» відповідає за конфігурацію та впровадження залежностей у вебзастосунок.

Компонент «Services» абстрагує залежності що має застосунок. Компонент «Services» та його підкомпоненти продемонстровані на рисунку 3.4. Він містить наступні підкомпоненти:

– «Email sender», сервіс призначений для надсилання листів на електрону пошту.;

– «Logger», сервіс призначений для ведення журналу особливих подій у програмі;

– «Authentication», сервіс призначений для здійснення автентифікації;

– «Authorization», сервіс призначений для здійснення авторизації;

– «Database context», сервіс призначений для управління даними.

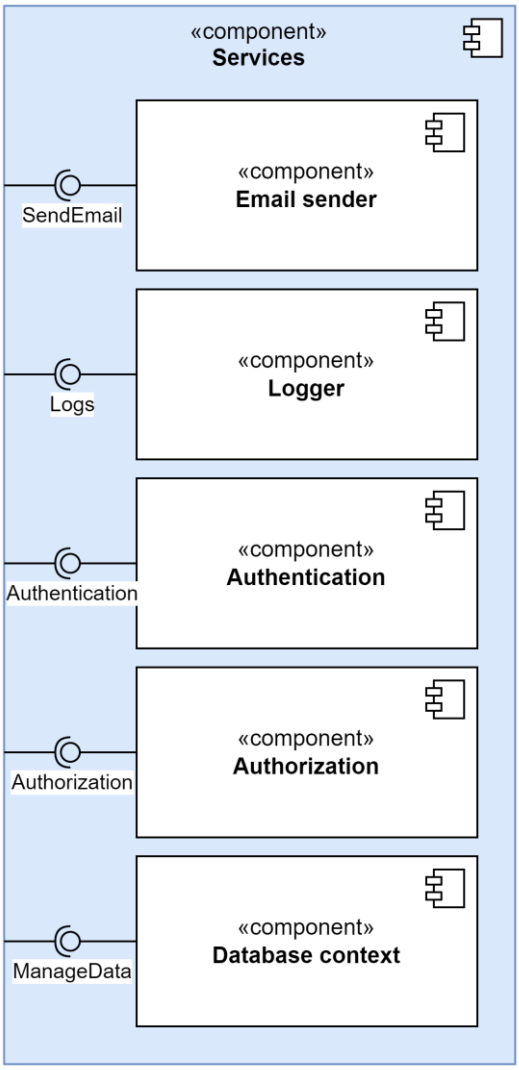

Рисунок 3.3 – Компонент «Services»

Також у архітектурі вебзастосунку можна виділити основні одиниці навколо котрих будується вся логіка, їх сукупність названа компонентом «Essential units» (рисунок 3.4), в склад якого входить:

– Tasks;
– Notes;
– Projects;
– Labels;
– Account;
– User manage.

Компонент «Account» агрегує дії пов’язані автентифікації.

Компонент «User manage» агрегує дії пов’язані з акаунтом користувача.

Компоненти: «Tasks», «Notes», Projects, Labels агрегують дії пов’язані відповідно з задачами, нотатками, проєктами, мітками.

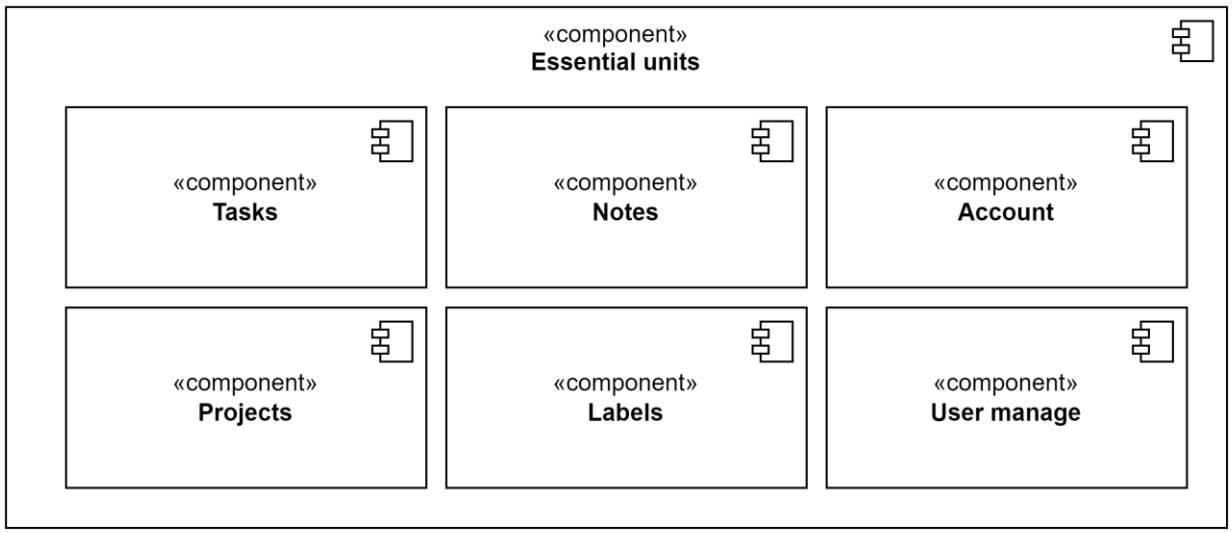

Рисунок 3.4 – Компонент «Essential units»

3.3.4 UML-діаграми прецедентів

Задля більш чіткого розуміння взаємодії користувача з вебзастосунком розглянемо діаграму прецедентів (рисунок 3.5). Ця діаграма

демонструє функціональні вимоги «верхнього» рівня розглянуті у пункті 3.1 як варіанти взаємодії користувача з системою. Також бачимо зв’язок дій користувача у вебзастосунку з сервером бази даних.

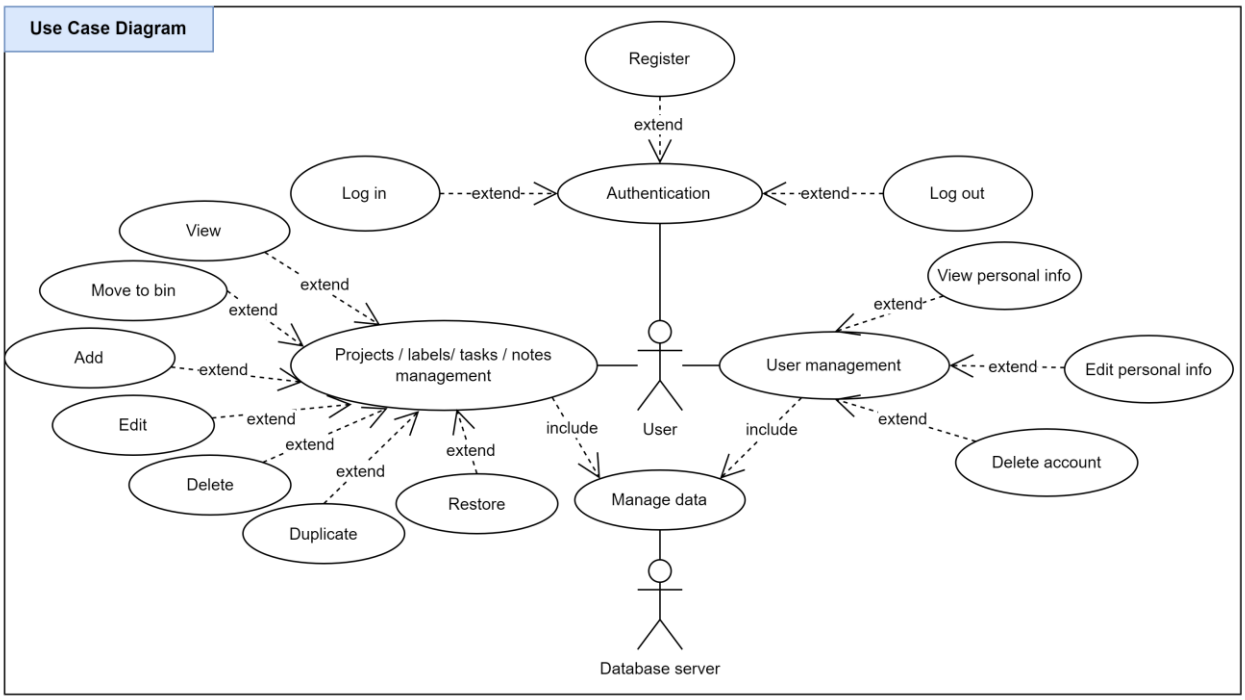

Рисунок 3.5 – Діаграма прецедентів

3.3.5 UML-діаграми станів

Основні одиниці навколо котрих будується вся логіка вебзастосунку були розглянуті за допомогою діаграми компонентів у пункті 3.3.3. Задля більш чіткого розуміння в яких станах можуть перебувати ці елементи розглянемо їх діаграми станів (рисунок 3.6 – 3.8). На рисунку 3.6 зображено наступні стани задачі:

– «Created», створена задача;
– «Undone», задача помічена як невиконана;
– «Done», задача помічена як виконана;
– «Moved to bin», задача переміщена до кошику;
– «Restored», задача що була відновлена;
– «Deleted forever», задача що була видалена назавжди.

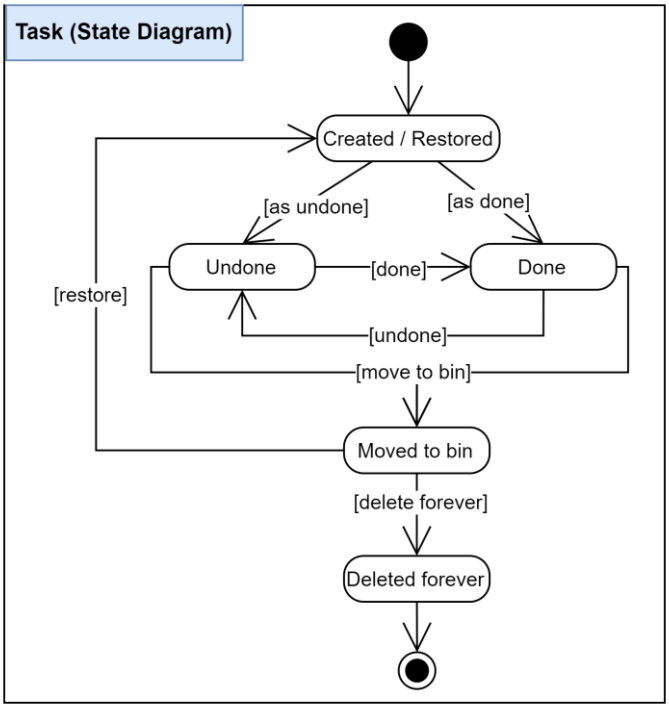

Рисунок 3.6 – Діаграма станів задачі

Стани нотаток та проєктів співпадають тому на рисунку 3.7 зображено наступні спільні стани:

– «Created», створено;
– «Moved to bin», переміщено до кошику;
– «Restored», було відновлено;
– «Deleted forever», було видалено назавжди.

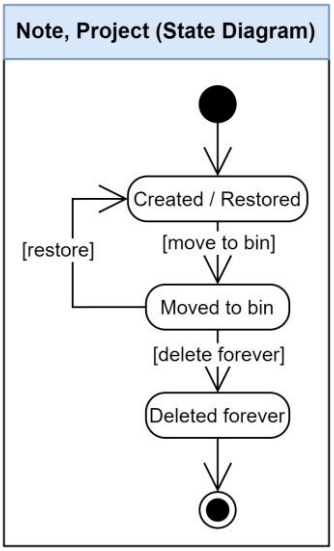

Рисунок 3.7 – Діаграма станів нотатки та проєкту

Акаунт користувача містить наступні стани (рисунок 3.8):

– «Created», створений користувач;
– «Unconfirmed», створений користувач не підтверджений;
– «Confirmed», створений користувач підтверджений;
– «Not authenticated», користувач не автентифікований;
– «Authenticated», користувач автентифікований;
– «Deleted forever», користувач був видалений назавжди.

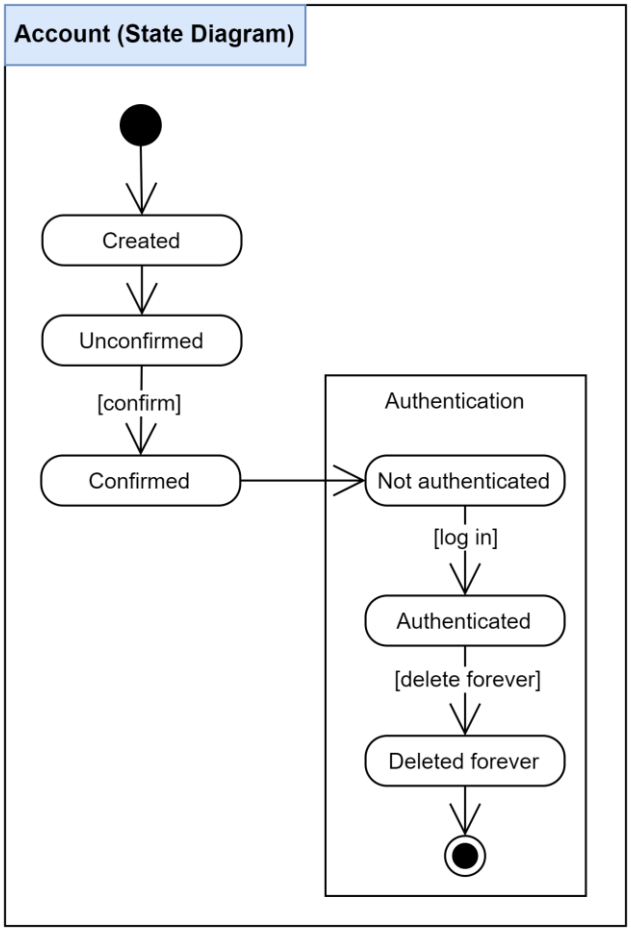

Рисунок 3.8 – Діаграма станів акаунту

3.3.6 Проєктування бази даних

Для автентифікації у вебзастосунку буде використано ASP.NET Core Identity – API який підтримує функцію автентифікації користувача. Він дозволяє керувати: користувачами, паролями, даними профілю, ролями,

підтвердженням електронної пошти тощо. Впровадження цієї залежності дозволить згенерувати таблиці відповідальні за автентифікацію користувача. Однією з цих таблиць є «AspNetUsers» що відповідальна за збереження основної інформації про користувача, такої як: пошта, пароль тощо [20]. Таким чином це дозволяє сфокусувати увагу на інших моделях вебзастосунку що стосуються бізнес логіки.

Сутність «User» (CustomName) – сутність що доповнює згенеровану ASP.NET Core Identity таблицю «AspNetUsers». Ця сутність додає ім'я користувача. Опис полів сутності та їх типу наведено у таблиці 3.1.

Таблиця 3.1 – Опис полів сутності «User»

№	Назва поля	Тип даних
1	CustomName	nvarchar

Сутність «Project» (Projectld, Userld, Title, CreatedDate, lsDeleted, IsDefault) – сутність що представляє собою проєкт користувача. Опис полів сутності та їх типу наведено у таблиці 3.2.

Таблиця 3.2 – Опис полів сутності «Project»

№	Назва поля	Тип даних
1	Projectld	int
2	Userld	nvarchar
3	Title	nvarchar
4	CreatedDate	datetime2
5	lsDeleted	bit
6	IsDefault	bit

Сутність «Label» (Labelld, Userld, Title, lsDeleted) – сутність що представляє собою мітку користувача. Опис полів сутності та їх типу наведено у таблиці 3.3.

Таблиця 3.3 – Опис полів сутності «Label»

№	Назва поля	Тип даних
1	LabelId	int
2	UserId	nvarchar
3	Title	nvarchar

Сутність «Task» (TaskId, ProjectId, Title, Description, lsCompleted, IsDeleted, CreatedAt, DueDate, DueTime) – сутність що представляє собою задачу користувача. Опис полів сутності та їх типу наведено у таблиці 3.4.

Таблиця 3.4 – Опис полів сутності «Task»

№	Назва поля	Тип даних
1	TaskId	int
2	ProjectId	int
3	Title	nvarchar
4	Description	nvarchar
5	lsCompleted	bit
6	IsDeleted	bit
7	CreatedAt	datetime2
8	DueDate	datetime2
9	DueTime	datetime2

Сутність «Note» (NoteId, ProjectId, NoteDescriptionId, Title, ShortDescription, IsDeleted, CreatedAt, DueDate, DueTime) – сутність що представляє собою нотатку користувача. Опис полів сутності та їх типу наведено у таблиці 3.5.

Таблиця 3.5 – Опис полів сутності «Note»

№	Назва поля	Тип даних
1	NoteId	int
2	ProjectId	int
3	NoteDescriptionId	int
4	Title	nvarchar
5	ShortDescription	nvarchar
6	IsDeleted	bit
7	CreatedAt	datetime2
8	DueDate	datetime2
9	DueTime	datetime2

Сутність «NoteDescription» (NoteDescriptionId, NoteId, Description) – сутність що є доповненням до сутності «Note» реалізованих за допомогою зв’язку один до одного. Опис полів сутності та їх типу наведено у таблиці 3.6.

Таблиця 3.6 – Опис полів сутності «NoteDescription»

№	Назва поля	Тип даних
1	NoteDescriptionId	int
2	NoteId	int
3	Description	int

Сутність «TaskLabel» (TaskLabelId, TaskId, LabelId) – сутність що дозволяє реалізувати зв’язок багато до багатьох між сутностями «Task» та «Label». Опис полів сутності та їх типу наведено у таблиці 3.7.

Таблиця 3.7 – Опис полів сутності «TaskLabel»

№	Назва поля	Тип даних
1	TaskLabelId	int
2	TaskId	int
3	LabelId	int

Сутність «NoteLabel» (NoteLabelId, NoteId, LabelId) – сутність що дозволяє реалізувати зв’язок багато до багатьох між сутностями «Task» та «Label». Опис полів сутності та їх типу наведено у таблиці 3.8.

Таблиця 3.8 – Опис полів сутності «NoteLabel»

№	Назва поля	Тип даних
1	NoteLabelId	int
2	NoteId	int
3	LabelId	int

Визначивши сутності можна побудувати концептуальну моделі бази даних що відображає їх зв’язки на рисунку 3.9.

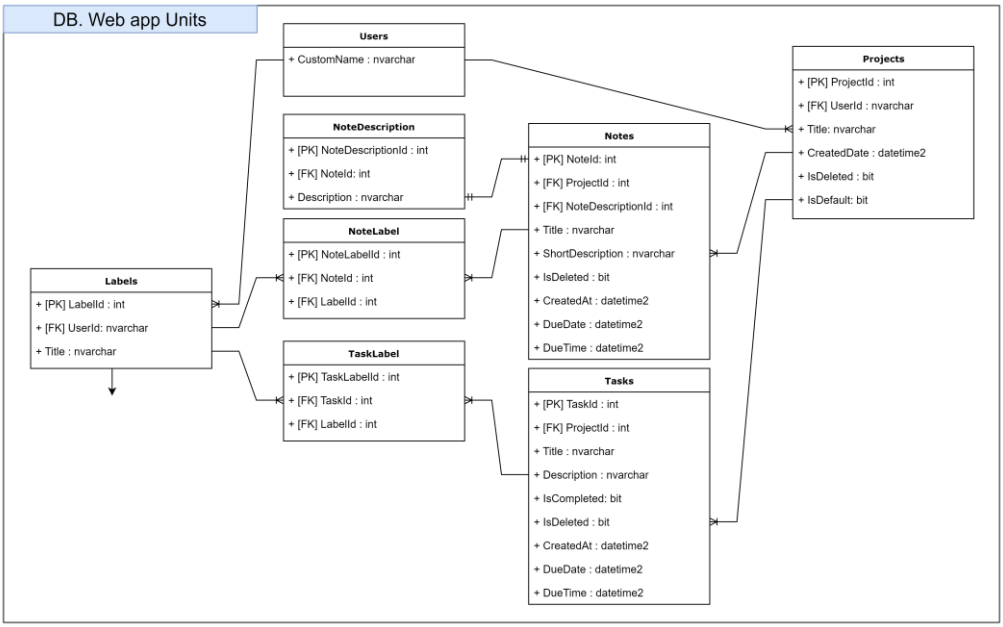

Рисунок 3.9 – Концептуальна модель бази даних

3.3.7 Опис модулів розроблюваної системи

Попередньо у пункті 3.3.3 було визначено основні одиниці навколо котрих будується вся логіка вебзастосунку, їх також можна назвати модулями системи. У таблиці 3.10 наведено опис модулів вебзастосунку з визначенням їх основних функцій.

Таблиця 3.10 – Опис модулів реалізованого ПЗ з визначенням основних функцій

Модуль	Опис
Tasks	Відповідає за управління задачами користувача: створення, редагування, переміщення до кошику, відновлення, виконання, видалення.
Notes	Відповідає за управління нотатками користувача: створення, редагування, переміщення до кошику, відновлення, видалення.
Projects	Відповідає за управління проєктами користувача: створення, редагування, переміщення до кошику, відновлення, видалення.
Labels	Відповідає за управління мітками користувача: створення, редагування, видалення.
Account	Відповідає за автентифікацію користувача: реєстрацію, вхід, вихід з системи.
User manage	Відповідає за змінення або видалення даних пов’язаних з акаунтом користувача.

3.4 Розробка користувацького інтерфейсу

Розробка користувацького інтерфейсу є невід'ємною частиною процесу створення вебзастосунку для управління задачами та нотатками. Цей

етап має вирішальне значення, оскільки інтерфейс визначає спосіб взаємодії користувача з програмою, його комфорт та задоволення від використання застосунку. Спочатку у цьому розділі буде розглянуто вайрфрейми, після чого буде інтерфейс буде описано більш детально за допомогою прототипів.

3.4.1 Базова структура інтерфейсу

Вайрфрейм зображує макет сторінки або розміщення вмісту вебсайту, включаючи елементи інтерфейсу та навігаційні системи, а також те, як вони працюють разом. Вайрфрейм зазвичай не має типографічного стилю, кольору чи графіки, оскільки основна увага приділяється функціональності, поведінці та вмісту. Іншими словами, він зосереджується на тому, що робить екран, а не на тому, як він виглядає. Каркаси можуть бути малюнками олівцем чи ескізами на дошці, або вони можуть створюватися за допомогою широкого спектру безкоштовних або комерційних програм [21].

Сторінка для входу або реєстрації користувача має секцію з формою для вводу даних та розташований збоку малюнок (рисунок 3.10).

Рисунок 3.10 – Вайрфрейм сторінки входу або реєстрації

Сторінка що відображує інформацію стосовно процесу автентифікації містить секцію з інформацією про процес та малюнок (рисунок 3.11). Прикладом такої сторінки може бути сторінка підтвердження електронної пошти.

Рисунок 3.11 – Вайрфрейм сторінки інформації стосовно процесу автентифікації

Головна сторінка (рисунок 3.12) містить наступні основні елементи інтерфейсу:

– хедер, містить навігаційні кнопки та профіль користувача;

– бокова панель, містить подання;

– основна частина, містить задачі та нотатки у двох окремих стовпчиках.

Рисунок 3.12 – Вайрфрейм головної сторінки

Сторінка для управління мітками та сторінка кошику відрізняється від головної сторінки кількістю стовпців (рисунок 3.13).

Рисунок 3.13 – Вайрфрейм сторінки міток або кошику

3.4.2 Шрифти та іконки

Google Fonts – бібліотека понад 950 вільно розповсюджуваних шрифтів, інтерактивний каталог для їх перегляду і прикладні програмні інтерфейси для використання вебшрифтів за допомогою CSS і на Андроїд [22]. Також ця бібліотека містить категорію іконок які також будуть використані при розробці вебзастосунку. Шрифтом за замовчуванням буде популярний шрифт Inter.

Також було вибрано наступні розміри шрифтів:

– 16, основний шрифт;
– 20, заголовки модальних вікон;
– 24, заголовки подань;
– 50, заголовки сторінок автентифікації.

3.4.3 Прототипи модальних вікон

Попередньо у пункті 3.3.3 було визначено основні одиниці навколо котрих будується вся логіка вебзастосунку. Ці дії будуть відбуватися за допомогою модальних вікон.

У графічному інтерфейсі користувача модальним називається вікно, що блокує роботу користувача з батьківським застосунком доти, доки користувач це вікно не закриє. У вигляді модальних переважно реалізуються діалогові вікна. Також модальні вікна часто використовуються для привернення уваги користувача до важливої події чи критичної ситуації [23].

Розглянемо прототип модального вікна створення задачі (рисунок 3.14). На рисунку ми можемо побачити:

– «Create», заголовок модального вікна;
– «Unsorted», випадаючий список для вибору проєкту задачі;
– «Task name», поле для введення назви задачі;
– «Description», поле для введення опису задачі

– «Date/Time», поле для вибору дати задачі;

– «Labels», випадаючий список для вибору міток задачі;

– кружечок зліва від назви, дозволяє змінювати статус виконання задачі;

– хрестик з справа зверху, кнопка закриття вікна;

– «Ok», кнопка відправки форми;

– «Cancel», кнопка закриття вікна.

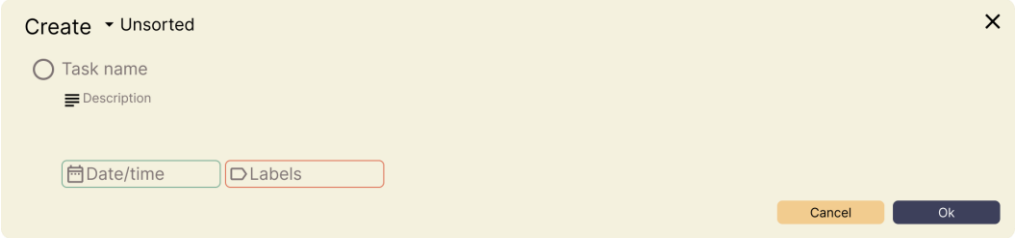

Рисунок 3.14 – Прототип модального вікна створення задачі

У прототипі порівняно з вайрфреймом ми вже практично бачимо картину того як буде виглядати вебзастосунок.

Розглянемо прототип модального вікна редагування задачі (рисунок 3.15). Окрім всіх елементів що було на модальному вікні створення задачі тут додається кнопка-трикрапка що викликає контекстне меню з доступними діями з задачею. Також при вибраній даті можна обирати час у полі «Time».

Рисунок 3.15 – Прототип модального вікна редагування задачі

За таким самим принципом розроблені модальні вікна:

– створення, редагування, видалення задачі;

– створення, редагування, видалення нотатки;

– створення, редагування, видалення проєкту;

– створення, редагування, видалення проєкту;

– змінення імені, змінення пошти, змінення паролю, встановлення паролю, видалення акаунту.

3.4.4 Прототипи сторінок

У пункті 3.4.1 було розглянуто базову структуру інтерфейсу за допомогою вайрфреймів. Було наведено наступні загальні шаблони:

– сторінка для входу або реєстрації;

– сторінка що відображує інформацію стосовно процесу автентифікації;

– головна сторінка;

– сторінка для управління мітками та сторінка кошику.

Розглянемо ці сторінки більш детально за допомогою прототипів. Розпочнемо зі сторінки реєстрації (рисунок 3.16). Порівняно з вайрфреймом ми можемо бачити конкретну назву заголовку, конкретні назви та іконки полів для введення. Також можна побачити логотип вебзастосунку зліва зверху.

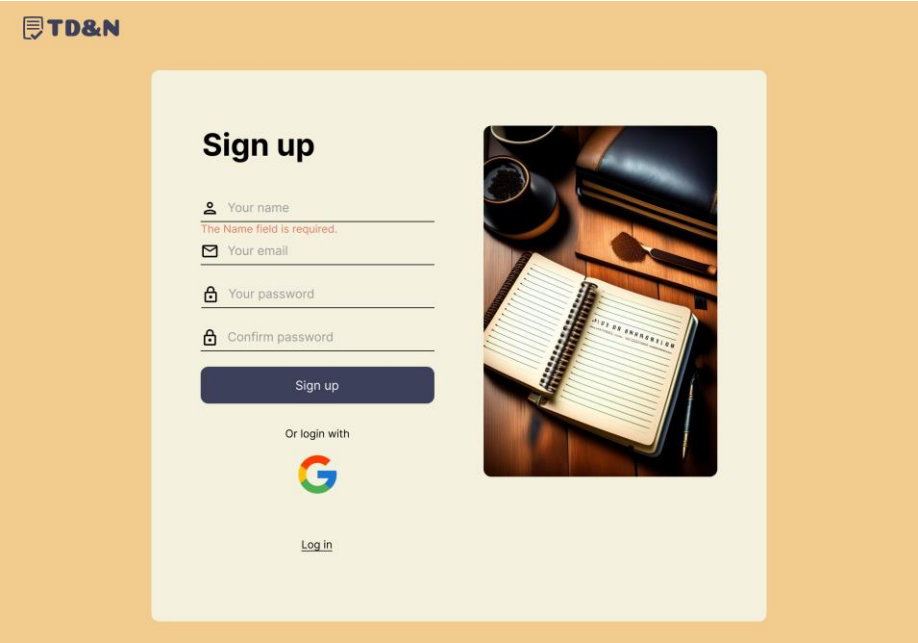

Рисунок 3.16 – Прототип сторінки реєстрації

Другим шаблоном було розглянуто сторінку що відображує інформацію стосовно процесу автентифікації. Прикладом такої сторінки може бути сторінка із запрошенням перевірити пошту (рисунок 3.17). Порівняно з вайрфреймом ми бачимо конкретне повідомлення пов’язане з процесом автентифікації.

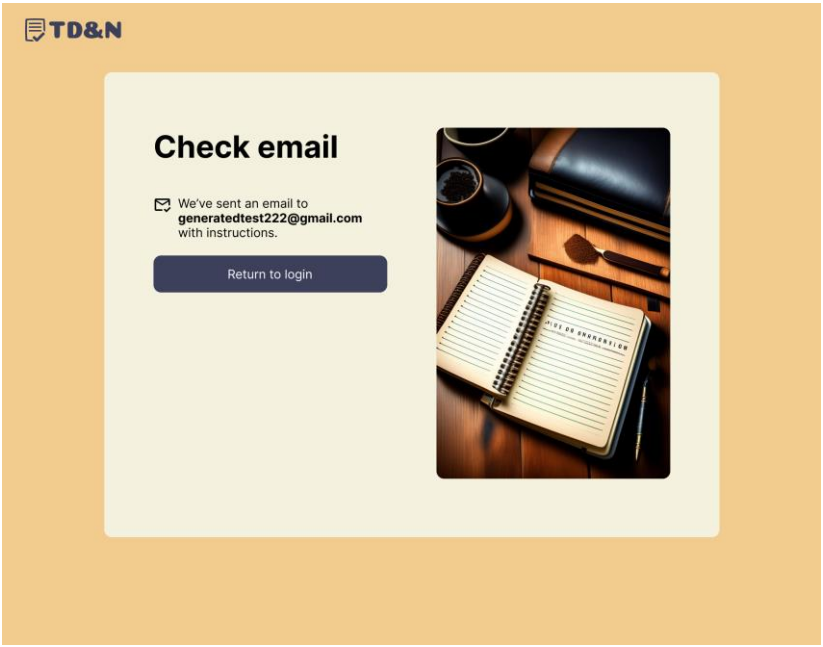

Рисунок 3.17 – Прототип сторінки перевірки пошти

Третім шаблоном було розглянуто головну сторінку. Прикладом такої сторінки може бути подання «Today» головної сторінки (рисунок 3.18). Порівняно з вайрфреймом у прототипі ми можемо бачити:

– назви подань;
– назви проєктів;
– списки задач та нотаток на сьогодні.

Також ми можемо бачити компактне розміщення даних задач та нотаток які заповнюються користувачами у модальних вікнах розглянутих у пункті 3.4.3.

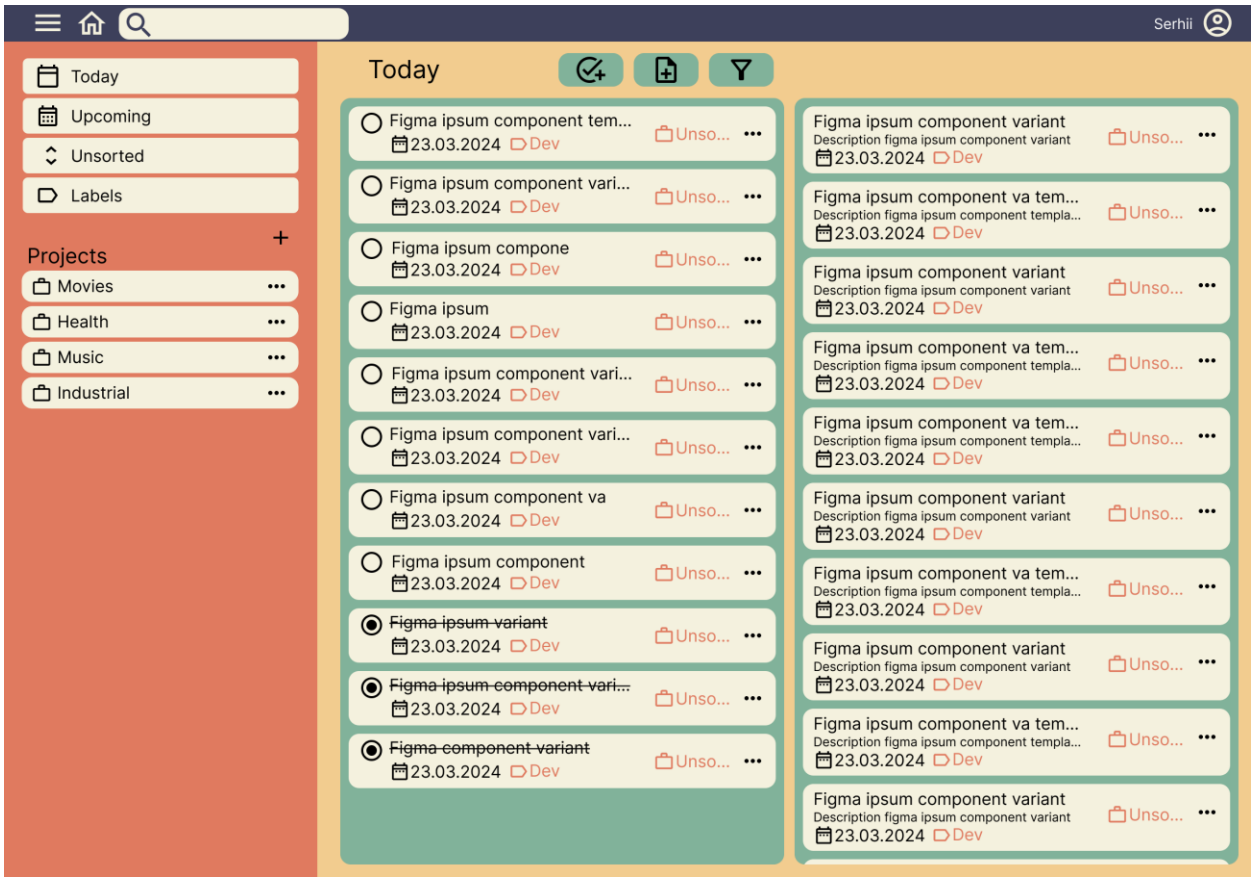

Рисунок 3.18 – Прототип подання «Today» головної сторінки

Останнім шаблоном було розглянуто сторінку для управління мітками та сторінка кошику. Прикладом такої сторінки є та сама сторінка кошику (рисунок 3.19). На цій сторінці ми бачимо видалені проєкти, задачі та нотатки. Також на рисунку продемонстровано приклади контекстних меню.

Рисунок 3.19 – Прототип сторінки кошику

Також було розроблено логотип вебзастосунку (рисунок 3.20) за допомогою сервісу «LOGO.com» що дозволяє створювати логотипи для бізнесів [24].

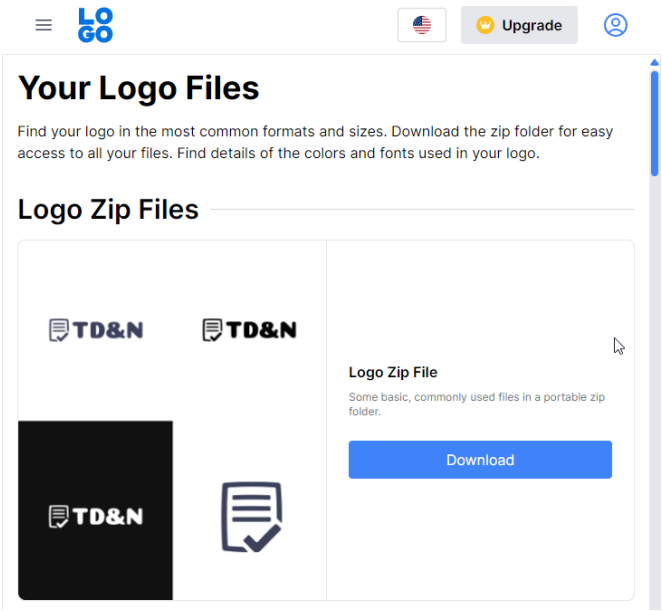

Рисунок 3.20 – Розроблений логотип вебзастосунку у сервісі «LOGO.com»

3.5 Висновки за розділом 3

У третьому розділі було:

– описано основні вимоги до програми;
– розроблено архітектуру програми;
– розроблено користувацький інтерфейс.

Після чого стала можлива реалізація вебзастосунку.

4 РОЗРОБКА ВЕБЗАСТОСУНКУ

У цьому розділі буде описано основні алгоритми програм за допомогою UML-діаграм послідовності. Також буде наведено більш детальний опис розроблених модулів системи.

4.1 Алгоритми та UML-діаграми послідовності

4.1.1 Алгоритм автентифікації

Користувач може здійснювати реєстрацію у вебзастосунку. Для цього він може використати реєстрацію за допомогою електронної адреси або зовнішній провайдер Google. Алгоритм також передбачає обробку помилок в ході валідації даних.

Зареєстрований користувач може здійснювати вхід в застосунок. Якщо користувач не зареєстрований то вхід в систему завершиться помилкою. Варіант входу користувача в систему залежить від того яким методом він реєструвався, інакше виникне помилка. Якщо користувач при реєстрації використовував вхід за допомогою Google то вхід в систему виконується так само через Google. Якщо користувач при реєстрації використовував пошту то вхід в систему виконується за допомогою пошти та паролю.

Алгоритм реєстрації та входу користувача у вебзастосунок наведено за допомогою діаграми послідовності на рисунку 4.1.

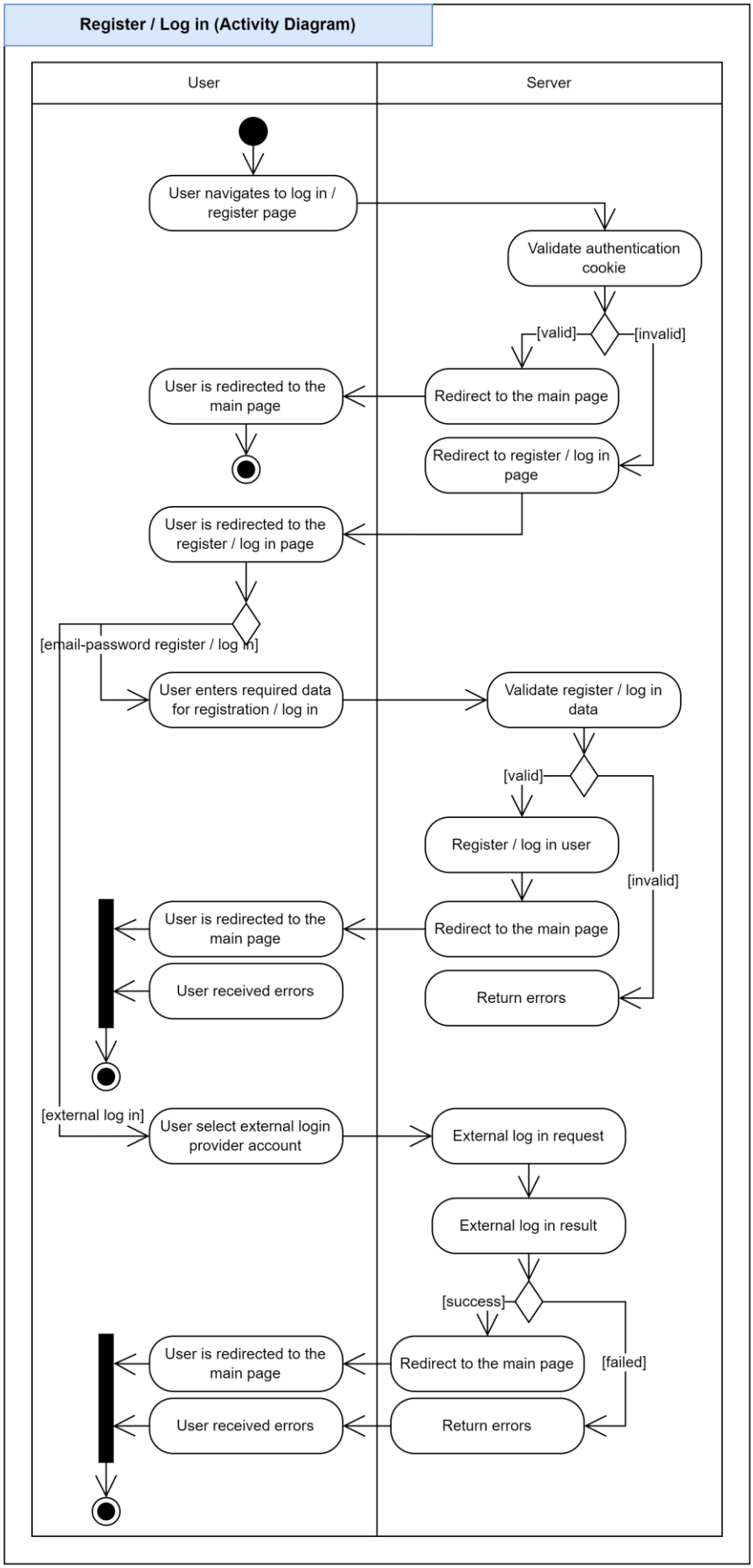

Рисунок 4.1 – Алгоритм реєстрації та входу

4.1.2 Алгоритм авторизації

Вебзастосунок має два види авторизації:

– на основі атрибутів;

– на основі ресурсів.

Авторизація на основі атрибутів дозволяє обмежити доступ до контролера або його методів. У ASP.NET Core MVC таким атрибут є [Authorize]. Прикладами обмежень які накладає атрибут можуть бути наступні:

– для автентифікованих користувачів;

– для автентифікованих користувачів певної ролі [25].

Авторизація на основі ресурсів дозволяє надавати контрольований доступ до ресурсів. Наприклад, до проєкту власником якого є один користувач не зможе отримати доступ інший користувач [26].

Процес авторизації можна узагальнити наступним алгоритмом на діаграмі послідовності (рисунок 4.2). Користувач запитує доступ до ресурсів, а сервер йому відповідає наданим або ненаданим доступом до ресурсу.

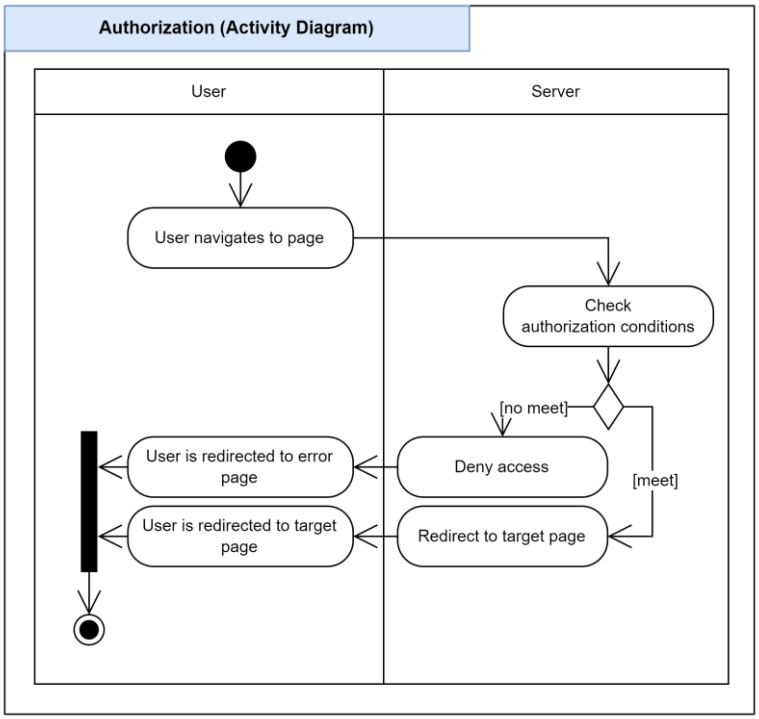

Рисунок 4.2 – Алгоритм авторизації

4.1.3 Алгоритми створення, редагування та видалення

Попередньо у пункті 3.3.3 було визначено основні одиниці навколо котрих будується вся логіка вебзастосунку. Дії над цими елементами будуть відбуватися за наступним алгоритмом (рисунок 4.3). Користувач за допомогою інтерфейсу вибирає дію, сервер в свою чергу реагує на запит та виконує його. В разі невиконання запиту відображається помилка.

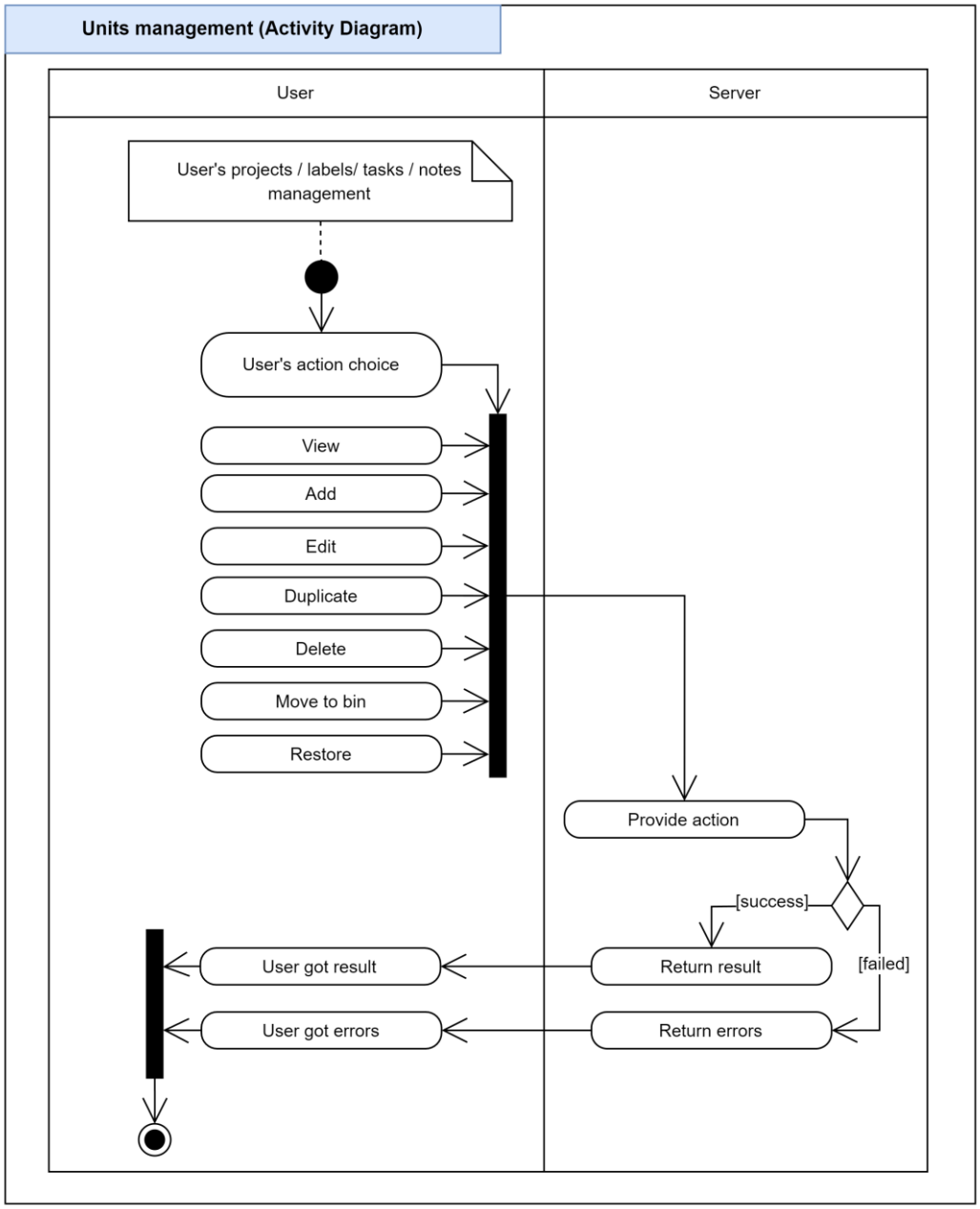

Рисунок 4.3 – Алгоритм авторизації

4.2 Опис модулів розроблюваної системи

4.2.1 Моделі

Попередньо у пункті 3.3.3 було визначено основні одиниці навколо котрих будується вся логіка вебзастосунку, моделі є елементами що зберігають певну інформацію що необхідна для цих основних одиниць.

Клас «User» – клас, що представляє собою користувача вебзастосунку. Опис полів та методів класу наведені у таблиці 4.1.

Попередньо у пункті 3.3.6 при проєктуванні бази даних було прийнято рішення використовувати ASP.NET Core Identity для автентифікації. Це надає нам можливість використовувати вбудовані у це API класи що перетворюється у таблиці у базі даних. Було визначено що, наприклад, таблиця «AspNetUsers» відповідає за збереження даних користувача. Цій таблиці відповідає класс IdentityUser [27]. Для того щоб додати певні дані до цього класу було прийнято рішення створити власний клас «User» що буде наслідуватися від класу IdentityUser.

Таблиця 4.1 – Опис полів та методів класу «User»

Поля та методи класу	Опис
Is-A:	
IdentityUser	Клас ASP.NET Core Identity що містить базові дані про користувача.
public:	
string? CustomName	Ім’я користувача.
ICollection<Project>? Projects	Список проєктів користувача.
ICollection<Label>? Labels	Список міток користувача.

Клас «Project» – клас, що представляє собою проєкт користувача. Опис полів та методів класу наведені у таблиці 4.2.

Таблиця 4.2 – Опис полів та методів класу «Project»

Поля та методи класу	Опис
public:	
int? ProjectId	Унікальний ідентифікатор проєкту.
string? UserId	Зовнішній унікальний ідентифікатор користувача.
User? User	Екземпляр що представляє користувача.
string? Title	Назва проєкту.
DateTime? CreatedDate	Дата створення проєкту.
bool IsDeleted { get; set; }	Змінна стану проєкту (видалений або ні).
bool IsDefault { get; set; }	Змінна стану проєкту (проєкт за замовченням або ні).
ICollection<Task>? Tasks	Список задач проєкту.
ICollection<Note>? Notes	Список нотаток проєкту.

Клас «Label» – клас, що представляє собою мітку користувача. Опис полів та методів класу наведені у таблиці 4.3. Ця модель містить у собі списки проміжних сутностей «NoteLabel» та «TaskLabel» що будуть описані далі у розділі.

Таблиця 4.3 – Опис полів та методів класу «Label»

Поля та методи класу	Опис
public:	
int? LabelId { get; set; }	Унікальний ідентифікатор мітки.
string? UserId { get; set; }	Зовнішній унікальний ідентифікатор користувача.

Продовження таблиці 4.3

Поля та методи класу	Опис
public:	
User? User { get; set; }	Екземпляр що представляє користувача.
string? Title	Назва проєкту.
DateTime? CreatedDate	Дата створення проєкту.
ICollection<NoteLabel>? NoteLabels	Список проміжних сутностей міток задачі.
ICollection<TaskLabel>? TaskLabels	Список проміжних сутностей міток нотатки.

Клас «TdnSortElement» – абстрактний клас, що представляє собою об’єкт що підлягає сортуванню. Опис полів та методів класу наведені у таблиці 4.4.

Колекція екземплярів «TdnSortElement» дозволяє містити в собі екземпляри інших класів що мають «TdnSortElement» як базовий клас. Цей клас надає список полів за допомогою яких відбувається сортування.

Таблиця 4.4 – Опис полів та методів класу «TdnSortElement»

Поля та методи класу	Опис
public:	
virtual DateOnly? DueDate	Термін (дата).
virtual TimeOnly? DueTime	Термін (час).
virtual bool IsCompleted	Статус (виконано, невиконано).

Клас «Task» – клас, що представляє собою задачу користувача. Опис полів та методів класу наведені у таблиці 4.5.

Ця модель містить у собі список проміжних сутностей «TaskLabel» що будуть розглянуті далі у розділі.

Ця модель наслідується від класу «TdnSortElement», тобто містить дані необхідні для сортування.

Таблиця 4.5 – Опис полів та методів класу «Task»

Поля та методи класу	Опис
Is-A:	
TdnSortElement	Клас, що містить дані для сортування.
public:	
int? TaskId	Унікальний ідентифікатор задачі.
int? ProjectId	Зовнішній унікальний ідентифікатор проєкту.
Project? Project	Екземпляр що представляє проєкт.
string? Title	Назва задачі.
string? Description	Опис задачі.
override bool IsCompleted	Змінна статусу задачі (виконана або ні).
bool IsDeleted	Змінна статусу задачі (видалена або ні).
DateTime? CreatedAt	Дата створення задачі.
override DateOnly? DueDate	Термін виконання задачі (дата).
override public TimeOnly? DueTime	Термін виконання задачі (час).
ICollection<TaskLabel>? TaskLabels	Список проміжних сутностей міток задачі.

Клас «Note» – клас, що представляє собою нотатку користувача. Опис полів та методів класу наведені у таблиці 4.6.

Ця модель містить у собі список проміжних сутностей «NoteLabel» що будуть розглянуті далі у розділі.

Ця модель наслідується від класу «TdnSortElement», тобто містить дані необхідні для сортування.

Ця модель містить зв’язок «один до одного» з моделлю «NoteDescription» що дозволяє розділити опис нотатки на дві таблиці задля прискорення завантаження даних.

Таблиця 4.6 – Опис полів та методів класу «Note»

Поля та методи класу	Опис
Is-A:	
TdnSortElement	Клас, що містить дані для сортування.
public:	
int? NoteId	Унікальний ідентифікатор нотатки.
int? ProjectId	Зовнішній унікальний ідентифікатор проєкту.
Project? Project	Екземпляр що представляє проєкт.
string? Title	Назва нотатки.
string? ShortDescription	Короткий опис нотатки.
NoteDescription? NoteDescription	Об’єкт повного опису нотатки.
bool IsDeleted	Змінна статусу задачі (видалена або ні).
DateTime? CreatedAt	Дата створення задачі.
override DateOnly? DueDate	Термін нотатки (дата).
override public TimeOnly? DueTime	Термін нотатки (час).
ICollection<NoteLabel>? NoteLabels	Список проміжних сутностей міток нотатки.
override bool IsCompleted	Змінна статусу нотатки (виконана або ні). Не відображається у базі даних.

Клас «NoteDescription» – клас, що представляє собою повний опис нотатки користувача. Опис полів та методів класу наведені у таблиці 4.7.

Таблиця 4.7 – Опис полів та методів класу «NoteDescription»

Поля та методи класу	Опис
public:	
int? NoteDescriptionId	Унікальний ідентифікатор опису нотатки.
int? NoteId	Зовнішній унікальний ідентифікатор нотатки.
Note? Note	Екземпляр що представляє нотатку.
string? Description	Повний опис нотатки.

Клас «TaskLabel» – клас, що представляє проміжну сутність між класом «Task» та «Label». Опис полів та методів класу наведені у таблиці 4.8.

Ця модель дозволяє реалізувати зв’язок «багато до багатьох» з моделлю «Label», тобто одній задачі може відповідати кілька міток та одній мітці – кілька задач.

Клас «NoteLabel» – клас, що представляє проміжну сутність між класом «Note» та «Label». Опис полів та методів класу наведені у таблиці 4.9. Ця модель так само як реалізує зв’язок «багато до багатьох».

Таблиця 4.8 – Опис полів та методів класу «TaskLabel»

Поля та методи класу	Опис
public:	
int? TaskLabelId	Унікальний ідентифікатор проміжної сутності.
Task? Task	Екземпляр що представляє задачу.
Label? Label	Екземпляр що представляє мітку.

Таблиця 4.9 – Опис полів та методів класу «NoteLabel»

Поля та методи класу	Опис
public:	
int? NoteLabelId	Унікальний ідентифікатор проміжної сутності.
Note? Note	Екземпляр що представляє нотатку.
Label? Label	Екземпляр що представляє мітку.

Перелічення «DaysViewName» – перелічення, що дозволяє спростити роботи з поданнями шляхом їх іменування. Може приймати наступні значення:

– «Today», подання на сьогодні;
– «Upcoming», майбутнє подання;
– «Unsorted», не сортоване подання.

Клас «EntityOperations» – статичний клас, що використовується для авторизації на основі ресурсів. Опис полів та методів класу наведені у таблиці 4.10.

При запиті на ресурс поле цього класу використовується для отримання об’єкту «авторизаційної вимоги», що в подальшому використовується при авторизації.

Таблиця 4.10 – Опис полів та методів класу «EntityOperations»

Поля та методи класу	Опис
public:	
static OperationAuthorizationRequirement FullAccess	Авторизаційна вимога.

Клас «Constants». Опис полів та методів класу наведені у таблиці 4.11.

Авторизаційні вимоги представлені вбудованим класом «OperationAuthorizationRequirement». Цей клас має поле «Name» що представляє назву авторизаційної вимоги. Для спрощення роботи з цим полем і було створено клас «Constants».

Таблиця 4.11 – Опис полів та методів класу «Constants»

Поля та методи класу	Опис
public:	
static readonly string FullAccessOperationName	Назва авторизаційної вимоги.

Клас «IsOwnerAuthorizationHandler» – клас, що призначений для виконання логіки авторизації на основі ресурсів. Він є похідним від вбудованого класу «AuthorizationHandler». Цей клас дозволяє перевірити чи має користувач доступ до запитуваного ресурсу. Опис полів та методів класу наведені у таблиці 4.12. У розділі 1 додатку Б можна побачити код цього класу.

Таблиця 4.12 – Опис полів та методів класу «IsOwnerAuthorizationHandler»

Поля та методи класу	Опис
Is-A:	
AuthorizationHandler<OperationAuthorizationRequirement, object>	Клас, що містить метод для перевизначення авторизації.
private:	
UserManager<Models.User> _userManager	Екземпляр класу для управління користувача.
IsOwnerAuthorizationHandler(UserManager<Models.User> userManager)	Конструктор.

Продовження таблиці 4.12

Поля та методи класу	Опис
private:	
override System.Threading.Tasks.Task HandleRequirementAsync(AuthorizationHandlerContext context, OperationAuthorizationRequirement requirement, object resource)	Перевизначений метод класу «AuthorizationHandler» що виконує авторизацію на основі контексту, вимоги та ресурсу.

Клас «AuthMessageSenderOptions» – клас, що призначений для об’єднання даних що стосуються сервісу для надсилання пошти – «EmailSender», ці наді знаходяться у конфігурації. Опис полів та методів класу наведені у таблиці 4.13.

Таблиця 4.13 – Опис полів та методів класу «AuthMessageSenderOptions»

Поля та методи класу	Опис
private:	
string? SendGridKey	API ключ для доступу до сервісу SendGrid.
string? SendGridFromEmail	Електрона адреса з якої виконується надсилання листа.

Клас «EmailSender» – клас, що призначений для надсилання електронного листа користувачу. Опис полів та методів класу наведені у таблиці 4.14. У розділі 2 додатку Б можна побачити код цього класу.

Таблиця 4.14 – Опис полів та методів класу «EmailSender»

Поля та методи класу	Опис
Is-A:	
IEmailSender	Інтерфейс, що містить метод для перевизначення логіки надсилання листа.
private:	
ILogger _logger	Екземпляр класу логування.
public:	
EmailSender(IOptions<AuthMessageSenderOptions> optionsAccessor, ILogger<EmailSender> logger)	Конструктор.
AuthMessageSenderOptions Options	Екземпляр класу з даними конфігурації.
async Task SendEmailAsync(string toEmail, string subject, string message)	Метод для надсилання листа (викликається через сервіс «IEmailSender»).
async Task Execute(string apiKey, string subject, string message, string toEmail)	Допоміжний метод для надсилання листа (фактичне надсилання).

Клас «TdnDbContext» – клас, що призначений для управління базою даних. Опис полів та методів класу наведені у таблиці 4.15.

Таблиця 4.15 – Опис полів та методів класу «TdnDbContext»

Поля та методи класу	Опис
Is-A:	
IdentityDbContext<User>	Базовий клас для бази даних на основі користувача.

Продовження таблиці 4.15

Поля та методи класу	Опис
protected:	
override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)	Метод для конфігурації бази даних.
override void OnModelCreating(ModelBuilder builder)	Метод для конфігурації моделей бази даних.
public:	
DbSet<Project> Projects	Колекція проєктів.
DbSet<Models.Task> Tasks	Колекція задач.
DbSet<Note> Notes	Колекція нотаток.
DbSet<Models.Label> Labels	Колекція міток.

4.2.2 Моделі-подання

Моделі-подання є елементом за допомогою якого реалізовується архітектура MVC. Саме моделі-подання є тими даними які отримує подання від контролера. Ці моделі-подання можуть містити попередньо розглянуті моделі. Також роль моделі-подання можуть виконувати і сама модель.

Моделі-подання можна розділити на три групи:

– «AccountViewModels», моделі-подання для автентифікаційної логіки;

– «ManageViewModels», моделі-подання для логіки роботи користувача з його акаунтом;

– «MainViewModels», моделі-подання для основної логіки.

Прикладом моделі-подання з групи «AccountViewModels» є клас «RegisterViewModel» що містить тільки ті дані класу «User» що необхідні для реєстрації. Опис полів та методів класу наведені у таблиці 4.16.

Таблиця 4.16 – Опис полів та методів класу «ManageLoginsViewModel»

Поля та методи класу	Опис
public:	
string Name	Ім’я користувача.
string Email	Пошта користувача.
string Password	Пароль користувача.
string ConfirmPassword	Підтвердження паролю користувача.

Прикладом моделі-подання з групи «ManageViewModels» є клас «ChangeEmailViewModel» що містить необхідні для зміни пошти користувача дані. Опис полів та методів класу наведені у таблиці 4.17.

Таблиця 4.17 – Опис полів та методів класу «ChangeEmailViewModel»

Поля та методи класу	Опис
public:	
string OldEmail	Стара електрона адреса.
string NewEmail	Нова електрона адреса.

Прикладом моделі-подання з групи «MainViewModels» є клас «GeneralViewModel» що містить дані необхідні для головної сторінки. Опис полів та методів класу наведені у таблиці 4.18.

Таблиця 4.18 – Опис полів та методів класу «GeneralViewModel»

Поля та методи класу	Опис
public:	
List<Project> Projects	Список проєктів користувача.
List<TdnSortElement>	Елементи що підлягають сортуванню (задачі та нотатки).

Продовження таблиці 4.18

Поля та методи класу	Опис
public:	
List<Label> Labels	Мітки користувача.
ManageViewModels.IndexViewModel Manage	Інша модель-подання необхідна для відображення інформації про користувача.
ManageViewModels.ManageLoginsViewModel Logins	Інші модель-подання необхідна для відображення інформації стосовно зовнішніх провайдерів автентифікації.

За таким самим принципом будуються і інші моделі-подання. Для групи «AccountViewModels» це:

– ForgotPasswordViewModel;
– LoginViewModel;
– RegisterViewModel;
– ResetPasswordViewModel.

Для групи «ManageViewModels» це:

– ChangeEmailViewModel;
– ChangeNameViewModel;
– ChangePasswordViewModel;
– IndexViewModel;
– ManageLoginsViewModel;
– RemoveLoginViewModel;
– SetPasswordViewModel.

Для групи «MainViewModels» це:

– BinViewModel;
– GeneralViewModel;
– NoteLabelsViewModel;

– ProjectLabelViewModel;
– TaskLabelsViewModel.

4.2.3 Контролери

Контролери є ключовим елементом за допомогою якого реалізовується архітектура MVC. Саме є першим хто реагує на запити користувача та вирішує які наступні дії виконує вебзастосунок. Наприклад, контролер може повернути подання з властивими йому даними. Подання розглянути далі у розділі.

Прикладом контролера є клас «TasksController» що реагує на запити користувача пов’язані із задачами. Опис полів та методів класу наведені у таблиці 4.19. У розділі 3 додатку Б можна побачити відповідний код цього контролеру.

Таблиця 4.19 – Опис полів та методів класу «TasksController»

Поля та методи класу	Опис
private:	
TdnDbContext _context	Екземпляр класу для управління базою даних.
UserManager<User> _userManager	Екземпляр класу для управління користувачами.
IAuthorizationService _authorizationService	Екземпляр класу сервісу для проведення авторизації.
bool TaskExists(int? id)	Метод для перевіри на існування задачі.
async Task<bool> SetSelectedLabelsAsync(TaskLabelsViewModel taskLabelsViewModel)	Метод призначений для встановлення вибраних міток у відповідну модель-подання.

Продовження таблиці 4.19

Поля та методи класу	Опис
private:	
IActionResult RedirectToLocal(string returnUrl)	Метод призначений для переадресування до певного шляху у вебзастосунку.
public:	
[HttpGet] async Task<IActionResult> CreatePartialAsync(string? returnUrl = null)	Повертає відповідне подання створення задачі для введення даних.
[HttpPost] async Task<IActionResult> CreatePartial(TaskLabelsViewModel taskLabels, string? returnUrl = null)	Реалізовує створення задачі на основі отриманої моделі-подання.
[HttpGet] public async Task<IActionResult> EditPartial(int? id, string? returnUrl = null)	Повертає відповідне подання редагування задачі на основі отриманого ідентифікатора.
[HttpPost] async Task<IActionResult> EditPartial(TaskLabelsViewModel taskLabels, string? returnUrl = null)	Реалізовує редагування задачі на основі отриманої моделі-подання.
[HttpPost] async Task<IActionResult> SoftDelete(int? id, string? returnUrl = null)	Реалізовує редагування задачі (змінної що відповідає за статус видалення задачі).
[HttpPost] async Task<IActionResult> Restore(int? id, string? returnUrl = null)	Реалізовує редагування задачі (змінної що відповідає за статус видалення задачі) на основі отриманого ідентифікатора.
[HttpGet] async Task<IActionResult> DeletePartial(int? id, string? returnUrl = null)	Повертає відповідне подання видалення задачі на основі отриманого ідентифікатора.

Продовження таблиці 4.19

Поля та методи класу	Опис
public:	
[HttpPost] async Task<IActionResult> DeleteConfirmed(int? id, string? returnUrl = null)	Реалізовує повне видалення задачі на основі отриманого ідентифікатора.
[HttpPost] async Task<IActionResult> Duplicate(int? id, string? returnUrl = null)	Реалізовує створення копії задачі на основі отриманого ідентифікатора.
[HttpPost] async Task<IActionResult> ToggleState(int? id, string? returnUrl = null)	Реалізовує редагування задачі (змінної що відповідає за статус виконання задачі) на основі отриманого ідентифікатора.

За таким самим принципом будуються і інші контролери, призначення котрих було визначено у таблиці у пункті 3.3.7:

– AccountController;
– HomeController;
– LabelsController;
– ManageController;
– NotesController;
– ProjectsController;
– TasksController.

4.2.4 Подання

Подання – частина архітектури MVC що відображає інтерфейс користувача з яким він безпосередньо взаємодіє. Подання, як правило, має назву від методу відповідного контролера, тому немає необхідності їх всіх перелічувати. Подання за видом розділено на три основні групи:

– «Views», подання що представляють повну сторінку;

– «PartialViews», часткові подання що представляють повторювані елементи інтерфейсу;

– «Layouts», подання що є шаблонами для інших подань з груп «Views» та «PartialViews».

Група «Views» поділяється на наступні категорії:

– «Account», подання які повертає контролер «AccountController»;

– «Home», подання які повертає контролер «HomeController».

Група «PartialViews» поділяється на наступні категорії:

– «Home», подання які повертає контролер «HomeController»;

– «Labels», подання які повертає контролер «LabelsController»;

– «Manage», подання які повертає контролер «ManageController»;

– «Notes», подання які повертає контролер «NotesController»;

– «Projects», подання які повертає контролер «ProjectsController»;

– «Tasks», подання які повертає контролер «TasksController».

Окрім трьох основних груп також є наступні умовні групи:

– «Imports», список просторів імен для подань;

– «Starts», встановлення шаблонного подання з групи «Layouts» для сторінки.

4.2.5 Сервіси

У програмі використовуються наступні сервіси:

– SendGrid;

– Google OAuth.

Сервіс «SendGrid» – сервіс, що надає можливості надсилати електроні листи [28]. Використовується розглянутим у пункті 4.2.1 класом «EmailSender».

Сервіс «Google OAuth» – сервіс, що дозволяє імплементувати автентифікацію за допомогою Google-акаунту [29]. Інтегрується у ASP.NET

Core Identity класи у ролі зовнішнього провайдера автентифікації. Використовується при вході або реєстрації користувача у контролері «AccountController».

Необхідні ключі для доступу до цих сервісів у вебзастосунку зберігаються у файлі конфігурації.

4.3 Висновки за розділом 4

У четвертому розділі було:

– описано основні алгоритми програм за допомогою UML-діаграм послідовності;

– наведено детальний опис розроблених модулів розроблюваної системи;

Після чого стало можливим приступати до тестування.

5 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

Цей розділ є важливою частиною роботи, оскільки він розкриває етапи, пов’язані з впровадженням та використанням розробленого вебзастосунку. У цьому розділі буде здійснено опис роботи, наведено інструкції з використання та проведено тестування вебзастосунку.

5.1 Опис роботи з програмою

5.1.1 Звернення до програми

Звернення до програми відбувається через рядок пошуку у браузері (URL-адресу), рисунок 5.1. Після переходу за URL-адресою сайту ми можемо бачити стартову сторінку яка пропонує нам або увійти або зареєструватися у вебзастосунку. Якщо сайт буде опублікований він може мати доменне ім’я відмінне від локального localhost.

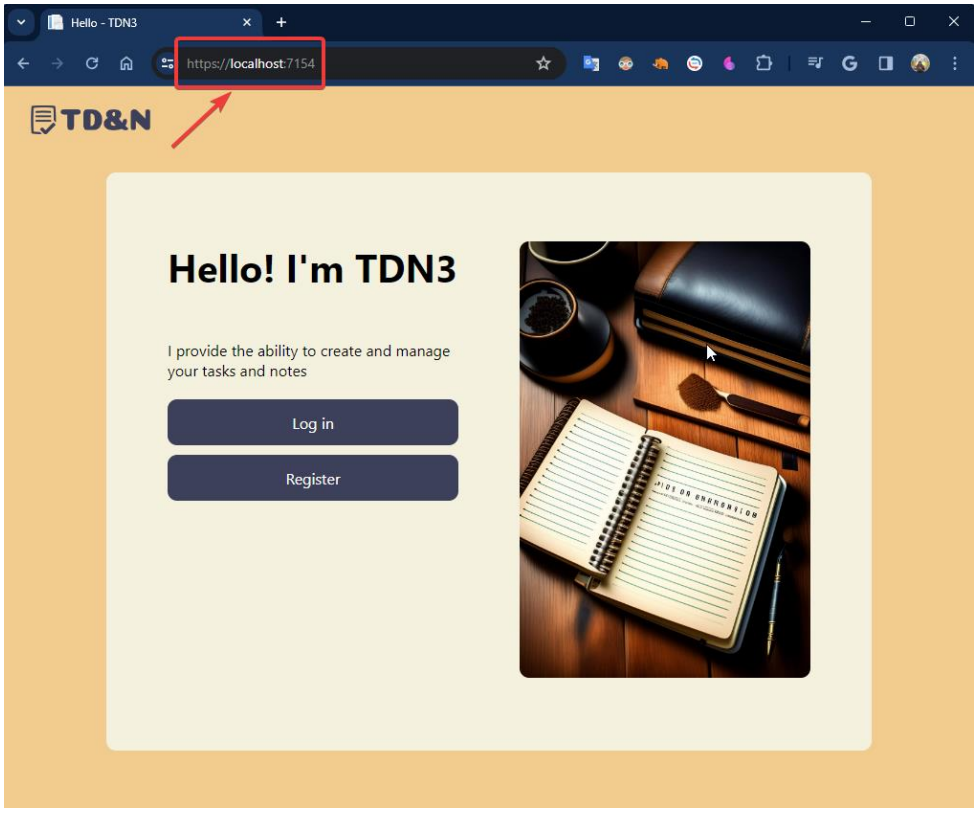

Рисунок 5.1 – Звернення до програми через рядок пошуку

5.1.2 Вхідні й вихідні дані

Вхідні та вихідні дані залежать від частини вебзастосунку з якою працює користувач. Якщо користувач працює з налаштуваннями акаунту то вхідними даними є:

– ім’я;
– електрона адреса;
– пароль користувача.

Якщо користувач працює з: проєктами, задачами, нотатками, мітками то вхідними даними є:

– назва поточного об’єкта;
– опис (у задач та нотаток) поточного об’єкта;
– статус (у задач) поточного об’єкта;
– термін виконання (у задач та нотаток) поточного об’єкта;
– пов’язаний проєкт (у задач та нотаток);
– пов’язані мітки (у задач та нотаток).

Введення вхідних даних відбувається через відповідні форми на сторінках, на рисунку 5.2 продемонстровано приклад введення даних необхідних для реєстрації користувача.

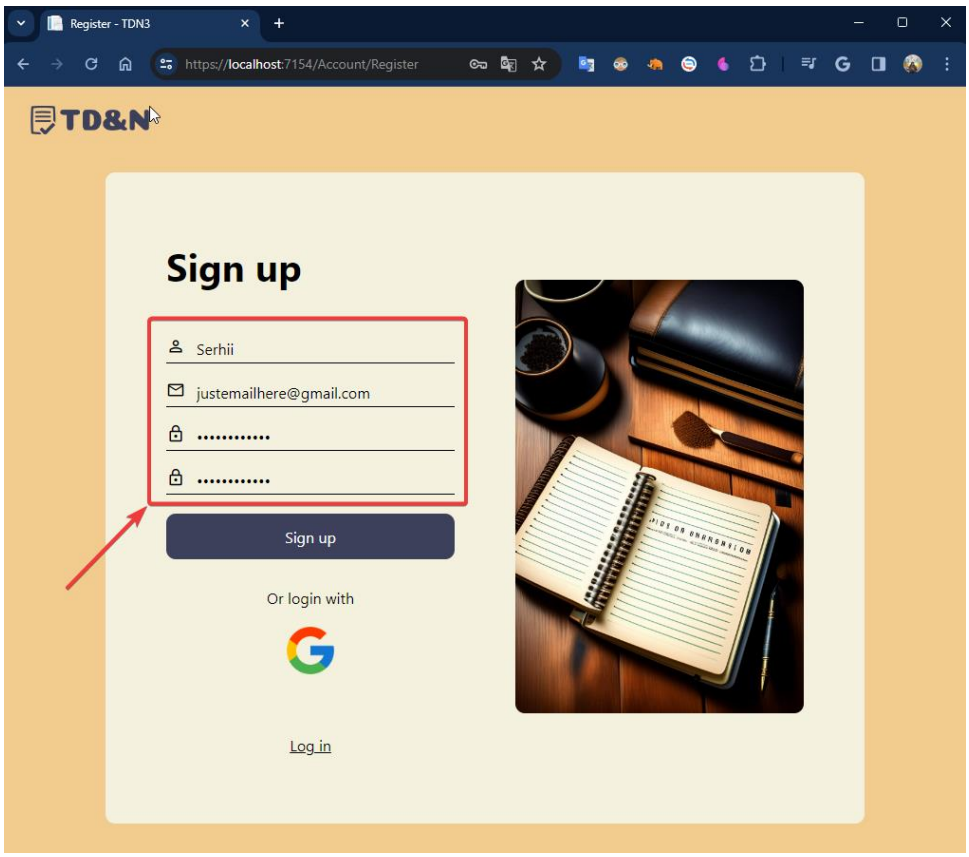

Рисунок 5.2 – Введені вхідні дані для реєстрації

Якщо користувач запитує доступ до якоїсь сторінки то вихідними даними є повернута сервером сторінка вебзастосунку. Якщо користувач працює з: проєктами, задачами, нотатками, мітками то вихідними даними є збереження, змінення або видалення цих об’єктів у базі даних та подальше відображення статусу цих змін на екрані шляхом повернння вебсторінки. На рисунку 5.3 продемонстрована повернута сервером сторінка з помилкою про неспівпадіння паролів.

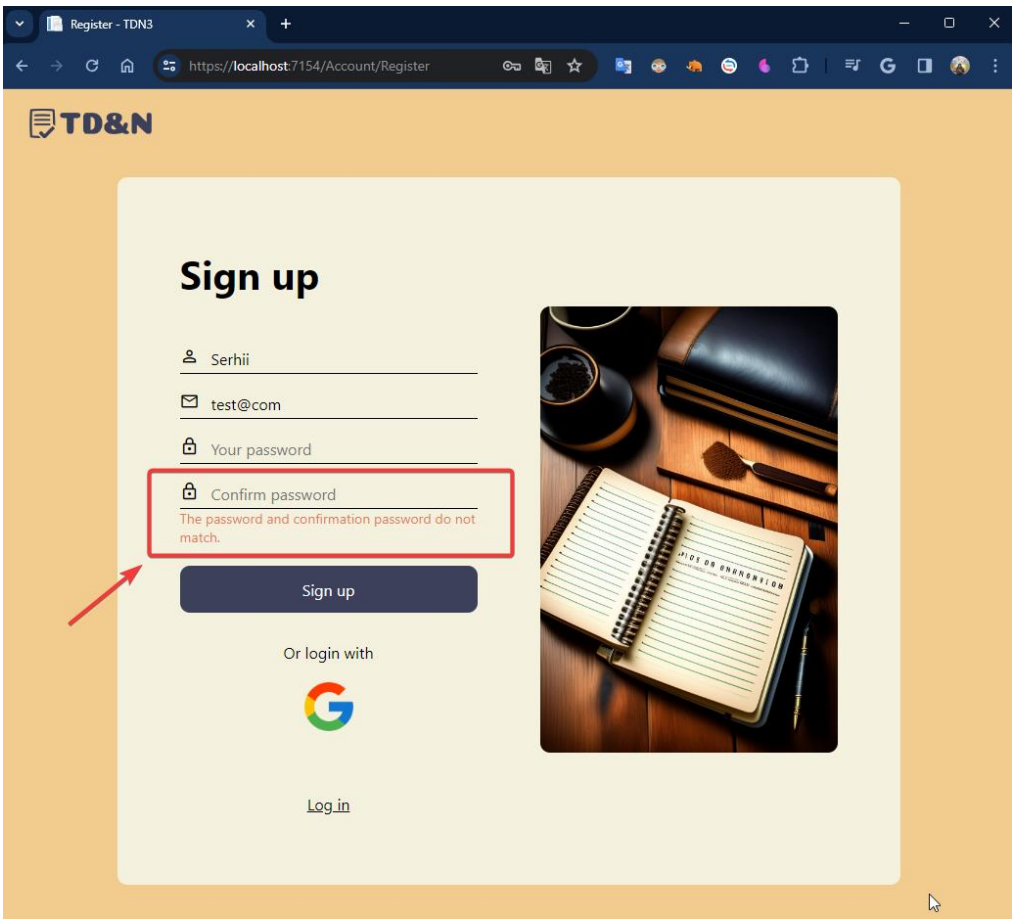

Рисунок 5.3 – Вихідні дані реєстрації

5.2 Інструкція з використання

Користувач, який використовує цей додаток, повинен мати базові навички роботи з технічним обладнанням, що використовується для його експлуатації, а також розуміти основи взаємодії з графічним інтерфейсом. Незважаючи на простоту інтерфейсу вебзастосунку, користувачеві рекомендується ознайомитися з документацією для більш ефективної роботи з вебзастосунком.

5.3 Тестування

Для тестування було вибрано ручний спосіб тестування (manual testing) коли тестувальники вручну виконують тести, не використовуючи ніяких засобів автоматизації. Ручне тестування – самий низькорівневий та

простий тип тестування, що не вимагає великої кількості додаткових знань [30].

Тестування буде проводитися на ПК, з характеристиками:

– процесор: AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz;

– оперативна пам’ять: 16,0 GB;

– операційна система: Windows 11.

Результати тестування частини вебзастосунку що стосується автентифікації користувача наведені у таблиці 5.1. З таблиці бачимо що всі 5 тестів пройдено успішно.

Таблиця 5.1 – Результати тестування системи автентифікації.

№	Назва	Дії	Очікуваний результат	Фактичний результат
1	Реєстрація з валідними даними.	1. Введення необхідних даних у форму: Serhii, sergiy.ssh@gmail.com, s^Aagad56sg, s^Aagad56sg. 2. Відправлення форми.	Переадресація на сторінку що запрошує перевірити електрону скриньку для підтвердження пошти.	Переадресація на сторінку що запрошує перевірити електрону скриньку для підтвердження пошти.

Продовження таблиці 5.1

№	Назва	Дії	Очікуваний результат	Фактичний результат
2	Реєстрація з невалідними даними у полі підтвердження паролю.	1. Введення необхідних даних у форму: Serhii, sergiy.ssh@gmail.com, s^Aagad56sg, s^Aagad56sg11. 2. Відправлення форми.	Сповіщення про помилку неспівпадіння полів паролю.	Сповіщення про помилку неспівпадіння полів паролю.
3	Вхід з валідними даними.	1. Введення необхідних даних у форму: sergiy.ssh@gmail.com, s^Aagad56sg. 2. Відправлення форми.	Успішний вхід та переадресація на головну сторінку.	Успішний вхід та переадресація на головну сторінку.
4	Вхід з невалідними даними у полі паролю.	1. Введення необхідних даних у форму: sergiy.ssh@gmail.com, s^Aagad56sg11. 2. Відправлення форми.	Сповіщення про помилку при вході.	Сповіщення про помилку при вході.

Продовження таблиці 5.1

№	Назва	Дії	Очікуваний результат	Фактичний результат
5	Вхід за допомогою Google.	1. Клік по відповідній іконці Google 2. Вибір Google-акаунту.	Успішний вхід та переадресація на головну сторінку.	Успішний вхід та переадресація на головну сторінку.

Результати тестування частини вебзастосунку що стосується дій пов’язаних із системою проєктів, міток, задач та нотаток наведені у таблиці 5.2. З таблиці бачимо що всі 14 тестів пройдено успішно.

Таблиця 5.2 – Результати тестування системи проєктів, міток, задач та нотаток.

№	Назва	Дії	Очікуваний результат	Фактичний результат
1	Створення з назвою.	1. Введення необхідних даних у форму: “Hello name”. 2. Відправлення форми.	Успішно створено.	Успішно створено.
2	Створення без назви.	Відправлення форми без заповненого поля назви.	Сповіщення про обов’язковість поля назви.	Сповіщення про обов’язковість поля назви.
3	Редагування.	1. Змінити дані у полях форми. 2. Відправлення форми.	Успішно відредаговано.	Успішно відредаговано.

Продовження таблиці 5.2

№	Назва	Дії	Очікуваний результат	Фактичний результат
4	Переміщення до кошику (все окрім міток).	Клік по відповідній кнопці «Move to bin».	Переміщено до кошику.	Переміщено до кошику.
5	Відновлення все окрім міток).	Клік по відповідній кнопці «Restore».	Відновлено.	Відновлено.
6	Видалення.	Клік по відповідній кнопці «Delete forever».	Видалено.	Видалено.
7	Пошуковий рядок.	1. Введення рядка: “Hello” у пошуковий рядок. 2. Натиснути кнопку «Enter».	Відображено результат пошуку серед задач та нотаток.	Відображено результат пошуку серед задач та нотаток.
8	Сортування за спаданням дати.	Клік по відповідній кнопці «Due date (desc)».	Задачі та нотатки відсортовано за спаданням дати.	Задачі та нотатки відсортовано за спаданням дати.
9	Сортування за зростанням дати.	Клік по відповідній кнопці «Due date (asc)».	Задачі та нотатки відсортовано за зростанням дати.	Задачі та нотатки відсортовано за зростанням дати.

Продовження таблиці 5.2

№	Назва	Дії	Очікуваний результат	Фактичний результат
10	Приховування виконаних задач.	Клік по відповідній кнопці «Completed» з іконкою ока.	Виконані задачі приховано.	Виконані задачі приховано.
11	Показати виконані задачі.	Клік по відповідній кнопці «Completed» з іконкою перекресленого ока.	Виконані задачі показано.	Виконані задачі показано.
12	Очищення кошику.	Клік по відповідній кнопці «Empty bin»	Вміст кошику видалено.	Вміст кошику видалено.
13	Виконання задачі.	Клік по кружечку створеної задачі.	Задача помічена як виконана.	Задача помічена як виконана.
14	Відмінити виконання задачі.	Клік по кружечку виконаної задачі.	Задача помічена як невиконана.	Задача помічена як невиконана.

Результати тестування частини вебзастосунку що стосується дій пов’язаних із управлінням акаунтом користувача наведені у таблиці 5.3. З таблиці бачимо що всі 6 тестів пройдено успішно.

Таблиця 5.3 – Результати тестування системи управління акаунтом користувача.

№	Назва	Дії	Очікуваний результат	Фактичний результат
1	Змінити ім’я.	1. Клік по відповідній кнопці «Change name». 2. Введення нового ім’я. 3. Відправлення форми.	Успішно змінено.	Успішно змінено.
2	Змінити пошту.	1. Клік по відповідній кнопці «Change email». 2. Введення нової пошти. 3. Відправлення форми.	Переадресація на сторінку що запрошує перевірити електрону скриньку для підтвердження пошти.	Переадресація на сторінку що запрошує перевірити електрону скриньку для підтвердження пошти.
3	Змінити пароль.	1. Клік по відповідній кнопці «Change password». 2. Введення даних форми. 3. Відправлення форми.	Пароль успішно змінено.	Пароль успішно змінено.

Продовження таблиці 5.3

№	Назва	Дії	Очікуваний результат	Фактичний результат
4	Підключити Google-акаунт.	1. Клік по відповідній кнопці «Connect Google». 2. Вибір Google-акаунту.	Google-акаунт під’єднано.	Google-акаунт під’єднано.
5	Відключити Google-акаунт.	Клік по відповідній кнопці «Disconnect Google».	Google-акаунт від’єднано.	Google-акаунт від’єднано.
6	Видалити акаунт.	1. Клік по відповідній кнопці «Delete account». 2. Відправлення форми.	Акаунт видалено.	Акаунт видалено.

5.4 Висновки за розділом 5

У п’ятому розділі було:

– здійснено опис роботи з програмою;

– наведено інструкції з використання;

– проведено тестування.

За результатами успішного проходження всіх тестів вебзастосунок можна вважати готовим до експлуатації.

ВИСНОВКИ

У ході виконання дипломної роботи, об’єктом дослідження якої є процес управління задачами, а предметом – вебзастосунок для управління задачами, було розроблено вебзастосунок, який надає користувачам зручні інструменти для керування задачами та нотатками.

Важливість дослідження полягає в його здатності покращити продуктивність та ефективність роботи користувачів у різних сферах їхньої діяльності у сучасний світі з властивим йому швидким темпом життя та постійним потоком інформації, що вимагає ефективних засобів для організації робочого та особистого часу.

У цій роботі було виконано поставлені перед нею завдання:

– проаналізовано існуючі підходи до керування задачами та нотатками;

– розроблено архітектуру та функціональні можливості вебзастосунку для керування задачами та нотатками;

– реалізовано розроблену систему та проведено її тестування.

Результатом цієї роботи став створений функціональний вебзастосунок, який забезпечить користувачам зручні інструменти для планування, організації та виконання задач, а також збереження нотаток. Такий застосунок може мати широкі практичні застосування у різних сферах діяльності, починаючи від особистого використання до використання в бізнесі та освіті.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке тайм-менеджмент? Основи управління часом [Електрон. ресурс]. – Режим доступу: https://edin.ua/shho-take-tajm-menedzhment-osnovi-upravlinnya-chasom/.

2. Tame your work, organize your life [Electronic resource]. – Access mode: https://evernote.com/.

3. Trello brings all your tasks, teammates, and tools together life [Electronic resource]. – Access mode: https://trello.com/.

4. Програма Microsoft To Do [Електрон. ресурс]. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/microsoft-to-do-list-app.

5. Google Keep [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Google_Keep.

6. Organize your work and life, finally [Electronic resource]. – Access mode: https://todoist.com/.

7. Що таке backend-розробка і чим вона відрізняється від frontend [Електрон. ресурс]. – Режим доступу: https://www.buh24.com.ua/shho-take-backend-rozrobka-i-chym-vona-vidriznyayetsya-vid-frontend/.

8. Рейтинг мов програмування 2023. JavaScript/TypeScript завойовують світ, Python увійшов у топ-3, Salesforce Apex випередив 1C [Електрон. ресурс]. – Режим доступу: https://dou.ua/lenta/articles/language-rating-2023/.

9. Чим відрізняються мови програмування? [Електрон. ресурс]. – Режим доступу: https://foxminded.ua/vidminnosti-mov-prohramuvannia/.

10. C Sharp (programming language) [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

11. What is jQuery? [Electronic resource]. – Access mode: https://jquery.com/.

12. Best C# IDEs in 2022 [Electronic resource]. – Access mode: https://www.codeguru.com/tools/7-best-c-sharp-ide/.

13. 5 найпопулярніших систем управління базами даних 2020 року [Електрон. ресурс]. – Режим доступу: https://www.management.com.ua/partners/2021/02/21/5-najpopulyarnishih-sistem-upravlinnya-bazami-danih-2020-roku/.

14. Шаблони проєктування: їх види, особливості й переваги [Електрон. ресурс]. – Режим доступу: https://refactoring.guru/uk/design-patterns/classification.

15. Choose an ASP.NET Core web UI [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-6.0.

16. ASP.NET Core Blazor [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0.

17. Вебсервер [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80.

18. Сервер даних [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.

19. Клієнт (інформатика) [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0).

20. Introduction to Identity on ASP.NET Core [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio.

21. Website wireframe [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Website_wireframe.

22. Google Fonts [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Google_Fonts.

23. Модальне вікно [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%B2%D1%96%D0%BA%D0%BD%D0%BE.

24. Launch your business with a free logo [Electronic resource]. – Access mode: https://logo.com/.

25. AuthorizeAttribute Class [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/previous-versions/aspnet/dd460317(v=vs.118).

26. Resource-based authorization in ASP.NET Core [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/resourcebased?view=aspnetcore-8.0.

27. Identity model customization in ASP.NET Core [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-8.0.

28. Get your emails to the inbox—where they belong [Electronic resource]. – Access mode: https://sendgrid.com/en-us.

29. Using OAuth 2.0 to Access Google APIs [Electronic resource]. – Access mode: https://developers.google.com/identity/protocols/oauth2.

30. Ручне та автоматизоване тестування [Електрон. ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/.

ДОДАТОК А

Технічне завдання

А.1 Вступ

А.1.1 Найменування

Найменування – «Вебзастосунок для управління задачами».

А.1.2 Коротка характеристика області застосування

Сучасний світ характеризується швидким темпом життя та постійним потоком інформації, що вимагає ефективних засобів для організації робочого та особистого часу. У зв'язку з цим виникає актуальна проблема керування задачами та нотатками.

Керування задачами та нотатками є одним з ключових елементів ефективного організаційного та особистого управління часом.

Керування задачами – це процес планування, організації та контролю за виконанням різних задач або обов'язків. Включає у себе призначення пріоритетів, визначення строків та відстеження прогресу.

Керування нотатками – це короткі записи або відзначення, які служать для збереження інформації або нагадування про певні пункти.

А.2 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Вебзастосунок для управління задачами», затверджене наказом Національного університету «Запорізька політехніка» № 168 від 24 квітня 2024 р.

А.3 Призначення розробки

А.3.1 Функціональне призначення

Функціональним призначенням вебзастосунку є задовільнення наступних функціональних вимог:

а) автентифікація користувача;

1) реєстрація користувача у системі;

2) вхід користувача у систему;

– вхід через електрону адресу;

– вхід за допомогою Google акаунту;

3) вихід користувача із системи;

б) авторизація;

в) управління акаунтом:

1) зміна імені користувача;

2) зміна електронної адреси користувача;

3) встановлення паролю;

4) змінення паролю;

5) під’єднання Google акаунту;

6) від’єднання Google акаунту;

7) видалення акаунту;

г) управління проєктами:

1) проєкт за замовчуванням;

2) створення проєкту;

3) перейменування проєкту;

4) створення копії проєкту;

5) переміщення проєкту до кошику;

6) відновити проєкт з кошику;

7) видалення проєкту назавжди;

ґ) управління мітками:

1) створення мітки;

2) перейменування мітки;

3) видалення мітки назавжди;

д) управління задачами:

1) створення задачі;

2) редагування задачі;

3) створення копії задачі;

4) переміщення задачі до кошику;

5) відновити задачу з кошику;

6) видалення задачі назавжди;

е) управління нотатками:

1) створення нотатки;

2) редагування нотатки;

3) створення копії нотатки;

4) переміщення нотатки до кошику;

5) відновити нотатку з кошику;

6) видалення нотатки назавжди;

є) кошик:

1) очистити весь кошик;

2) видалення окремого проєкту/задачі/нотатки;

2) відновлення окремого проєкту/задачі/нотатки;

ж) фільтрування:

1) за поданням;

2) за проєктом;

3) за міткою;

4) за пошуковим рядком;

з) сортування:

1) за спаданням дати;

2) за зростанням дати;

3) за статусом задач;

– показувати виконані задачі;

– приховати виконані задачі.

А.3.2 Експлуатаційне призначення

Вебзастосунок призначений для управління задачами та нотатками.

А.4 Вимоги до програми чи програмному виробу

А.4.1 Вимоги до функціональних характеристик

А.4.1.1 Групи користувачів

Вебзастосунок призначений для використання одним типом користувачів – «User». Цей тип користувача повинен мати можливість реалізації всіх перелічених у пункті 3.1 функціональних вимог у своєму інтерфейсі.

А.4.1.2 Вимоги до складу виконуваних функцій

Список функціональних вимог було наведено у пункті А.3.1. Деякі з них потребують додаткового опису:

– проєкт за замовчуванням;

– очистити весь кошик.

Функціональна вимога «Проєкт за замовчуванням». Повинна бути реалізована у вигляді проєкту який створюється при першому успішному вході користувача у вебзастосунок. Цей проєкт не може бути змінений, перейменований або видалений користувачем. За замовчуванням, задачі та нотатки що створюються знаходяться у цьому проєкті.

Функціональна вимога «Очистити весь кошик». Повинна бути реалізовано у вигляді видалення всіх даних з кошику назавжди

А.4.1.3 Вимоги до організації вхідних даних

Вхідні та вихідні дані залежать від частини вебзастосунку з якою працює користувач. Якщо користувач працює з налаштуваннями акаунту то вхідними даними є:

– ім’я;

– електрона адреса;

– пароль користувача.

Якщо користувач працює з: проєктами, задачами, нотатками, мітками то вхідними даними є:

– назва поточного об’єкта;
– опис (у задач та нотаток) поточного об’єкта;
– статус (у задач) поточного об’єкта;
– термін виконання (у задач та нотаток) поточного об’єкта;
– пов’язаний проєкт (у задач та нотаток);
– пов’язані мітки (у задач та нотаток).

А.4.1.4 Вимоги до організації вихідних даних

Якщо користувач запитує доступ до якоїсь сторінки то вихідними даними є повернута сервером сторінка вебзастосунку.

Якщо користувач працює з: проєктами, задачами, нотатками, мітками то вихідними даними є збереження, змінення або видалення цих об’єктів у базі даних та подальше відображення статусу цих змін на екрані шляхом повернння вебсторінки.

А.4.2 Вимоги до надійності

А.4.2.1 Вимоги до забезпечення надійного функціонування програми

Надійність функціонування вебзастосунку забезпечується виконанням наступних основних вимог:

– безперервне постачання живлення на пристрій який взаємодіє з вебзастосунком;
– стабільне інтернет-з’єднання;
– виконання вимог до складу та параметрів технічних засобів (пункт 4.4).

А.4.2.2 Збійні ситуації

Можуть виникати наступні збійні ситуації:

– користувач вже зареєстрований;
– неправильний логін або пароль;
– створення проєктів, міток, задач або нотаток без назв;
– запит на ресурсу без відповідного доступу;
– інші збійні ситуації на стороні сервера.

Інформація стосовно збійних ситуацій повинна бути представлена користувачу у вигляді повідомлень на поточній сторінці або перенаправленні на спеціальну сторінко для помилок.

А.4.3 Умови експлуатації

А.4.3.1 Кліматичні умови експлуатації

Кліматичні умови експлуатації вебзастосунку повинні відповідати умовам експлуатації технічних засобів наведених у пункті А.4.4.

А.4.3.2 Вимоги до персоналу

Користувач повинен мати базові навички роботи з технічним обладнанням, що використовується для його експлуатації, а також розуміти основи взаємодії з графічним інтерфейсом. Незважаючи на простоту інтерфейсу вебзастосунку, користувачеві рекомендується ознайомитися з документацією для більш ефективної роботи з вебзастосунком.

А.4.4 Вимоги до складу та параметрів технічних засобів

Вимоги до технічних засобів:

– Процесор Intel Core i3-8100F з тактовою частотою, ГГц – 3,6 або еквівалентний;

– Оперативна пам'ять: DDR3 з об’ємом мінімум 4 Гб;

– Засоби виводу інформації: монітор з роздільною здатністю не менше 1280x800 пікселів;

– Дисковий простір: мінімум 1 ГБ вільного місця на жорсткому диску.

Вимоги до програмного забезпечення:

– Веббраузер: останні версії Google Chrome, Mozilla Firefox, Microsoft Edge, Safari або аналогічних;

– Інтернет-з'єднання: Стабільне з'єднання з Інтернетом для завантаження та використання вебзастосунку;

– Підтримка JavaScript: увімкнена підтримка веббраузером для виконання скриптів JavaScript.

– Операційна система: Windows 10 або вище, macOS 10.13 або вище, Linux (Ubuntu, Fedora тощо);

А.4.5 Вимоги до інформаційної та програмної сумісності

А.4.5.1 Вимоги до інформаційних структур та методів рішення

Інформаційні структури мають бути розроблені згідно структури вхідних та вихідних даних.

Архітектурний підхід – MVC (Model-View-Controller).

Фреймворк – ASP.NET Core MVC.

А.4.5.2 Вимоги до вихідних кодів, мов програмування та програмних засобів

Вихідні коди для серверної частини вебзастосунку повинні бути реалізованими мовою програмування C#, для клієнтської – JavaScript з бібліотекою jQuery. Також базуючись на мовах програмування використовуватиметься середовище розробки – Visual Studio. З огляду на

екосистему (компанії Microsoft), використовуватиметься MS SQL Server у ролі СУБД.

А.4.6 Вимоги до маркування й упакування

Програма поставляється у вигляді посилання на репозиторій GitHub на якому знаходяться.

А.4.6.1 Вимоги до маркування

Вимоги до маркування відсутні.

А.4.6.2 Вимоги до упакування

Вимоги до упакування відсутні. Вебзастосунок поставляється за допомогою посилання на репозиторії GitHub.

А.4.7 Вимоги до транспортування та збереження

Вимоги до транспортування та збереження відсутні.

А 4.8 Спеціальні вимоги

Перелік нефункціональних вимог:

– назва вебзастосунку – «To Do & Notes 3»

– мова вебзастосунку – англійська;

– задачі та нотатки повинні розміщуватися окремо у двох стовпчиках.

А.5 Вимоги до програмної документації

Склад програмної документації повинен включати технічне завдання розроблене відповідно до предметної області, призначення вебзастосунку.

А.6 Техніко-економічні показники

Ефективність програмного забезпечення має оцінюватися в зекономленому при плануванні користувачем своїх задач та нотаток.

Орієнтована економічна ефективність не враховується, програма не потребує жодних економічних вливань та є цілком безкоштовною.

А.7 Стадії та етапи розробки

Розробка ведеться у 5 етапів:

– розробка технічного завдання (1 тиждень);
– розробка архітектури програмного забезпечення (2 тиждень);
– розробка програмного забезпечення (3-4 тиждень);
– тестування програмного забезпечення (5 тиждень).

А.8 Порядок контролю та приймання

Розробник повинен провести випробовування всього функціоналу вебзастосунку. Упевнившись у повній працездатності вебзастосунку, він передається замовнику.

ДОДАТОК Б

Текст програми

Б.1 Клас IsOwnerAuthorizationHandler

```
        public class IsOwnerAuthorizationHandler :
AuthorizationHandler<OperationAuthorizationRequirement, object>
        {
            private readonly UserManager<Models.User> _userManager;
            public IsOwnerAuthorizationHandler(UserManager<Models.User> userManager)
            {
                _userManager = userManager;
            }
            protected override System.Threading.Tasks.Task
HandleRequirementAsync(AuthorizationHandlerContext context,
OperationAuthorizationRequirement requirement, object resource)
            {
                if (context.User == null || resource == null) { return
Task.CompletedTask; }
                if (requirement.Name != Constants.FullAccessOperationName) { return
Task.CompletedTask; }

                if (resource == null) { return Task.CompletedTask; }

                string? ownerId = null;
                Func<Models.Project, string?> getProjectOwnerIdFunc = project =>
project?.UserId;
                Func<Models.Label, string?> getLabelOwnerIdFunc = label =>
label?.UserId;
                Func<Models.Task, string?> getTaskOwnerIdFunc = task =>
task?.Project?.UserId;
                Func<Models.Note, string?> getNoteOwnerIdFunc = note =>
note?.Project?.UserId;

                switch (resource)
                {
                    case Models.Project p:
                        ownerId = getProjectOwnerIdFunc(p);
                        break;
                    case Models.Label l:
                        ownerId = getLabelOwnerIdFunc(l);
                        break;
                    case Models.Task t:
                        ownerId = getTaskOwnerIdFunc(t);
                        break;
                    case Models.Note n:
```

```
                ownerId = getNoteOwnerIdFunc(n);
                break;
            }

            if (ownerId == _userManager.GetUserId(context.User))
            {
                context.Succeed(requirement);
                Console.WriteLine("Success authorization: " + DateTime.Now);
            }
            return Task.CompletedTask;
        }
    }
```

Б.2 Клас EmailSender

```
public class EmailSender : IEmailSender
{
    private readonly ILogger _logger;

    public EmailSender(IOptions<AuthMessageSenderOptions> optionsAccessor,
                       ILogger<EmailSender> logger)
    {
        Options = optionsAccessor.Value;
        _logger = logger;
    }

    public AuthMessageSenderOptions Options { get; } //Set with Secret
Manager.

    public async Task SendEmailAsync(string toEmail, string subject, string
message)
    {
        if (string.IsNullOrEmpty(Options.SendGridKey))
        {
            throw new Exception("Null SendGridKey");
        }
        await Execute(Options.SendGridKey, subject, message, toEmail);
    }
    public async Task Execute(string apiKey, string subject, string message,
string toEmail)
    {
        var client = new SendGridClient(apiKey);
        var msg = new SendGridMessage()
        {
            From = new EmailAddress(Options.SendGridFromEmail, "Password
Recovery"),
            Subject = subject,
            PlainTextContent = message,
            HtmlContent = message
        };
        msg.AddTo(new EmailAddress(toEmail));

        // Disable click tracking.
        // See https://sendgrid.com/docs/User_Guide/Settings/tracking.html
        msg.SetClickTracking(false, false);
        var response = await client.SendEmailAsync(msg);
        _logger.LogInformation(response.IsSuccessStatusCode
                               ? $"Email to {toEmail} queued successfully!"
                               : $"Failure Email to {toEmail}");
    }
}
```

Б.3 Клас AccountController

```
public class AccountController : Controller
{
    private readonly UserManager<User> _userManager;
    private readonly SignInManager<User> _signInManager;
    private readonly IEmailSender _emailSender;
    private readonly ILogger _logger;

    public AccountController(
        UserManager<User> userManager,
        SignInManager<User> signInManager,
        IEmailSender emailSender,
        ILoggerFactory loggerFactory)
    {
        _userManager = userManager;
        _signInManager = signInManager;
        _emailSender = emailSender;
        _logger = loggerFactory.CreateLogger<AccountController>();
    }

    // GET: Account/Register
    [HttpGet]
    [AllowAnonymous]
    public IActionResult Register(string returnUrl = null)
    {
        ViewData["ReturnUrl"] = returnUrl;
        if (User?.Identity?.IsAuthenticated == true)
        {
            return RedirectToLocal(returnUrl);
        }
        return View();
    }

    // POST: Account/Register
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Register(RegisterViewModel model,
string returnUrl = null)
    {
        ViewData["ReturnUrl"] = returnUrl;
        if (ModelState.IsValid)
```

```
            {
                var user = await _userManager.FindByEmailAsync(model.Email);
                IdentityResult result = IdentityResult.Failed();

                if (user is null)
                {
                    user = new User { UserName = model.Email, Email =
model.Email, CustomName = model.Name };
                    result = await _userManager.CreateAsync(user,
model.Password);
                }
                else if(user is not null && user?.EmailConfirmed == false)
                {
                    user.UserName = model.Email;
                    user.Email = model.Email;
                    user.CustomName = model.Name;
                    result = await _userManager.UpdateAsync(user);
                }

                if (result.Succeeded)
                {
                    // Send/resend an email
                    var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                    var callbackUrl = Url.Action("ConfirmEmail", "Account", new
{ userId = user.Id, code = code }, protocol: HttpContext.Request.Scheme);
                    await _emailSender.SendEmailAsync(model.Email, "Confirm your
account",
                        "Please confirm your account by clicking this link: <a
href=\"" + callbackUrl + "\">link</a>");
                    _logger.LogInformation(3, "User created a new account with
password.");
                    return View("CheckEmail", user.Email);
                }
                AddErrors(result);
            }
            return View(model);
        }

        // GET: Account/CheckEmail
        [HttpGet]
        [AllowAnonymous]
        public IActionResult CheckEmail(string email)
        {
```

```
            return View("CheckEmail", email);
        }

        // GET: Account/ConfirmEmail
        [HttpGet]
        [AllowAnonymous]
        public async Task<IActionResult> ConfirmEmail(string userId, string
code)
        {
            if (userId == null || code == null)
            {
                return View("Error");
            }
            var user = await _userManager.FindByIdAsync(userId);
            if (user == null)
            {
                return NotFound($"Unable to load user with ID '{userId}'.");
            }
            var result = await _userManager.ConfirmEmailAsync(user, code);
            return View(result.Succeeded ? "ConfirmEmail" : "Error");
        }

        // GET: Account/ChangeEmailPartial
        [HttpGet]
        public async Task<IActionResult> ChangeEmailConfirm(string userId,
string email, string code)
        {
            if (userId == null || email == null || code == null)
            {
                return View("Error");
            }

            var user = await _userManager.FindByIdAsync(userId);
            if (user == null)
            {
                return NotFound($"Unable to load user with ID '{userId}'.");
            }

            // Check if the new email already exists
            var existingUser = await _userManager.FindByEmailAsync(email);
            if (existingUser != null)
            {
                return View("Error");
            }
```

```
                code = Encoding.UTF8.GetString(WebEncoders.Base64UrlDecode(code));
                var result = await _userManager.ChangeEmailAsync(user, email, code);
                if (!result.Succeeded)
                {
                    // Handle failure
                    return NotFound();
                }

                // In our UI email and user name are one and the same, so when we
update the email
                // we need to update the user name.
                var setUserNameResult = await _userManager.SetUserNameAsync(user,
email);
                if (!setUserNameResult.Succeeded)
                {
                    // Handle failure
                    return NotFound();
                }

                await _signInManager.RefreshSignInAsync(user);
                return View(nameof(ConfirmEmail));
            }

            // GET: Account/Login
            [HttpGet]
            [AllowAnonymous]
            public IActionResult Login(string returnUrl = null)
            {
                ViewData["ReturnUrl"] = returnUrl;

                if (User?.Identity?.IsAuthenticated == true)
                {
                    return RedirectToLocal(returnUrl);
                }
                return View();
            }

            // POST: Account/Login
            [HttpPost]
            [AllowAnonymous]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> Login(LoginViewModel model, string
returnUrl = null)
```

```
            {
                ViewData["ReturnUrl"] = returnUrl;

                if (ModelState.IsValid)
                {
                    var result = await
_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe,
lockoutOnFailure: false);
                    if (result.Succeeded)
                    {
                        _logger.LogInformation(1, "User logged in.");
                        return RedirectToLocal(returnUrl);
                    }
                    else
                    {
                        ModelState.AddModelError(string.Empty, "Invalid login
attempt.");
                        return View(model);
                    }
                }
                return View(model);
            }

            // POST: Account/LogOff
            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> LogOff()
            {
                await _signInManager.SignOutAsync();
                TempData.Clear();
                _logger.LogInformation(4, "User logged out.");
                return RedirectToAction(nameof(HomeController.Index), "Home");
            }

            // POST: Account/ExternalLogin
            [HttpPost]
            [AllowAnonymous]
            [ValidateAntiForgeryToken]
            public IActionResult ExternalLogin(string provider, string returnUrl =
null)
            {
                var redirectUrl = Url.Action("ExternalLoginCallback", "Account", new
{ ReturnUrl = returnUrl });
```

```
                var properties =
_signInManager.ConfigureExternalAuthenticationProperties(provider, redirectUrl);
                return Challenge(properties, provider);
            }

            // GET: Account/ExternalLoginCallback
            [HttpGet]
            [AllowAnonymous]
            public async Task<IActionResult> ExternalLoginCallback(string returnUrl
= null, string remoteError = null)
            {
                if (remoteError != null)
                {
                    ModelState.AddModelError(string.Empty, $"Error from external
provider: {remoteError}");
                    return View(nameof(Login));
                }
                var info = await _signInManager.GetExternalLoginInfoAsync();
                if (info == null)
                {
                    return RedirectToAction(nameof(Login));
                }

                var result = await
_signInManager.ExternalLoginSignInAsync(info.LoginProvider, info.ProviderKey,
isPersistent: false);
                if (result.Succeeded)
                {
                    await
_signInManager.UpdateExternalAuthenticationTokensAsync(info);

                    _logger.LogInformation(5, "User logged in with {Name}
provider.", info.LoginProvider);
                    return RedirectToLocal(returnUrl);
                }
                if (result.IsLockedOut)
                {
                    return View("Lockout");
                }
                else
                {
                    // Create user
                    var claims = info.Principal.Claims.ToList();
                    var email = info.Principal.FindFirstValue(ClaimTypes.Email);
```

```
                    var name = info.Principal.FindFirstValue(ClaimTypes.Name);
                    User user = new User { Email = email, UserName = email,
CustomName = name, EmailConfirmed = true };

                    var createResult = await _userManager.CreateAsync(user);
                    if (createResult.Succeeded)
                    {
                        createResult = await _userManager.AddLoginAsync(user, info);
                        if (createResult.Succeeded)
                        {
                            await _signInManager.SignInAsync(user, isPersistent:
false);
                            _logger.LogInformation(6, "User created an account using
{Name} provider.", info.LoginProvider);

                            // Update any authentication tokens as well
                            await
_signInManager.UpdateExternalAuthenticationTokensAsync(info);

                            return RedirectToLocal(returnUrl);
                        }
                    }
                    AddErrors(createResult);
                    return View(nameof(Login));
                }
            }

            // GET: Account/ForgotPassword
            [HttpGet]
            [AllowAnonymous]
            public IActionResult ForgotPassword()
            {
                return View();
            }

            // POST: Account/ForgotPassword
            [HttpPost]
            [AllowAnonymous]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> ForgotPassword(ForgotPasswordViewModel
model)
            {
                if (ModelState.IsValid)
                {
```

```
                    var user = await _userManager.FindByEmailAsync(model.Email);
                    if (user == null)
                    {
                        // Don't reveal that the user does not exist or is not
confirmed
                        return View("CheckEmail");
                    }

                    if (!await _userManager.IsEmailConfirmedAsync(user))
                    {
                        var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                        var callbackUrl = Url.Action("ConfirmEmail", "Account", new
{ userId = user.Id, code = code }, protocol: HttpContext.Request.Scheme);
                        await _emailSender.SendEmailAsync(model.Email, "Reset
Password",
                            "To change your password, you must first confirm your
email by clicking this <a href=\"" + callbackUrl + "\">link</a>. Then try to change
the password again.");
                    }
                    else
                    {
                        var code = await
_userManager.GeneratePasswordResetTokenAsync(user);
                        var callbackUrl = Url.Action("ResetPassword", "Account", new
{ email = user.Email, code = code }, protocol: HttpContext.Request.Scheme);
                        await _emailSender.SendEmailAsync(model.Email, "Reset
Password",
                            "Please reset your password by clicking here: <a
href=\"" + callbackUrl + "\">link</a>");
                    }

                    return View("CheckEmail");
                }

                return View(model);
            }

            // GET: Account/ForgotPasswordConfirmation
            [HttpGet]
            [AllowAnonymous]
            public IActionResult ForgotPasswordConfirmation()
            {
                return View();
```

```
            }

            // GET: Account/ResetPassword
            [HttpGet]
            [AllowAnonymous]
            public IActionResult ResetPassword(string code = null, string email =
null)
            {
                return code == null || email == null ? View("Error") : View();
            }

            // POST: Account/ResetPassword
            [HttpPost]
            [AllowAnonymous]
            [ValidateAntiForgeryToken]
            public async Task<IActionResult> ResetPassword(ResetPasswordViewModel
model)
            {
                if (!ModelState.IsValid)
                {
                    return View(model);
                }

                var user = await _userManager.FindByEmailAsync(model.Email);
                if (user == null)
                {
                    // Don't reveal that the user does not exist
                    return
RedirectToAction(nameof(AccountController.ResetPasswordConfirmation), "Account");
                }
                var result = await _userManager.ResetPasswordAsync(user, model.Code,
model.Password);
                if (result.Succeeded)
                {
                    return
RedirectToAction(nameof(AccountController.ResetPasswordConfirmation), "Account");
                }
                AddErrors(result);
                return View();
            }

            // GET: Account/ResetPasswordConfirmation
            [HttpGet]
            [AllowAnonymous]
```

```
            public IActionResult ResetPasswordConfirmation()
            {
                return View();
            }

            // GET: Account/Error
            [HttpGet]
            [AllowAnonymous]
            public IActionResult Error()
            {
                return View();
            }

            #region Helpers

            private void AddErrors(IdentityResult result)
            {
                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }

            private IActionResult RedirectToLocal(string returnUrl)
            {
                if (Url.IsLocalUrl(returnUrl))
                {
                    return Redirect(returnUrl);
                }
                else
                {
                    return RedirectToAction(nameof(HomeController.Main), "Home", new
{ daysViewName = DaysViewName.Today });
                }
            }

            #endregion
        }
```

ДОДАТОК В

Слайди презентації

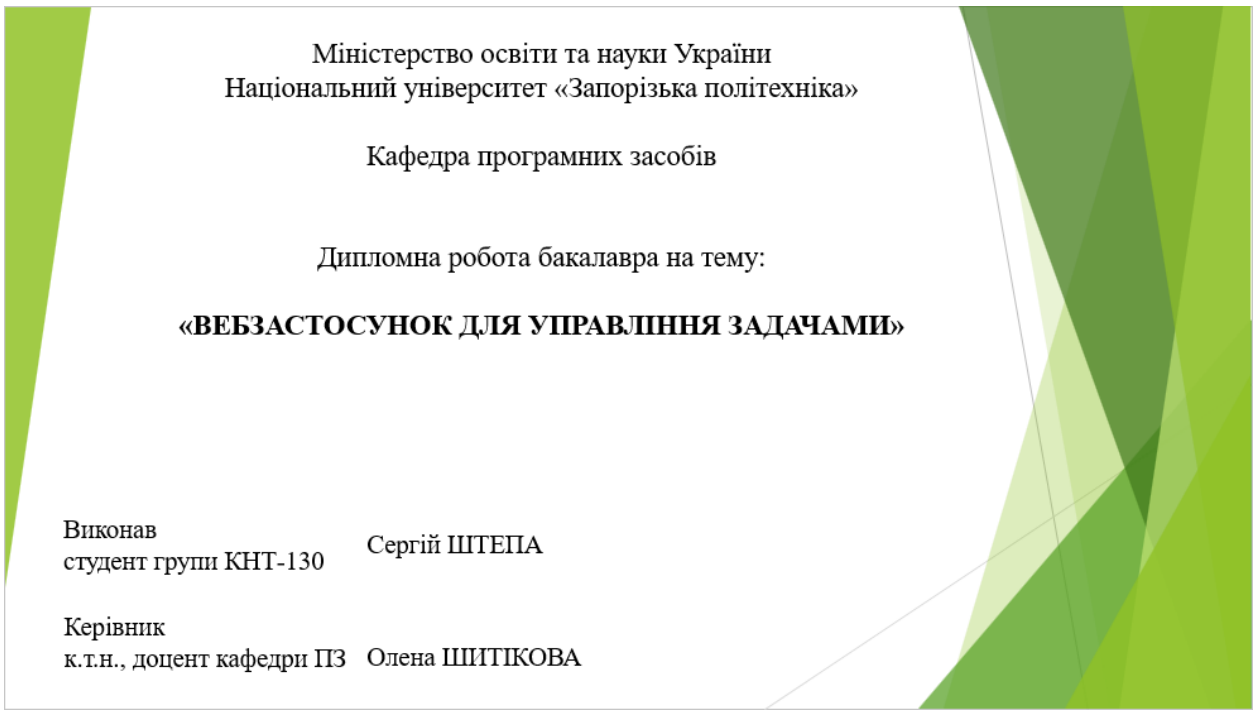

Рисунок В.1 – Слайд №1

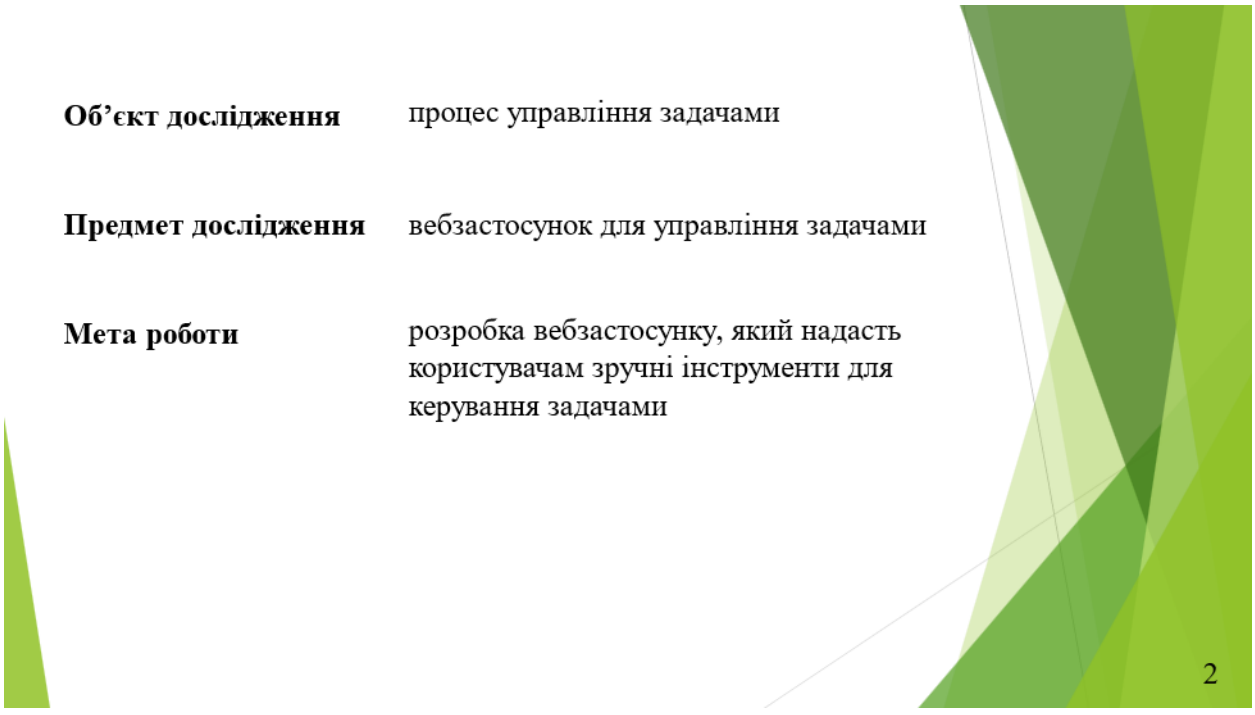

Рисунок В.2 – Слайд №2

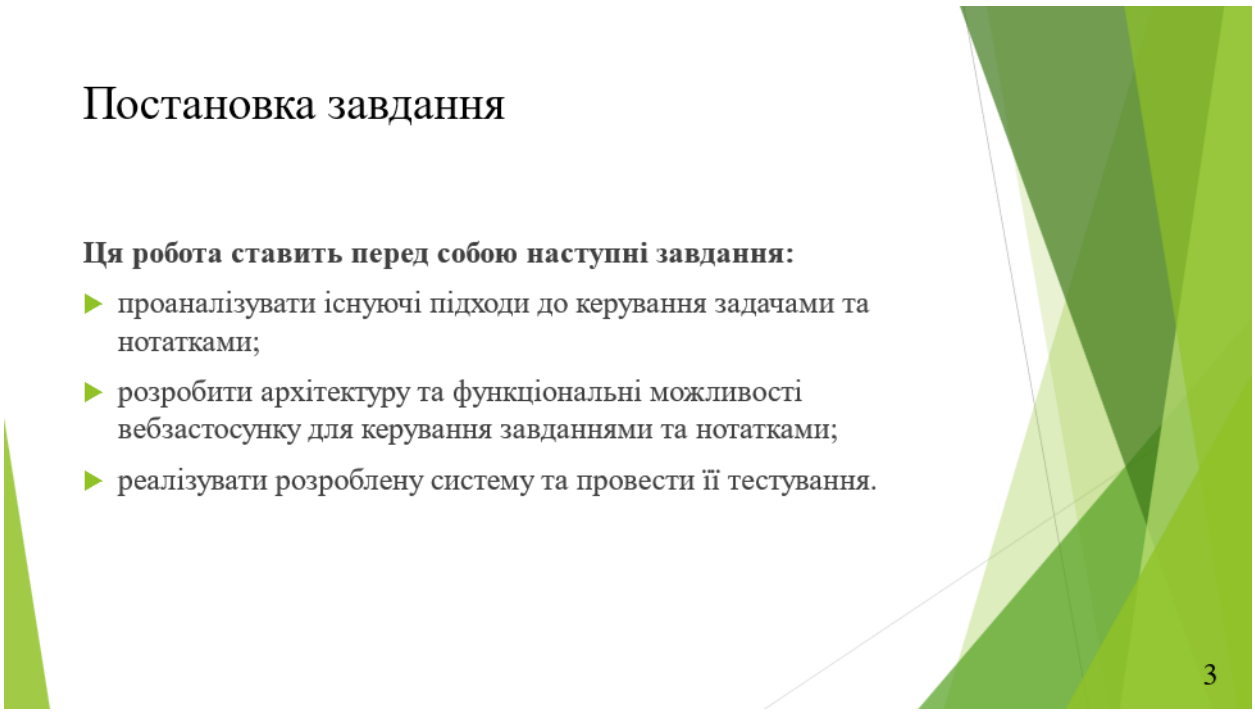

Рисунок В.3 – Слайд №3

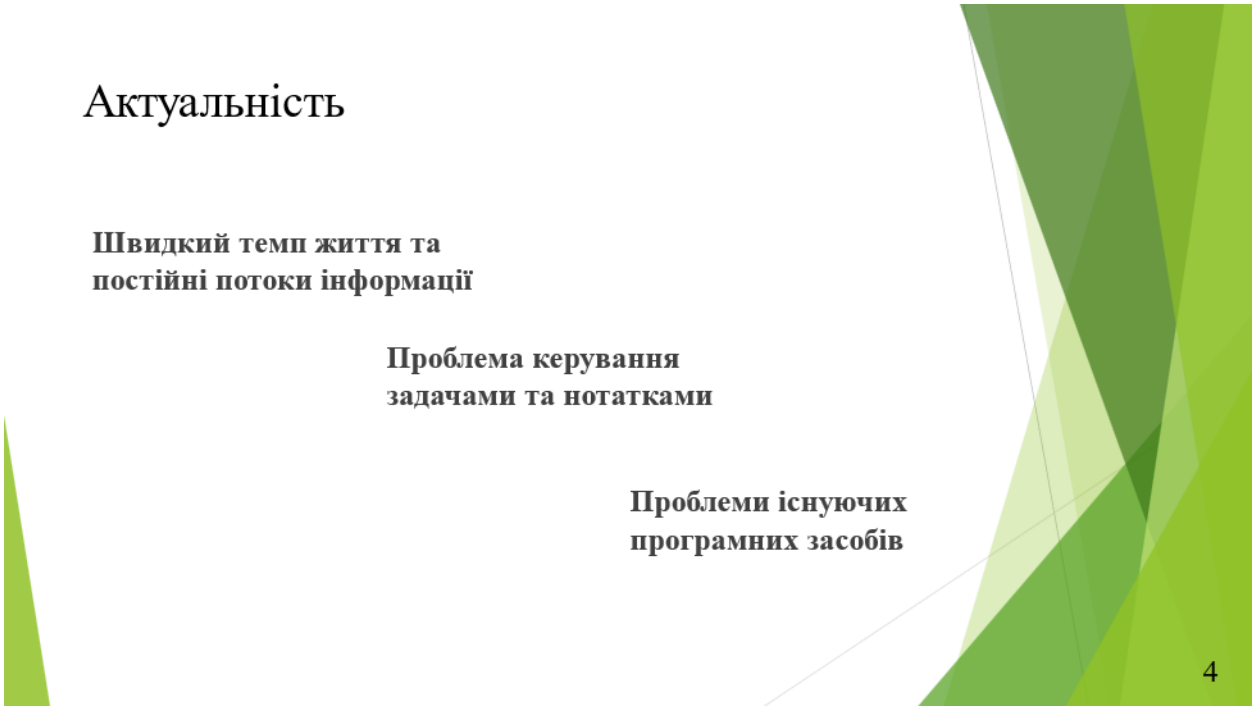

Рисунок В.4 – Слайд №4

Порівняння існуючих аналогів

	Evernote	Trello	Microsoft To Do	Google Keep	Todoist	Розроблений вебзастосунок
Задачі	+	+	+	-	+	+
Нотатки	+	+	-	+	-	+
Система проєктів	+	+	+	-	+	+
Система міток	+	+	-	+	+	+
Простий інтерфейс	-	-	+	+	+	+

5

Рисунок В.5 – Слайд №5

Функціональні вимоги

Основні функціональні вимоги:

- Автентифікація та авторизація
- Управління акаунтом
- Управління проєктами, мітками, задачами та нотатками
- Кошик
- Фільтрування
- Сортування

6

Рисунок В.6 – Слайд №6

Порівняння мов програмування

Мова програмування	Переваги	Недоліки
C#	Продуктивність, синтаксис, екосистема, бібліотеки, фреймворки.	Складність платформи .NET Core для новачків.
Java	Широке використання, продуктивність, багатоплатформність.	Складність синтаксису.
JavaScript	Універсальність, простота вивчення, велика спільнота.	Продуктивність, може бути складною для масштабування.
Python	Читабельність коду, велика кількість бібліотек.	Продуктивність, може бути складною для розробки складних вебзастосунків.

7

Рисунок В.7 – Слайд №7

Порівняння фреймворків

Фреймворк	Переваги	Недоліки
ASP.NET Core Blazor	Серверна та клієнтська логіка.	Можливості на клієнті.
ASP.NET Core MVC	Архітектура MVC. Максимальна гнучкість серверної логіки.	Тільки серверна логіка.
ASP.NET Core Razor Pages	Розмежування відповідальностей між сторінками «Razor Page».	Тільки серверна логіка.

8

Рисунок В.8 – Слайд №8

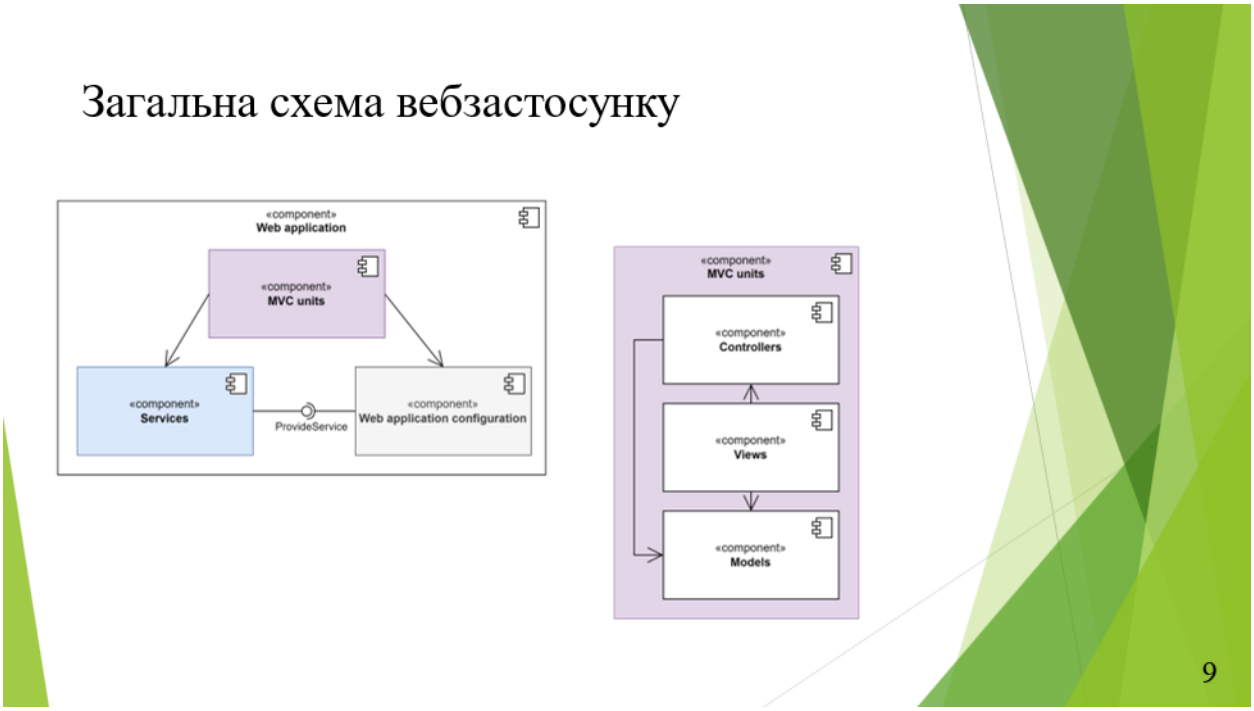

Рисунок В.9 – Слайд №9

Рисунок В.10 – Слайд №10

Інтерфейс вебзастосунку

11

Рисунок В.11 – Слайд №11

Висновки

У цій роботі було виконано поставлені перед нею завдання:

- проаналізовано існуючі підходи до керування задачами та нотатками;
- розроблено архітектуру та функціональні можливості вебзастосунку для керування завданнями та нотатками;
- реалізовано розроблену систему та проведено її тестування.

12

Рисунок В.12 – Слайд №12

Рисунок В.13 – Слайд №13