ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему**: «РОЗРОБКА ВЕБ-СЕРВІСУ ВЕДЕННЯ НОТАТОК З ОБЛАДНАННЯ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#»**

Виконав: студент 4 курсу, групи ПД–43

спеціальності

121 _Інженерія програмного забезпечення__

(шифр і назва спеціальності)

_____Цатурян Е.Р. _____

(прізвище та ініціали)

Керівник __Гаманюк І.М.____ _____

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерія програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Напрям підготовки -121 – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерія програмного

забезпечення

О.В. Негоденко

«_____»_____2022 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Цатуряну Едуарду Руслановичу

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка веб-сервісу ведення нотаток з обладнання з використанням мови програмування C#»
   Керівник роботи ст. викладач Гаманюк І.М. ,
   (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
   затверджені наказом вищого навчального закладу від «16» лютого 2022 року №22
2. Строк подання студентом роботи «3» червня 2022 року.
3. Вихідні дані до роботи: програмний застосунок розроблений у середовищі Rider на платформі .Net Core з використанням мови C#.
4. Зміст розрахунково-пояснювальної записки(перелік питань, які необхідно розробити):
   4.1. Аналіз та дослідження існуючих аналогів
   4.2. Дослідження програмних засобів
   4.3. Розробка структури веб-сервісу нотаток на мові програмування C#
   4.4. Програмна реалізація сервісу
   4.5. Висновки
5. Перелік графічного матеріалу
   5.1. Титульний слайд.

5.2. Аналоги.
5.3. Мета, об'єкт та предмет дослідження.
5.4. Технічне завдання.
5.5. Програмні засоби реалізації.
5.6. Діаграма прецедентів.
5.7. Архітектура застосунку.
5.8. Схема бази даних.
5.9. Відношення між шарами.
5.10. Репозиторії сервісу.
5.11. Екранні форми
5.12. Апробація результатів дослідження.
5.13. Висновки.

Дата видачі завдання: «11» квітня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04 – 12.04	Виконано
2	Аналіз та дослідження існуючих аналогів	13.04 – 14.04	Виконано
3	Дослідження програмних засобів	15.04 – 23.04	Виконано
4	Розробка структури веб-сервісу	25.04 01.05	Виконано
5	Програмна реалізація	02.05 – 25.05	Виконано
6	Висновки, оформлення роботи	26.05	Виконано
7	Розробка демонстраційних матеріалів	27.05 – 01.06	Виконано
8	Попередній захист роботи	03.06	Виконано
9	Здача роботи		

Студент _____ Цатурян Е.Р.
(підпис) (прізвище та ініціали)
Керівник роботи _____ Гаманюк І.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить: 76 ст., 10 табл., 50 рис., 1 дод., 36 посилань.

Мета роботи: покращення ведення нотаток за допомогою онлайн-засобів.

Об'єкт розробки: ведення нотаток за допомогою онлайн засобів.

Предмет розробки: веб-сервіс для ведення нотаток та керування ними.

Методи розробки: аналіз програмного забезпечення для створення і проєктування веб-сервісів; методи реєстрації, аутентифікації та валідації; засоби проєктування користувацького інтерфейсу.

У наш час людей оточує велика кількість даних та фактів. Для того, щоб не забувати певну інформацію або донести її іншим людям, необхідний універсальний засіб для структурування та ефективної передачі даних. Дана проблема може виникнути у бізнесі, навчанні та будь-якій іншій сфері сучасного життя.

Задачі, які повинні бути вирішені за допомогою розроблюваного продукту:

1. Зручність створення і представлення інформації.
2. Наочність інформації.
3. Зручність керування даними.
4. Зручність розповсюдження даних.

Результатом проведеної роботи є розроблений веб-сервіс для ведення та управління нотатками у вигляді серверної та клієнтської частини системи.

Ключові слова: ВЕБ-СЕРВІС, АУТЕНТИФІКАЦІЯ, C#, BACK-END, ENTITY FRAMEWORK, API, JWT, MSSQL SERVER, ANGULAR, SPA.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОСИЛАНЬ, СКОРОЧЕНЬ

Фреймворк – це програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проєкту.

Валідація – перевірка на правильність.

CRUD - Create, Read, Update, Delete – 4 базові функції управління даними «створення, зчитування, зміна і видалення».

HTTP - протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпертекстових документів.

IDE - Integrated Development Environment – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду.

SQL - structured query language – структурована мова запитів, являється основною для роботи з реляційними СКБД.

API — програмний інтерфейс додатку.

.NET – програмна технологія, як платформа для створення як звичайних програм, так і веб-застосунків.

JWT – JSON Web Token - це запропонований Інтернет-стандарт для створення даних із додатковим підписом та/або необов’язковим шифруванням, корисне навантаження якого містить JSON, який стверджує певну кількість претензій.

NuGet - система управління пакетами для платформ розробки Microsoft.

DTO – data transfer object – об’єкт передачі даних між шарами.

GitHub – платформа для збереження коду на віддалених серверах.

UI - User interface – інтерфейс програми видимий користувачу.

LINQ - Language-Integrated Query – компонент C# для запитів даних.

Скорочення:

БД – база даних.

ОС – операційна система.

ПЗ – програмне забезпечення.

ПС – програмна система.

СКБД – система керування базами даних.

ПК – персональний комп’ютер.

ЕЦП – електронний цифровий підпис.

СКВ – система керування версіями.

ВСТУП

У наш час людей оточує велика кількість даних та фактів. Для того, щоб не забувати певну інформацію або донести її іншим людям, необхідний універсальний засіб для структурування та ефективної передачі даних. Дана проблема може виникнути у бізнесі, навчанні та будь-якій іншій сфері сучасного життя.

Так як велика кількість людей працює та навчається онлайн, було б зручніше робити нотатки за допомогою браузера та ділитися потрібними нотатками з друзями або колегами. Існуюче програмне забезпечення як правило потрібно завантажувати. Також більшість додатків не надають можливості поділитися нотатками. Таким чином, актуальною задачею являється створення програмного продукту, що вирішує вищезазначені проблеми.

Онлайн ведення нотаток - веб-сервіс, що є програмним продуктом, що працює прямо через інтернет, що дозволяє вести нотатки і працювати з будь-якої точки світу. Під веб-сервісами розуміють послуги, що надаються в Інтернеті за допомогою спеціальних програм.

Наприклад, поширені такі сервіси, як пошукова система, хостинг, електронна пошта, зберігання в Інтернеті різної інформації (файли, закладки), календар і т.д. Важлива властивість веб-сервісу полягає в тому, що він не залежить від вашого провайдера, комп'ютера чи браузера – ви можете працювати зі своїми даними у будь-якій точці світу, де у вас є доступ до інтернету.

Мета роботи: покращення ведення нотаток за допомогою онлайн-засобів.

Об’єкт розробки: ведення нотаток за допомогою онлайн засобів.

Предмет розробки: веб-сервіс для ведення нотаток та керування ними.

Основна ідея полягає в тому, щоб допомогти користувачеві легко і швидко створити власний акаунт, де він зможе додавати нові нотатки та керувати вже створеними, відкривати доступ до нотаток іншим користувачам або певним користувачам(доступ за посиланням).

Завдання дослідження:

1. Аналіз та дослідження існуючих аналогів
2. Дослідження програмних засобів
3. Розробка структури веб-сервісу нотаток на мові програмування C#
4. Програмна реалізація сервісу

Методи дослідження:

– аналіз програмного забезпечення для створення і проєктування веб-сервісів;
– методи реєстрації, аутентифікації та валідації;
– засоби управління базами даних;
– засоби проєктування користувацького інтерфейсу.

Практична значущість результатів полягає у впровадженні веб-сервісу у повсякденне життя великої кількості людей та спрощення структурування та запису інформації користувачами. Зручність додатку полягає також у можливості ділитися нотатками з друзями, колегами або з усіма користувачами сервісу.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ АНАЛОГІВ

1.1 Програма для цифрових нотаток «OneNote»

Програма для цифрових нотаток «OneNote» - це веб-сервіс, який дозволяє створювати необмежену кількість нотаток, які можна розбити на сторінки та розділи. Можна вставляти відео або зображення, документи, голосові повідомлення. Нотатки можна помічати тегами для швидкого доступу до них. Для нотаток можна встановити пароль та відправити нотатку друзям[1].

Перед початком роботи у сервісі необхідно пройти процедуру реєстрації. Для реєстрації необхідний обліковий запис Microsoft. Якщо акаунт вже існує, то у сервісі достатньо авторизуватися за його допомогою. Після входу відображається головна сторінка сервісу(рис. 1.1).

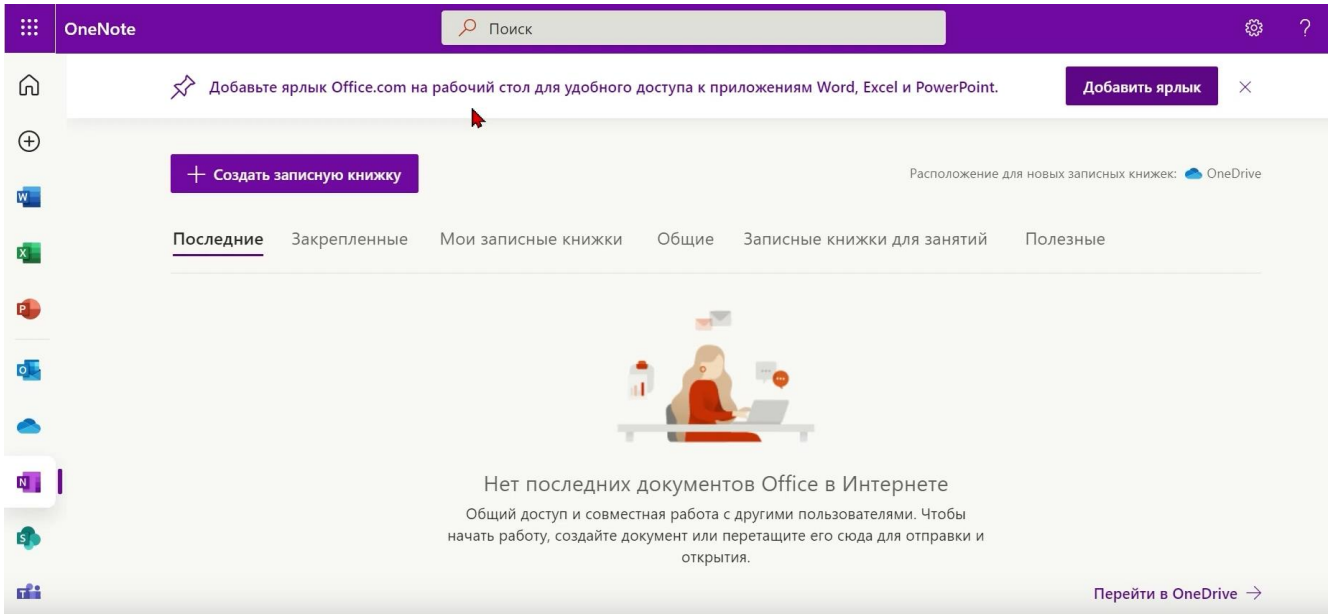


Рисунок 1.1 – Головна сторінка сервісу «OneNote»

З наведеного вище рисунку видно, що на головній сторінці доступна кнопка «Створити записну книжку» та вкладки, що відображають відповідні нотатки.

При створенні нової записної книжки фізичний файл зберігається у сервісі OneDrive, що входить до сімейства Microsoft. Створений файл можна редагувати

та налаштовувати до нього доступ(рис 1.2.). До файлу можна додати будь-який документ створений за допомогою програм Microsoft Office(презентацію PowerPoint, таблицю Excel, текст Word тощо).

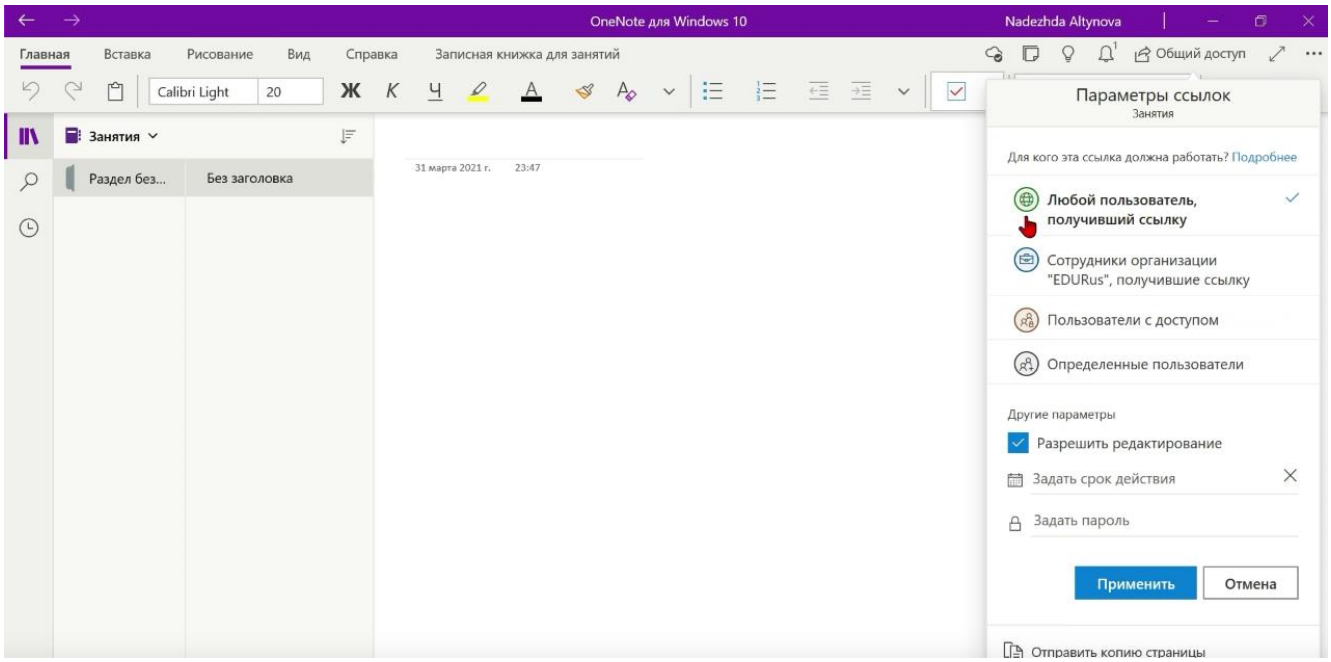


Рисунок 1.2 – Редагування нотатки та доступу до неї у сервісі «OneNote»

До недоліків даного веб-сервісу для цифрових нотаток можна віднести те, що:

- Синхронізація не найсильніша сторона Microsoft, тому сервіс не завжди правильно завантажує необхідні дані.
- Організація записів відрізняється від інших, до неї треба звикнути
- При видаленні нотаток вони залишаються збереженими на сервісі OneDrive, що може займати багато місця та призвести до непорозумінь у майбутньому, оскільки сервіс не повідомляє про такий функціонал користувача
- Інтерфейс не є інтуїтивно зрозумілим, багато функціоналу невідомого початковому користувачу, у якому треба розбиратися.
- Створену нотатку не можна зберегти у вигляді документу.

1.2 Онлайн сервіс для ведення нотаток «Evernote»

Онлайн сервіс для ведення нотаток «Evernote» - сервіс для збору нотаток, і одночасно робочий простір для записів, блокнотів, задач, проєктів, нагадувань. Evernote дозволяє створювати будь-які нотатки, списки справ, цілі дослідження, таблиці, видавати доступ до всієї інформації будь-яким іншим пристроям і користувачам. Сюди можна копіювати статті, збирати знімки, зберігати рукописні нотатки, створювати фотографії документів, сканувати їх, надсилати колегам або друзям. [2].

Перед початком користування необхідно пройти процедуру реєстрації, що можна побачити на сторінці сервісу при першому вході(рис. 1.3). У сервісі присутня інтеграція з Google, тому реєстрація може зайняти менше часу, якщо у користувача є акаунт Google.

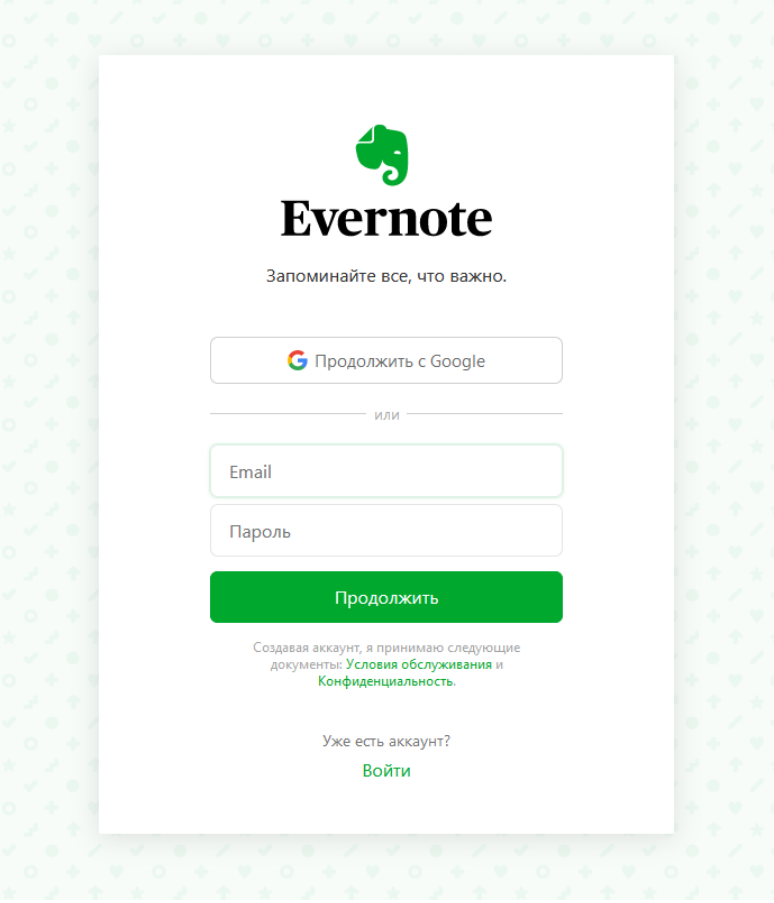


Рисунок 1.3 – Сторінка реєстрації у сервісі «Evernote»

Після входу до акаунту користувачу доступні наступні функції:

– Створення нотаток.
– Організація нотаток(можливість прикріпити їх до блокнотів або додати теги).
– Синхронізація між пристроями(у безкоштовній версії доступна синхронізація лише між двома пристроями)
– Пошук по тексту
– Можливість додати документ до нотатки
– Можливість встановлювати нагадування на певний час

Головну сторінку веб-сервісу можна побачити на рис. 1.4.

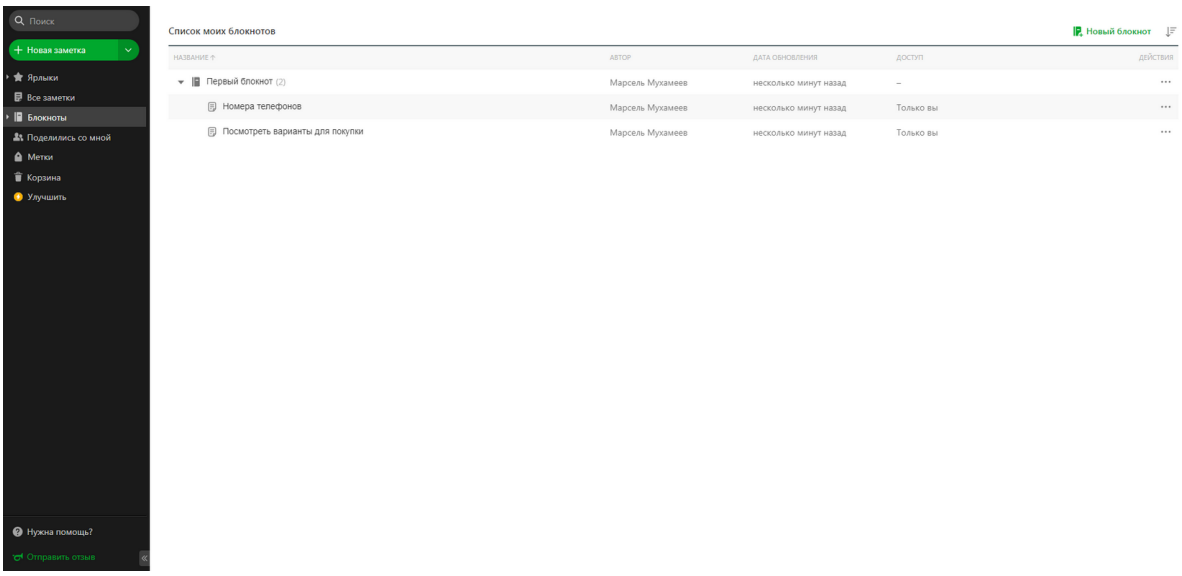


Рисунок 1.4 – Головна сторінка веб-сервісу «Evernote»

Безкоштовна версія сервісу містить наступні можливості[3]:

– Синхронізація лише між двома пристроями
– Максимальний розмір нотатки – 25Мб(у платній версії 200Мб)
– На місяць дається 60Мб на додавання нових нотаток(у платній версії 20Гб)
– Пошук по своїх нотатках
– Прикріплення документів та зображень

- Копіювання веб-сторінок

До недоліків даного веб-сервісу для ведення нотаток можна віднести те, що:

- Для використання повного функціоналу сервісу необхідно оформлювати підписку.
- Відсутність синхронізації із Google календарем
- Відсутність багатофакторної авторизації
- Не зручний інтерфейс для користувача, який тільки почав працювати із сервісом, необхідно звикати та читати про функціонал

- Відсутність можливості встановити пароль на нотатку

1.3 Програма для ведення нотаток «Google Keep»

Програма для ведення нотаток «Google Keep» - інструмент для створення і зберігання нотаток. Вона надає можливість зберігати і ділитися з друзями, колегами та співробітниками[4].

У додатку існує три типи нотаток:

- Нотатка-список: чек-лист, після створення якого користувач може відмітити внесені до списку вирішені справи та задачі.
- Нотатка з малюнком: користувачу доступна можливість малювати свої ідеї у редакторі.
- Фотонотатка: можна додавати фотографію або зображення та написати до них опис.

Як виглядає вкладка з нотатками можна побачити на рисунку 1.5.

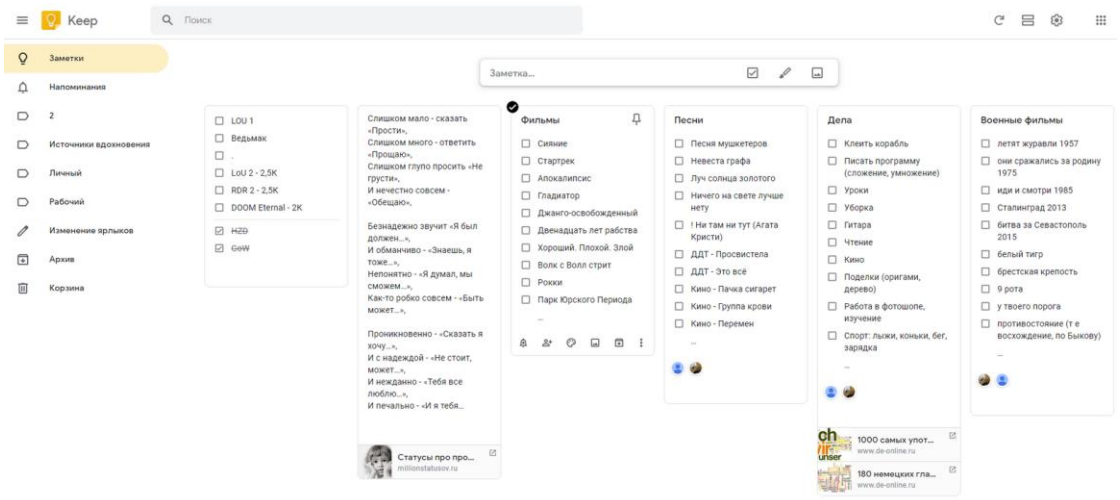


Рисунок 1.5 – Вкладка з нотатками додатку «Google Keep»

У додатку «Google Keep» користувачу доступні наступні функції:

– Безмежне сховище для нотаток.
– Пошук та система тегів.
– Встановлення нагадувань.
– Фотонотатки та нотатки з малюнками.
– Виділення нотаток кольорами.
– Синхронізація між пристроями.

Також у додатку можна переглядати архів нотаток(рис. 1.6).

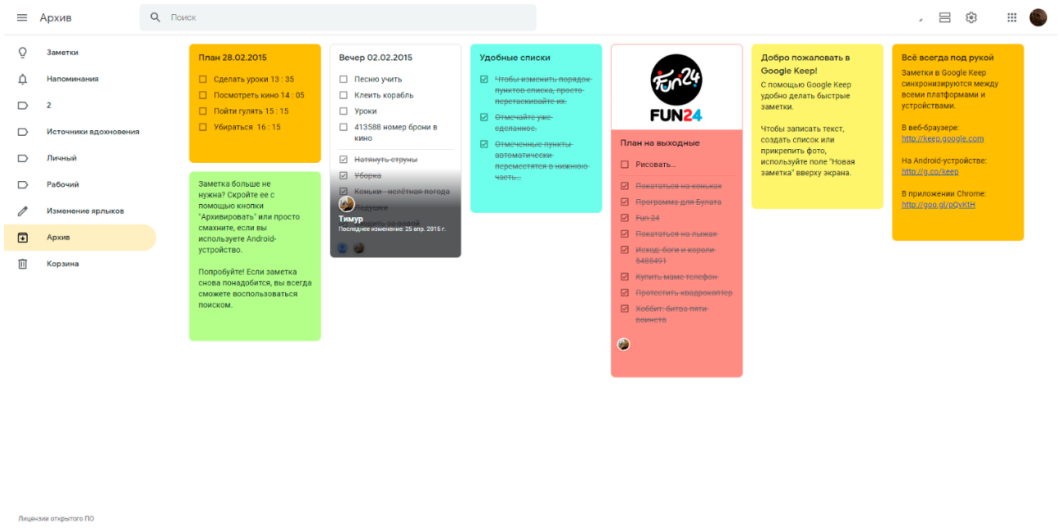


Рисунок 1.6 – Вкладка з архівом нотаток додатку «Google Keep»

До недоліків даної програми для ведення нотаток можна віднести те, що:

– Відсутня синхронізація із календарем.
– Відсутня багатофакторна авторизація.
– Відсутня можливість розділяти доступ до нотаток.
– Відсутня можливість відкривати доступ до нотатки за посиланням.
– Відсутність можливості робити підзадачі у нотатках
– Відсутність можливості зберігати нотатку у вигляді документу.

1.4 Огляд програмних продуктів-аналогів

Аналіз програм аналогів показав, що більшість проблем у додатках для ведення нотаток пов’язані із перенавантаженим та не зрозумілим інтерфейсом. Більшість компаній-розробників, створюючи продукт для ведення нотаток, додають до нього забагато функціоналу. У тому числі серед аналогів дуже мала кількість додатків, що доступні без завантаження. Також не у всіх сервісів доступна можливість відкрити доступ до нотатки за посиланням або ділитися нотаткою із іншими користувачами.

Узагальнене порівняння переваг та недоліків додатків представлена у таблиці 1.1.

Таблиця 1.1 – Порівняння аналогів.

Назва додатку	Основні переваги	Основні недоліки
OneNote	Можливість розбити нотатку на розділи. Можливість працювати із зображеннями та документами. Можливість налаштовувати доступ до нотатки.	Незрозумілий та перевантажений інтерфейс для звичайного користувача. Відсутність можливості зберігати нотатку у вигляді документу на пристрій користувача.

Продовження таблиці 1.1 – Порівняння аналогів.

Назва додатку	Основні переваги	Основні недоліки
Evernote	Встановлення нагадування. Можливість працювати із зображеннями та документами. Можливість встановити нагадування.	Безкоштовна версія занадто обмежена. Інтерфейс перенавантажений. Відсутність можливості встановити пароль на нотатку.
Google Keep	Пошук нотаток по тегам. Встановлення нагадувань. Виділення нотаток кольорами.	Відсутня можливість надавати доступ до нотатки іншим користувачам. Відсутність можливості зберігати нотатку у вигляді документу.

1.5 Аналіз вимог до майбутнього програмного забезпечення.

При розробці веб-сервісу було вирішено зробити акцент на простоті використання, конфіденційності інформації користувачів та зручному розмежуванні доступу. Сервіс повинен буди кросплатформним, тобто бути доступним на будь-якому пристрої з будь-якого браузеру, повинен бути невеликого розміру та містити в собі мінімальний функціонал, який необхідний для зручного створення та управління нотатками.

Функціональні вимоги:

- Створення та управління нотатками.
- Можливість встановлювати нотаткам рівень доступу(приватні, публічні та доступ за посиланням).
- Форматування тексту нотаток.
- Відображення нотаток користувача у власному акаунті

- Глобальний пошук по публічним

Нефункціональні вимоги:

- Доступність: система доступна і повністю працездатна у будь який день та час при наявності інтернет з’єднання.
- Продуктивність: Система знаходить дані достатньо швидко, не використовує місце на диску користувача та використовує найшвидші алгоритми пошуку даних.
- Надійність: У системі присутня авторизація користувачів, обмеження доступу користувачам з певними ролями. Система надає користувачу гарантію без-пеки та конфіденційності його даних. Вся інформація користувача використовується у системі лише за призначенням.
- Легкість в експлуатації: Система оснащена інтуїтивно зрозумілим інтерфейсом.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір засобів для розробки

2.1.1 Платформа .Net Core

.NET — це безкоштовна платформа розробки з відкритим вихідним кодом для створення різних типів програм:

- Веб-додатки
- Безсерверні функції
- Хмарні додатки
- Мобільні додатки
- Класичні додатки(Windows WPF, Windows Forms, UWP)
- Ігри
- Машинне навчання
- Консольні програми

По суті .NET Core - це практично повне перезавантаження стека .NET Framework. З нової платформи з різних причин було виключено низку технологій. Слід розуміти, що платформа .NET Core розрахована насамперед на розробку для серверних та хмарних рішень. Для десктопних програм краще підходять класичний .NET для Windows (з підтримкою WPF та Windows Forms). Мобільні проєкти можна створювати за допомогою Xamarin.[5] На схемі(рис 2.1) показано, як технології розподілені всередині різних реалізацій .Net.

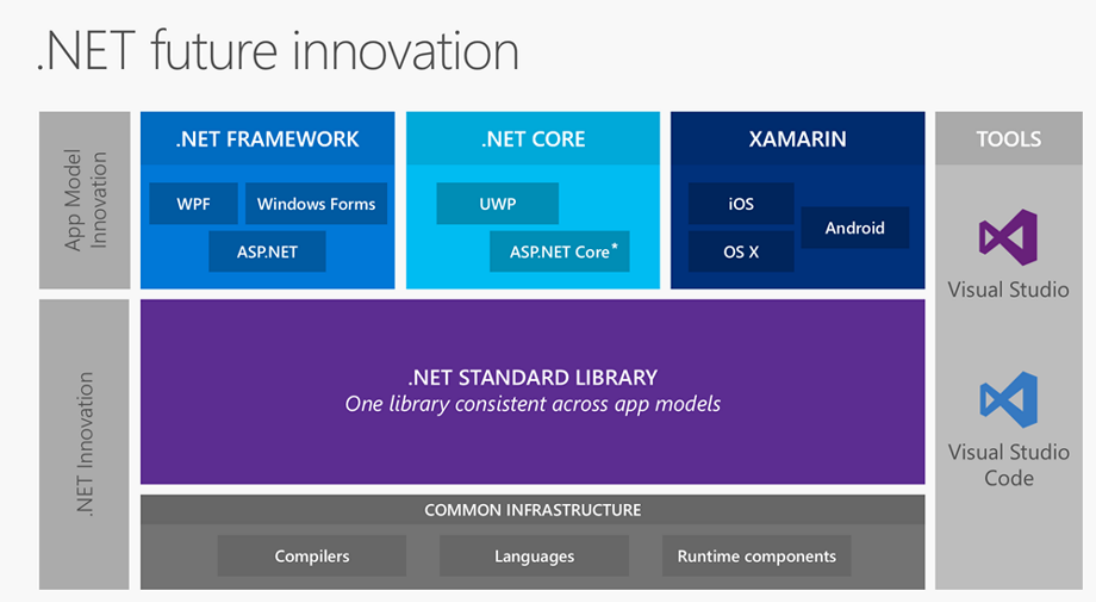


Рисунок 2.1 – Схема технологій .Net

Виходячи з усього вищесказаного, можна зробити висновок, що .NET Core дозволяє невеликим проєктам і стартапам отримати всі переваги платформи корпоративного рівня, при цьому надаючи зручні засоби розробки, а також недорогу інфраструктуру. У майбутньому ж неминучий прихід платформи і великий корпоративний ринок.

2.1.2 Вибір мови програмування

.NET підтримує три мови програмування:

– C#
– F#
– Visual Basic

C#

C# — сучасна, об’єктно-орієнтована та безпечна для типів мова програмування. C# має свої коріння в сімействі мов C і буде відразу знайомий програмістам C, C++, Java та JavaScript.

C# підходить для створення та застосування програмних компонентів. З моменту створення мова C# збагатилася функціями для підтримки нових робочих

навантажень та сучасними рекомендаціями щодо розробки ПЗ. В основному C# - об'єктно-орієнтована мова. Ви визначаєте типи та їх поведінку[6].

Деякі функції мови:

- Збір сміття: автоматично звільнює пам’ять, зайняту недосяжними об'єктами, що не використовуються.
- Типи, що допускають значення null: забезпечують захист від змінних, які посилаються на виділені об'єкти.
- Обробка винятків: надає структурований та розширюваний підхід до виявлення помилок.
- Лямбда-вираження: підтримують прийоми функціонального програмування.
- Асинхронні операції: забезпечує синтаксис для створення розподілених систем
- Синтаксис LINQ: створює спільний шаблон для роботи з даними будь-якого джерела

F#

F# — мультипарадигмальна мова програмування із сімейства мов .NET, що підтримує функціональне програмування на додаток до імперативного (процедурного) та об'єктно-орієнтованого програмування.

F# — це універсальна мова програмування для написання короткого, надійного та продуктивного коду. Дозволяє писати простий, самодокументований код, де ви зосереджуєтесь на проблемній області, а не на деталях програмування[7].

Функції мови:

- Легкий синтаксис
- Незмінність за замовчуванням
- Виводження типу та автоматичне узагальнення
- Функції першого класу
- Регулярні вирази

– Асинхронне програмування

Visual Basic

Visual Basic .NET (VB.NET) — об'єктно-орієнтований язык програмування, який можна розглядати як черговий вік еволюції Visual Basic (VB), реалізований на платформі .NET Framework. VB.NET не має зворотної сумісності з більш ранньою версією (Visual Basic 6.0)[8].

Як і всі інші мови .NET, VB.NET повністю підтримують объектно-орієнтовані концепції. Все у VB.NET є об’єктом, включаючи всі примітивні типи (Short, Integer, Long, String, Boolean тощо) І визначають використовувані типи, події та навіть збірки. Все об'єкти наследуются від базового класу Object.

Деякі функції мови:

– Булеві умови
– Автоматичний збір сміття
– Стандартна бібліотека
– Властивості та події
– Делегати
– Індексатори
– Багатопоточність

Проаналізувавши існуючі мови програмування було обрану мову C# оскільки у цій мові використовується імперативна парадигма програмування(тобто пишуться інструкції для комп’ютера), а для реалізації проєкту необхідна робота із об’єктами.

2.1.3 Вибір інструментів для розробки інтерфейсу користувача

Серед найпопулярніших фреймворків для розробки користувацьких інтерфейсів можна виділити:

– Vue.js
– React

– Angular 13

Vue.js

Vue.js - це фреймворк JavaScript для створення інтерфейсів користувача. Він побудований на основі стандартних HTML, CSS і JavaScript і надає декларативну модель програмування на основі компонентів, яка допомагає вам ефективно розробляти інтерфейси користувача, як прості, так і складні[9].

До переваг даного фреймворку можна віднести наступне:

– Можливість оптимізувати обробку HTML-блоків з використанням різних компонентів.
– Докладна документація.
– Масштабування: дозволяє розробляти великі шаблони для повторного використання
– Можливість використання фреймворку для односторінкових додатків та для більш важких веб-сервісів.

Недоліки фреймворку:

– Недостатня кількість ресурсів
– При інтеграції рішень у більші проєкти можуть виникати проблеми

React

React - це JavaScript-бібліотека для розробки інтерфейсу користувача. React спочатку був спроектований так, щоб його можна було поступово впроваджувати[10].

Переваги:

– Простий синтаксис
– Висока гнучкість
– Відкритий код бібліотеки
– Проста міграція між версіями

Недоліки:

– Недостатня кількість документації
– Бібліотека не має чіткої цілі

– Необхідно більше часу для засвоєння

Angular 13

Angular — це фреймворк дизайну додатків і платформа розробки для створення ефективних і складних програм[11].

Переваги:

– Функції новіші ніж у аналогів
– Докладна документація
– Можливість працювати в одному розділі програми з використанням одного і того ж набору даних.
– Впровадження залежностей функцій пов'язаних із компонентами з модулями та модульності в цілому.

Недоліки:

– Важкий синтаксис
– Проблеми з міграціями між версіями

2.1.4 Середовище розробки

Project Rider - IDE від компанії JetBrains, тієї самої компанії, яка створила ReSharper, напевно, найпопулярніший плагін для Visual Studio. В основі Rider лежать IntelliJ IDEA (яка є основною для цілого сімейства IDE) та напрацювання, що використовуються в ReSharper. Зараз Rider підтримує розробку як під Mono, так і під .NET Core[12].

На рисунку 2.2 зображено як виглядає середовище розробки Project Rider.

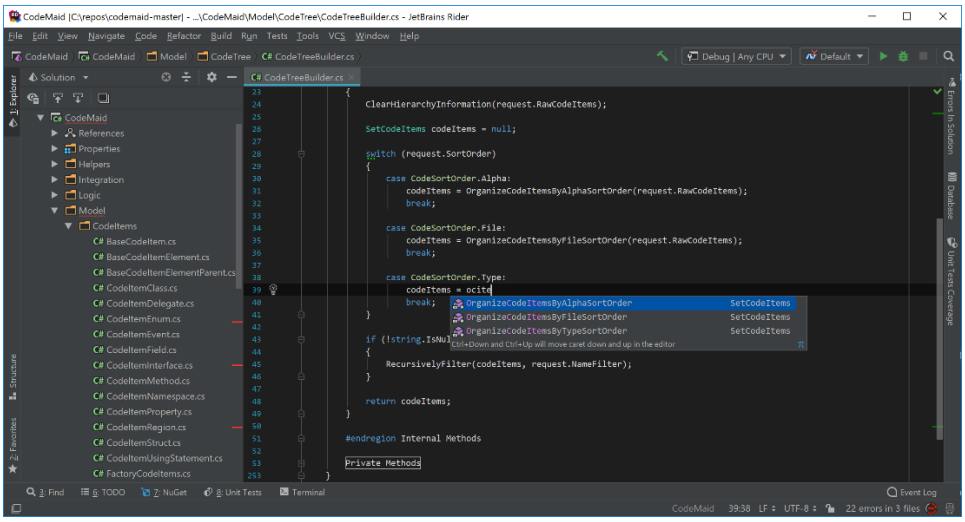


Рисунок 2.2 – Середовище розробки Project Rider

Функціональність:

– Аналіз коду: Rider надає більше 2200 інспекцій коду та автоматизованих виправлень. Механізм аналізу помилок по всьому рішенню шукатиме помилки в кодовій базі та повідомляти про них, навіть якщо проблемний файл не відкритий у редакторі.
– Редагування коду: редактор Rider надає різні види автодоповнення та шаблонів, автоматично вставляє парні дужки та імпортує відсутні простору імен.
– Рефакторинг: запозичує з ReSharper понад 60 рефакторингів та передбачає понад 450 контекстних дій
– Інструменти запуску юніт-тестів
– Відладчик: Вбудований налагоджувач для додатків на .NET Framework, Mono та .NET Core підтримує покрокове виконання
– Робота з базами даних і SQL: можна працювати із зазначеними технологіями у самій IDE.
– Підтримка фронтенд-технологій: Rider підтримує JavaScript, TypeScript, HTML, CSS та Sass.

2.2 Система керування версіями

Система керування версіями — програмний інструмент для керування версіями одиниці інформації. Розробники можуть паралельно проводити роботу над груповим проєктом, не заважаючи один одному. Також СКВ дозволяють легко відстежувати зміни у коді або документах. [13].

Серед найпопулярніших СКВ:

- Git - розподілена система керування версіями. Проєкт був створений Лінусом Торвальдсом для керування розробкою ядра Linux[14].
- SVN – безкоштовна система контролю версій з відкритим кодом. Допускає атомарні операції, операції з розгалуженням коду менш затратні, має широкий вибір плагінів IDE[15].
- Mercurial - кроссплатформна розподілена система керування версіями, розроблена для ефективної роботи з дуже великими репозиторіями коду. Насамперед вона є консольною програмою. Має докладну документацію та легка для засвоєння[16].

Для керування версіями даного проєкту було обрано систему Git, а для збереження коду сервіс GitHub.

В основу Git закладалися концепції, покликані створити швидшу розподілену систему контролю версій.

Переваги:

- Значне збільшення швидкодії.
- Дешеві операції з гілками коду.
- Повна історія розробки доступна офлайн.
- Розподілена модель.

Недоліки:

- Високий поріг входження (навчання).
- Обмежена підтримка Windows (порівняно з Linux).

GitHub, Inc. – провайдер інтернет-хостингу для розробки програмного забезпечення і контролю версій за допомогою Git. Він пропонує розподілену систему контролю версій і управління вихідного коду функції (SCM) в Git. А також надає ряд функцій для контролю своїх проєктів та спільної розробки.

2.3 Вибір засобів для збереження даних

При написанні програм в любій сфері неможливо обійтись без використання баз даних, оскільки навіть невеликі системи потребують зберігання та відтворення даних.

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; [17].

При проєктуванні системи було обрано реляційну СКБД, оскільки сервіс має строгу і стабільну структуру.

Реляційна база даних — це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеня.

Реляційна база даних є сукупністю елементів даних, організованих у вигляді набору формально описаних таблиць, з яких дані можуть бути доступними або повторно зібрані багатьма різними способами без необхідності реорганізації таблиць бази даних.[18]

Існують наступні реляційні системи керування даними:

- MySQL
- PostgreSQL
- MSSQL Server

MySQL

MySQL — вільна система керування реляційними базами даних, яка була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам[19].

Переваги:

- Простота у використанні: досить легко встановлюється, а наявність безлічі плагінів та допоміжних програм спрощує роботу з базами даних.
- Великий функціонал: має практично весь необхідний інструментарій, який може знадобитися в реалізації практично будь-якого проєкту.
- Безпека: система спочатку створена в такий спосіб, що багато вбудованих функцій безпеки у ній працюють за умовчанням.
- Масштабованість: однаково легко можна використовувати для роботи і з малими, і з великими обсягами даних.
- Швидкість: висока продуктивність системи забезпечується за рахунок спрощення деяких стандартів, що використовуються в ній.

Недоліки:

- Недостатня надійність. У питаннях надійності деяких процесів роботи з даними MySQL поступається деяким іншим СУБД.
- Низька швидкість розробки: системі не вистачає деякої технічної досконалості, що часом позначається на ефективності процесів розробки.

PostgreSQL

PostgreSQL – вільна об’єктно-реляційна СКБД. Існує в реалізації для більшості UNIX-подібних систем, а також для сімейства ОС Windows. Базується на мові SQL та підтримує більшість можливостей стандарту SQL.[20]

Переваги:

- Легка у використанні.
- Відкритий код.
- Велика підтримка спільноти.
- Використовує процедури, що зберігаються.
- Пвдтримка ACID, тобто. атомарність, узгодженість, ізоляція, довговічність.

Недоліки:

- Для кожного клієнта створюється окремий сервіс, що призводить до використання великої кількості пам'яті.

– Погана продуктивність.
– Важке встановлення.

MSSQL Server

Microsoft SQL Server – популярна система керування базами даних (СУБД), розроблена корпорацією Майкрософт. Доступна у кількох редакціях. Може працювати на ПК, ноутбуці, сервері, на віртуальній машині або у хмарі.[21]

Переваги:

– Масштабування системи: взаємодіяти з нею можна як на простих ноутбуках, так і на комп'ютері з потужним процесором, який здатний обробляти великий обсяг запитів.
– Дані отримуються швидко, а складну інформацію зручніше зберігати. Система обробляє транзакції в інтерактивному режимі, є динамічне блокування.
– Зручний пошук. Його можна здійснювати за фразами, словами, тексту чи створювати ключові індекси.
– Підтримка роботи з іншими рішеннями Microsoft, у тому числі з Excel, Access.

Недоліки:

– Залежність від ОС. Система працює лише з Windows.

Отже, проаналізувавши існуючі системи керування базами даних для розробки даної дипломної роботи було обрано MSSQL Server.

Робота з MS SQL та додаток для роботи з нею Microsoft SQL Server Manager зображені на рис. 2.3.

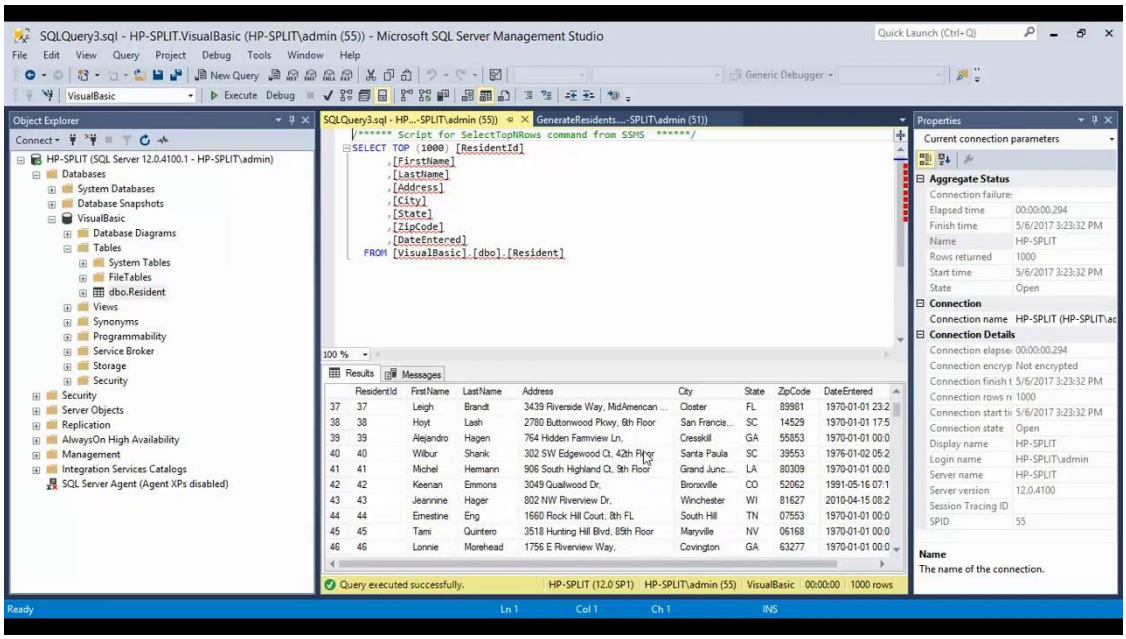


Рисунок 2.3 – Робота з СКБД MS SQL та програмний засіб для роботи з нею Microsoft SQL Server Manager

2.4 Шифрування

Криптографічні алгоритми поділяють на безключеві, з одним ключем, та з двома ключами:

– Безключові алгоритми не використовують у обчисленнях ніякі ключі;
– Алгоритми з одним ключем використовують один ключовий параметр(або секретний ключ);
– Алгоритми що використовують два ключі: секретний(приватний) та відкритий(публічний).

До безключових алгоритмів відносяться хеш-функції, тобто функції, що перетворюють вхідні дані будь-якого розміру в дані фіксованого, а результатом обробки є хеш-сума. Найпопулярнішими представниками цього типу є алгоритм MD5, SHA-1, SHA-2 та SHA – 3, bcrypt тощо.

Алгоритми з одним ключем(або симетричні) в свою чергу поділяють на блочні та потокові. Блочні шифри оброблюють інформацію блоками(64 – 256 біт), застосовуючи до блоку ключ у встановленому порядку, зазвичай декількома циклами змішування та підстановки. Потокові шифри – шифри, в яких

шифрування виконується над кожним бітом вихідного тексту з використанням шифру XOR, тобто накладанні послідовності, що складається з випадкових чисел на вихідний текст. Симетричні алгоритми використовують один ключовий параметр(секретний ключ) для шифрування та розшифровки даних. Використання одного ключа для цих операцій робить процес простим та ефективним. Одним з головних недоліків цього типу алгоритмів є передача секретного ключа по спеціальних захищених каналах задля запобігання заволодіння його третіми особами. Типовими та найпопулярнішими представниками являються алгоритми: DES, AES, Blowfish, RC4, RC5 тощо[22].

Алгоритми з двома ключами(або асиметричні) – алгоритми, що використовують різні ключі для шифрування та розшифрування даних. Публічний ключ здатний лише шифрувати дані або перевіряти електронний цифровий підпис(далі ЕЦП). Цей ключ передається по відкритому(незахищеному, доступному для спостерігання) каналу тому всі заохочені можуть ним скористатися. Приватний ключ має ті самі властивості що й публічний та має доповнення у вигляді розшифровки даних та генерацію ЕЦП. Це дозволяє людям, що не мають домовленостей про безпеку, обмінюватися секретними даними.

Проаналізувавши існуючі криптографічні алгоритми для розробки проєкту було вирішено обрати алгоритм, що використовує два ключі, а саме асиметричний алгоритм RSA(абревіатура від прізвищ Rivest, Shamir та Adleman)[23].

RSA стоїть біля витоків асиметричної криптографії. Він став першим асиметричним алгоритмом придатним як для шифрування так і для електронного цифрового підпису[24].

3 РОЗРОБКА СТРУКТУРИ ВЕБ-СЕРВІСУ НОТАТОК

3.1 Вибір архітектури для розробки веб-сервісу

У створенні розподілених мережних застосунків домінуючою архітектурою є архітектура «клієнт-сервер»(рис 3.1).

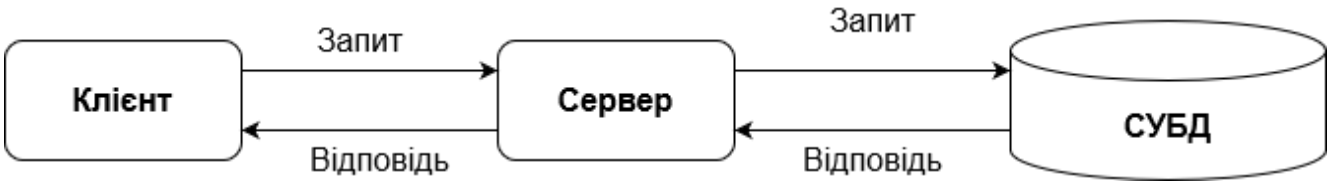


Рисунок 3.1 – Загальна схема архітектури «клієнт-сервер»

Архітектура передбачає наступний набір компонентів:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
– набір клієнтів, які використовують сервіси, що надаються серверами;
– мережа, яка забезпечує взаємодію між клієнтами та серверами. [25].

Розроблений проєкт буде містити два сервери, один з яких відповідає за аутентифікацію користувачів, інший – сам сервіс(рис. 3.2).

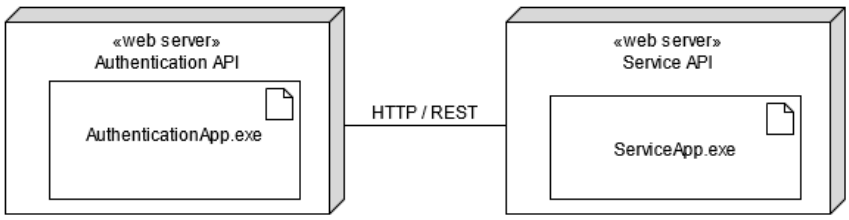


Рисунок 3.2 - Діаграма розгортання

Це дає можливість в подальшому розширити функціонал проєкту на декілька сервісів, при цьому, використовуючи лише один акаунт на всі сервіси(рис 3.3).

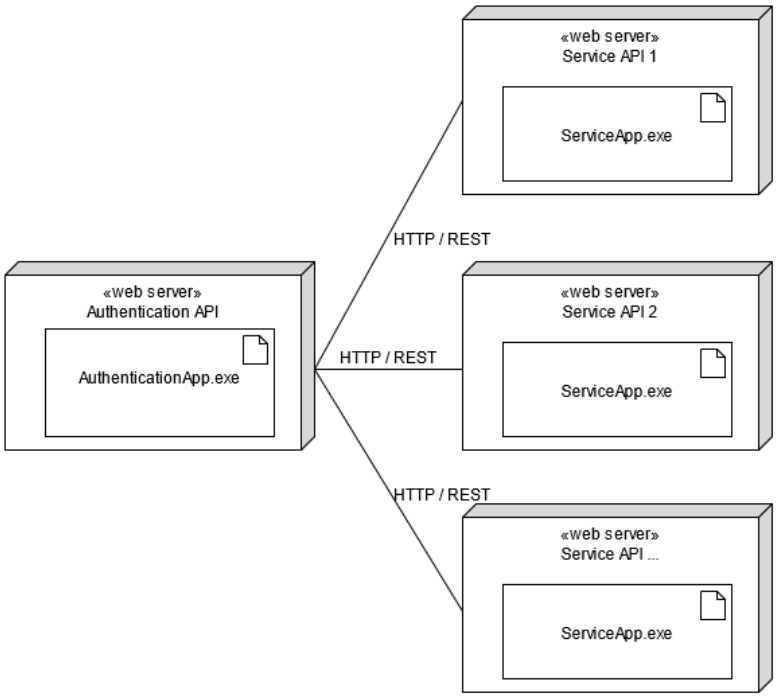


Рисунок 3.3 - Розширена діаграма розгортання

Для роботи з системою користувач використовує стандартне програмне забезпечення – звичайний браузер. Веб-оглядач формує запит та пересилає його до ресурсного сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних.

При реєстрації або авторизації користувача дані у зашифрованому(публічним ключем) вигляді передаються на аутентифікаційний сервер, де вони спочатку розшифровуються(приватним ключем) потім виконується їх валідація. У разі успішного проходження валідації, у випадку реєстрації нового користувача, дані хешуються за допомогою хеш-функції і зберігаються на аутентифікаційному сервері. Після перевірки формується відповідь сервера з певним набором даних, підписаних ЕЦП. Сервер сервісу за допомогою публічного ключа перевіряє валідність цього підпису та надає певний функціонал.

3.2 Проєктування системи

Для проєктування веб-сервісу було обрано уніфіковану мову моделювання UML. Для розроблення діаграм класів, кооперації, діяльності, компонентів та розгортання було використано методологію об’єктно-орієнтованого аналізу і проєктування та мову UML.

Спочатку створення UML було мотивоване бажанням стандартизувати розрізнені системи позначень і підходи до проєктування програмного забезпечення. Він був розроблений в Rational Software в 1994-1995 рр., А їх подальша розробка велася до 1996 р.

Unified Modeling Language (UML) – уніфікована мова моделювання. Розшифруємо: modeling передбачає створення моделі, що описує об'єкт. Unified (універсальний, єдиний) - підходить для широкого класу проєктованих програмних систем, різних галузей додатків, типів організацій, рівнів компетентності, розмірів проєктів. UML описує об'єкт в єдиному заданому синтаксисі, тому де б не створювалася діаграма, її правила будуть зрозумілі для всіх, хто знайомий із цією графічною мовою [26].

Види UML-діаграм, які описані у даній роботі:

- Діаграма прецедентів
- Діаграма класів
- Діаграма послідовності
- Діаграма діяльності
- Діаграма розгортання
- Діаграма пакетів
- Діаграма предметної області
- Діаграма станів меню
- Діаграма станів

3.3 Функціональні можливості користувачів

Для роботи користувачів веб-сервісу розмежування доступу здійснюється шляхом дозволу або заборони на редагування та перегляд певної інформації.

Більш детально усі види користувачів описані у таблиці 3.1.

Таблиця 3.1 – Опис діючих осіб

Актор	Опис
Користувач-гість	Будь-яка людина, яка відвідала сайт сервісу. Має лише можливості перегляду публічних нотаток та загальні можливості роботи з веб-системою.
Авторизований користувач	Авторизований користувач має такі ж можливості, як гість, та додатково може переглядати власний акаунт та створювати приватні нотатки, які будуть доступні у акаунті.
Модератор	Модератора назначає адміністратор. Може переглядати усі необроблені нотатки та здійснювати їх перевірку, після чого публікувати у загальний доступ.
Адміністратор	Адміністратор може встановлювати ролі користувачам та управляти ними.

Для наглядного розуміння функцій кожного актору було створено діаграму прецедентів, що зображені не рис. 3.4.

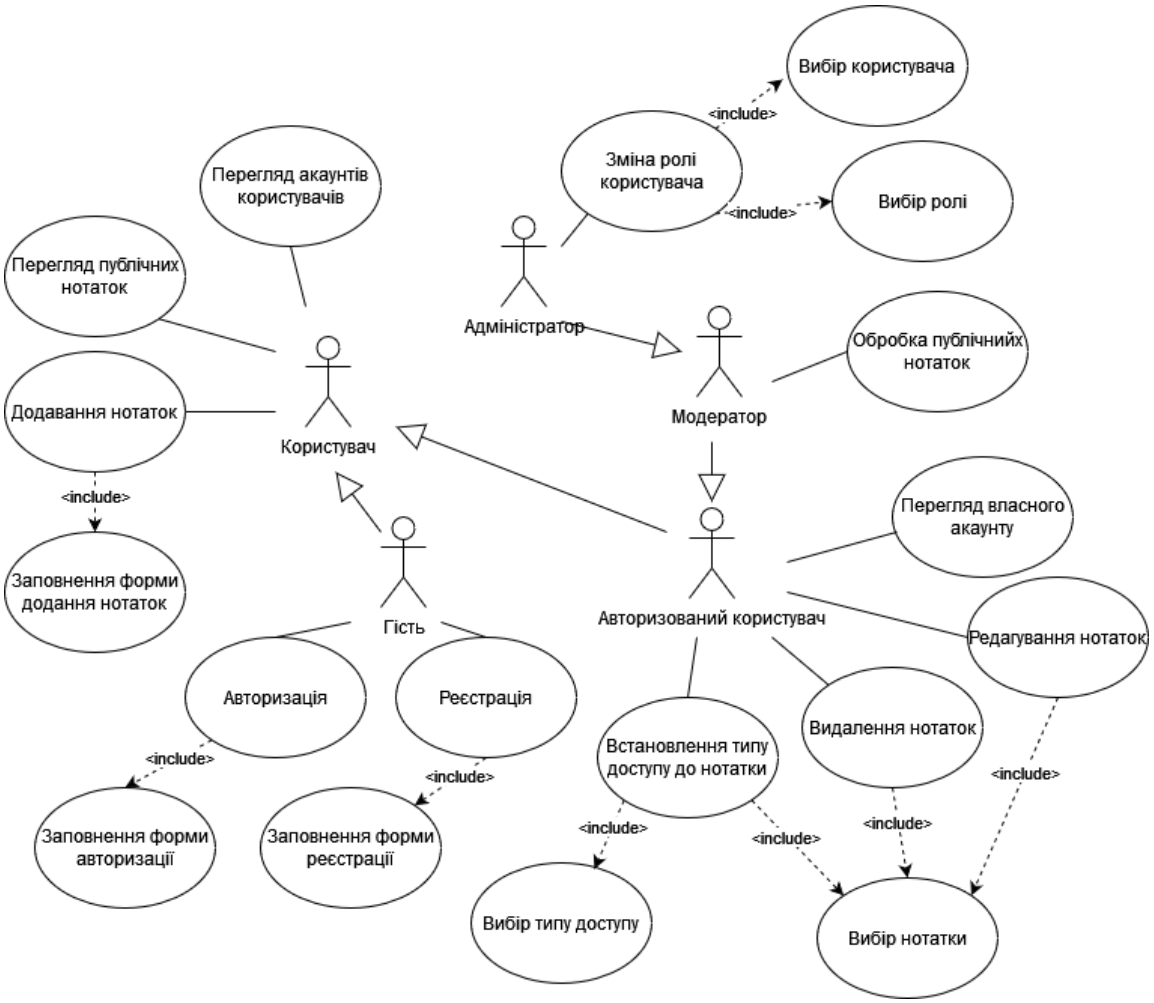


Рисунок 3.4 – Діаграма прецедентів сервісу

На діаграмі можна побачити, що актор «Гість» має базовий набір функцій, який мають усі інші актори. Авторизований користувач успадковує від гостя всі базові функції та має свої розширені. Так само Модератор переймає всі функції авторизованого користувача та має свою унікальну функцію обробки публічний нотаток. Адміністратор містить всі функції інших акторів та свою унікальну – можливість змінювати ролі користувачам.

На діаграмі послідовностей(рис. 3.5) продемонстровано процес додання нотатки будь-яким користувачем. Спочатку користувач відправляє запит на створення нотатки, заповнює поля нотатки та відправляє запит на ресурсний сервер, де перевіряється аутентифікація та відбувається перевірка полів. У разі правильності заповнення полів ресурсний сервер зберігає нотатку у базі даних.

Рисунок 3.5 – Діаграма послідовностей прецеденту «Створення нотатки»

Після того, як визначено типи користувачів та їх функціональні можливості можна перейти до проєктування бази даних.

3.4 Проєктування бази даних сервісу

Для бази даних аутентифікаційного серверу даного веб-сервісу були визначені наступні сутності:

- Користувач
- Токен оновлення

Для бази даних ресурсного серверу даного веб-сервісу були визначені наступні сутності:

- Користувач;
- Роль користувача;
- Нотатка;
- Необроблена(немодерована) нотатка.

Кожна таблиця має містити стовпець або набір стовпців, які однозначно ідентифікують кожний рядок, що зберігається в таблиці. Часто це унікальний ідентифікаційний номер, наприклад номер паспорта працівника або серійний номер. У термінології баз даних така інформація називається первинним ключем таблиці[27].

Первинний ключ завжди має містити значення. Якщо на певному етапі значення в стовпці може стати непризначеним або невідомим (значення відсутнє), його не можна використовувати як компонент первинного ключа.

Список сутностей бази даних веб-сервісу нотаток представлені у таблицях 3.2-3.3.

Таблиця 3.2 – Список сутностей БД аутентифікаційного серверу сервісу ведення нотаток.

Назва сутності БД	Опис сутності
Users	Містить інформацію про всіх користувачів сервісу.
RefreshTokens	Містить інформацію про токени оновлення всіх користувачів.

Таблиця 3.3 – Список сутностей БД ресурсного серверу сервісу ведення нотаток

Назва сутності у БД	Опис сутності
Users	Містить інформацію про всіх користувачів сервісу.
Roles	Містить усі види ролей користувача(звичайний користувач, модератор, адміністратор).
Notes	Містить інформацію про нотатки всіх користувачів.
RawNotes	Таблиця, у яку поміщаються публічні нотатки для подальшої модерації, перенесення інформації до таблиці «Notes» та відображення користувачам.

Далі визначимо атрибути для кожної таблиці та сформуємо структури баз даних(табл. 3.4 – 3.9).

Таблиця 3.4 – Структура сутності «Users» бази даних аутентифікаційного сервера.

Назва поля	Назва латиницею	Тип	Ключ	Примітка
Ідентифікатор	Id	Guid	PK	Not null
Ім’я	FirstName	nvarchar	-	Not null
Прізвище	LastName	nvarchar	-	Not null
Псевдонім	Nickname	nvarchar	-	Not null
Пошта	Email	nvarchar	-	Not null
Пароль	Password	nvarchar	-	Not null

Таблиця 3.5 – Структура сутності «RefreshTokens» бази даних аутентифікаційного сервера.

Назва поля	Назва латиницею	Тип	Ключ	Примітка
Ідентифікатор	Id	Guid	PK	Not null
JWT ідентифікатор	Jti	Guid	-	Not null
Ідентифікатор користувача	UserId	Guid	FK	Not null
Є невалідним	IsInvalidated	Bit	-	Not null
Є використаним	IsUsed	Bit	-	Not null
Дата створення	Created	DateTime	-	Not null
Дата закінчення строку придатності	Expiry	DateTime	-	Not null

Таблиця 3.6 – Структура сутності «Users» бази даних ресурсного сервера

Назва поля	Назва латиницею	Тип	Ключ	Примітка
Ідентифікатор	Id	Guid	PK	Not null
Ім’я	FirstName	nvarchar	-	Not null
Прізвище	LastName	nvarchar	-	Not null
Псевдонім	Nickname	nvarchar	-	Not null
Пошта	Email	varchar	-	Not null
Ідентифікатор ролі	RoleId	Guid	FK	Not null

Таблиця 3.7 – Структура сутності «Roles» бази даних ресурсного сервера

Назва поля	Назва латиницею	Тип	Ключ	Примітка
Ідентифікатор	Id	Guid	PK	Not null
Назва	Name	varchar	-	Not null

Таблиця 3.8 – Структура сутності «Notes» бази даних ресурсного сервера

Назва поля	Назва латиницею	Тип	Ключ	Примітка
Ідентифікатор	Id	Guid	PK	Not null
Заголовок	Title	varchar	-	Not null
Текст нотатки	Text	varchar	-	Not null
Час існування нотатки	LifeTime	DateTime	-	Null
Тип доступу	AccessType	Int	-	Not null
Ідентифікатор користувача	UserId	Guid	FK	Null
Дата створення	Created	DateTime	-	Not null
Автор	CreatedBy	varchar	-	Not null
Дата модифікації	LastModified	DateTime	-	Null
Автор модифікації	LastModifiedBy	varchar	-	Null

Таблиця 3.9 – Структура сутності «RawNotes» бази даних ресурсного сервера

Назва поля	Назва латиницею	Тип	Ключ	Приміт ка
Ідентифікатор	Id	Guid	PK	Not null
Заголовок	Title	varchar	-	Not null
Текст нотатки	Text	varchar	-	Not null
Час існування нотатки	LifeTime	DateTime	-	Null
Ідентифікатор користувача	UserId	Guid	FK	Null
Дата створення	Created	DateTime	-	Not null
Автор	CreatedBy	varchar	-	Not null
Дата модифікації	LastModified	DateTime	-	Null
Автор модифікації	LastModifiedBy	varchar	-	Null

Після опису структур сутностей необхідно встановити зв’язки між таблицями БД.

Зв’язок між таблицями — це відношення між загальними полями (стовпцями) у двох таблицях. Зв’язок може бути таких типів: зв’язок «один-до-одного», зв’язок «один-до-багатьох» або зв’язок «багато-до-багатьох». [28]

Під час створення зв’язку між таблицями або додавання об’єднання до запиту зв’язуванні поля повинні мати однакові або сумісні типи даних. Наприклад, не можна створити об’єднання між числовими та текстовими полями, навіть якщо значення в цих полях збігаються.

Для демонстрації зв’язків між таблицями було створено концептуальну модель бази даних ресурсного та аутентифікаційного серверів(рисунки 3.6 – 3.7).

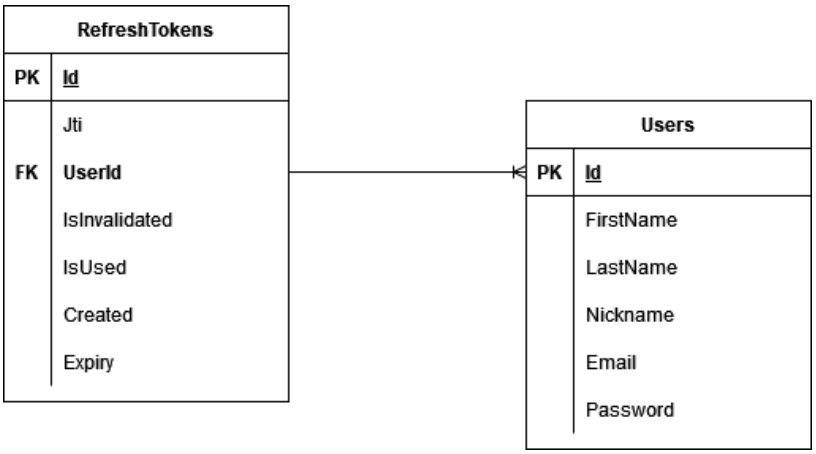


Рисунок 3.6 – Схема бази даних аутентифікаційного серверу сервісу

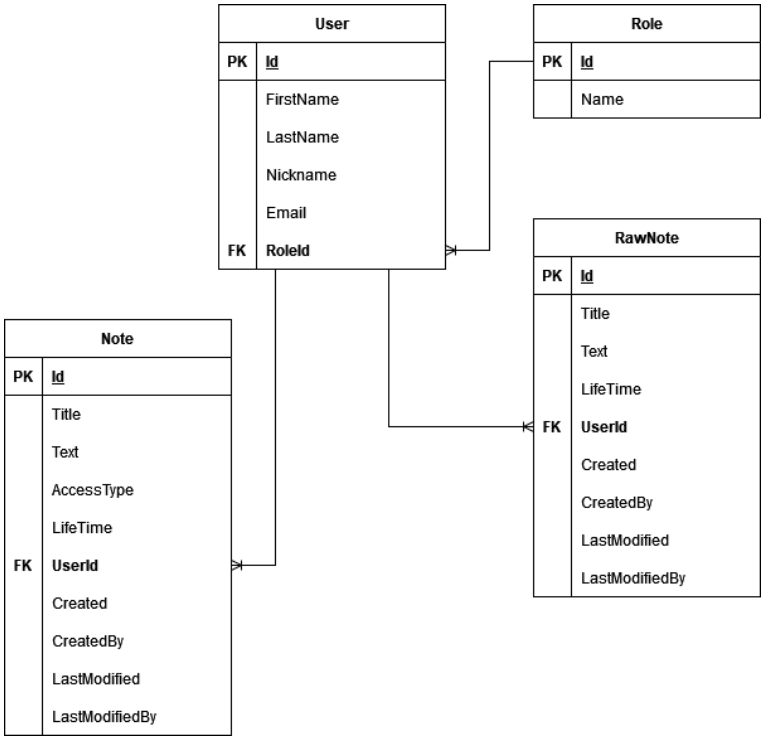


Рисунок 3.7 – Схема бази даних ресурсного серверу сервісу

Відповідно до структури сутностей та встановленим зв’язкам створені бази даних відповідають третій нормальній формі, оскільки бази даних відповідають всім критеріям[29], а саме:

- Кожна таблиця має первинний ключ.
- Відсутність повторень груп.
- Кожен атрибут має одне значення.

– Дані, що повторно зустрічаються у декількох записах виносяться в окремі таблиці.
– Всі поля, що залежать від основного ключа винесено в окремі таблиці.

Також для демонстрації взаємозв’язку на діаграмі предметної області(рис. 3.8.) зображено об’єкти проєкту.

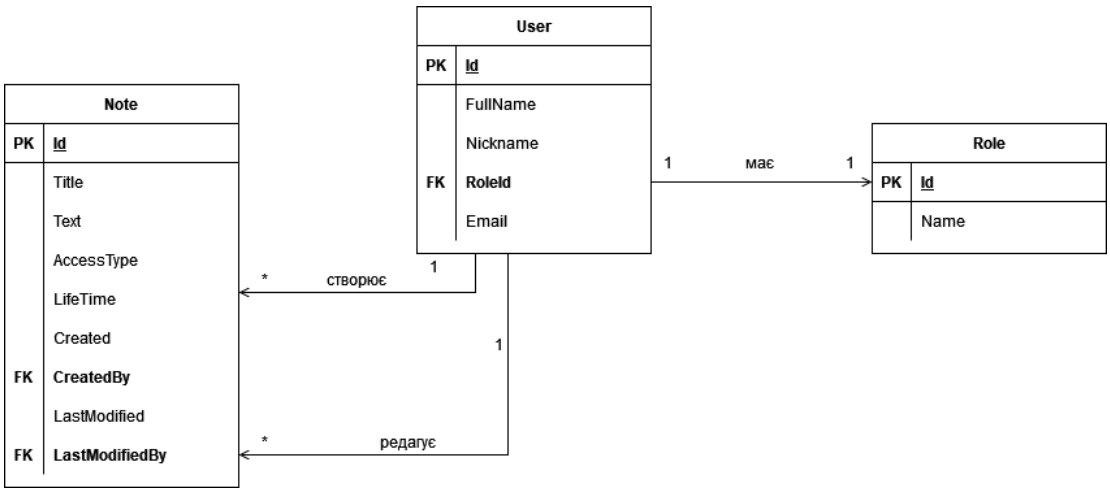


Рисунок 3.8 – Діаграма предметної області

3.5 Проєктування класів сервісу

Для написання веб додатків необхідно правильно обирати шаблон проєктування програмного забезпечення. Архітектура самого проєкту повинна бути зрозумілою розробнику та структурованою за певними принципами для подальшого простого оновлення або доробки системи.

При проєктуванні було вирішено обрати «Чисту архітектуру». До основних ідей чистої архітектури відносяться:

– Незалежність від фреймворків: архітектура не залежить від існування будь-якої бібліотеки.
– Можливість тестування: бізнес-правила повинні тестуватися без користувацького інтерфейсу, бази даних, веб-серверу або будь-якого іншого зовнішнього компоненту.

– Незалежність від UI: Користувацький інтерфейс можна змінювати, не змінюючи всю іншу систему.
– Незалежність від бази даних: Бізнес правила не пов’язані із базою даних. Можна обрати будь-яку базу даних та змінити у будь-який момент.
– Незалежність від зовнішніх сервісів: бізнес правила не повинні знати про існування зовнішніх інтерфейсів[30].

Схему розташування шарів відображено на діаграмі(рис. 3.9).

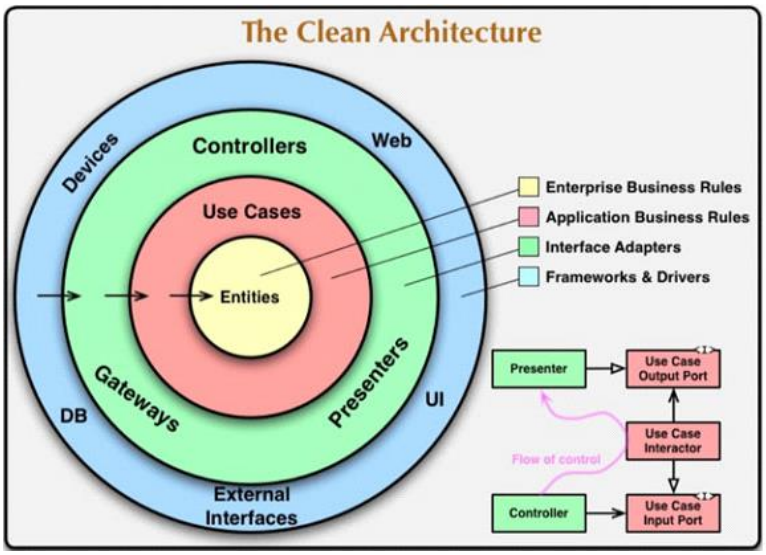


Рисунок 3.9 – Схема чистої архітектури.

На схемі можна виділити наступні шари, на які повинен ділитися проєкт:

– Сутності.
– Сценарії.
– Інтерфейс-Адаптери.
– Фреймворки та драйвера[31].

Сам автор у своїй книзі писав, що дана схема не є строгою і необхідно лише притримуватися принципів чистої архітектури, а особливо правилу залежностей. Це правило свідчить, що залежності у вихідному коді можуть вказувати лише усередину. Ніщо з внутрішнього кола не може знати щось про зовнішнє коло, ніщо з внутрішнього кола не може вказувати на зовнішнє коло.

Тому для реалізації даного проєкту було спроектовано наступні шари, які можна побачити на рисунку 3.10.

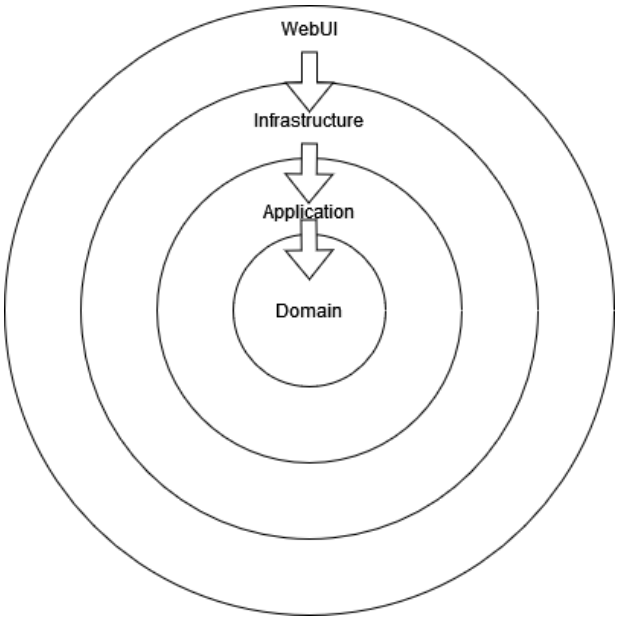


Рисунок 3.10 – Схема архітектури проєкту сервісу

Рівень інтерфейсу та інфраструктури залежать від ядра(Domain), але ядро не залежить від жодного рівня. Це досягається шляхом додавання абстракцій або інтерфейсів всередині прикладного рівня, які потім реалізуються за межами прикладного рівня на інших.

Шар Domain відповідає за зв’язок із базою даних та містить представлення всіх таблиць бази даних. Класи-сутності даного шару:

- AccessType – перерахування(тип enum) – містить всі типи доступу до нотаток, а саме(приватні, публічні та за посиланням). Не являється сутністю, оскільки потреби у цьому немає.
- AuditableEntity – абстрактний клас, що містить інформацію лише про редагування та створення нотаток.
- Note – клас наслідується від AuditableEntity та відповідає сутності «Notes», містить такі ж поля як в таблиці, що були описані у пункті 3.3.
- Pagination
- PaginationResult

– RawNote – клас, що містить такі ж поля, як звичайна нотатка. Існує для того, щоб модератори мали можливість перевіряти вміст тексту публічних нотаток. На діаграмі станів нотатки(рис. 3.11) можна побачити процес додання нотатки.

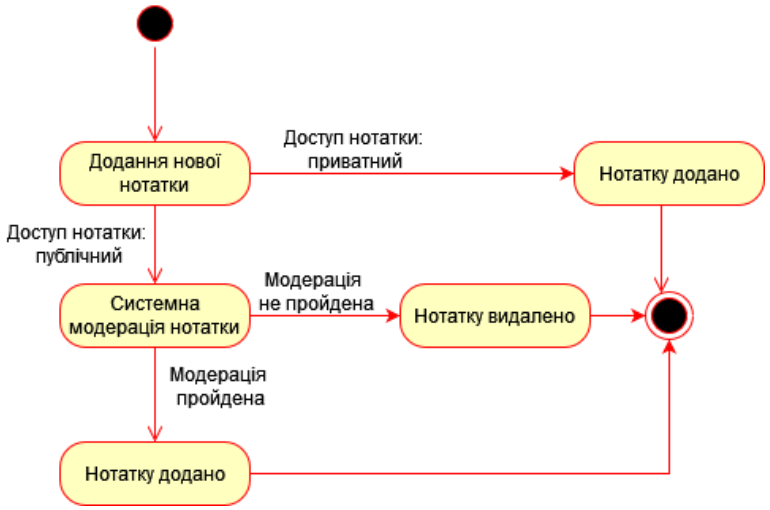


Рисунок 3.12 – Діаграма станів нотатки

– Role
– User

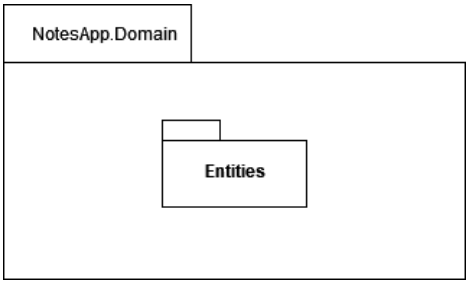


Рисунок 3.13 – Діаграма пакетів шару Domain

Шар Application містить інтерфейси, які потім реалізуються на зовнішніх рівнях, таких як Infrastructure, DTO-моделі, валідатори, винятки.

Структура даного рівня:

– Dto – папка, в якій зберігаються всі DTO-моделі сервісу.
– Exceptions – папка, що містить класи, які оброблюють винятки.

– Mapping – папка, що містить класи налаштувань перетворнеь(див. пункт 4.4.2)
– Persistence – папка, в якій зберігаються інтерфейси репозиторіїв та інтерфейс UnitOfWork(див. пункт 4.3).
– Services – папка, яка містить інтерфейси всіх сервісів проєкту.
– Validators – папка, що містить класи налаштування валідаторів(див. пункт 4.4.1).

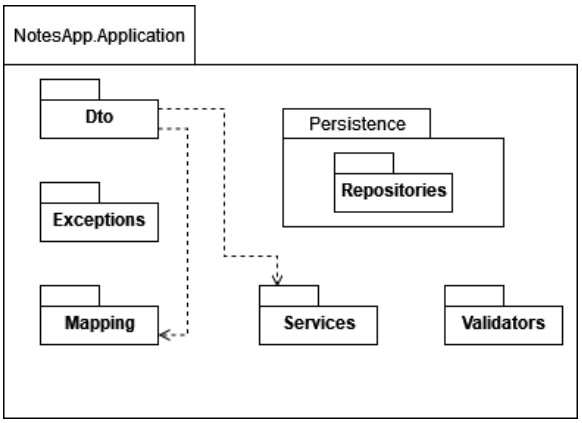


Рисунок 3.14 – Діаграма пакетів шару Application

Шар Infrastructure реалізує всі інтерфейси із шару Application, тобто містить реалізацію репозиторіїв, сервісів, UnitOfWork, також на даному рівні міститься контекст бази даних, налаштування моделей та зберігаються міграції проєкту(більш детально описано у пункті 4.1).

Структура даного рівня:

– Migrations
– Persistence
  o Configurations
  o Database
  o Repositories
  o UnitOfWork
– Services

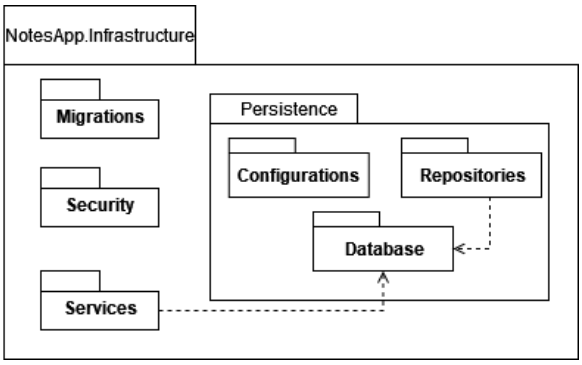


Рисунок 3.15 – Діаграма пакетів шару Infrastructure

Шар WebUI являється ASP.NET Core Web Application, містить класи, що використовуються на клієнті. На даному рівні повинні бути реалізовані контролери, класи налаштування перетворення, Request-моделі. Слід відмітити, що при доданні поточного проєкту за замовчування створюються певні файли з налаштуваннями. Наприклад клас Startup.cs містить налаштування залежносей проєкту.

Структура даного рівня:

– Properties
– Controllers
– Mapping
– Requests
– appsettings.json
– Program.cs
– Startup.cs

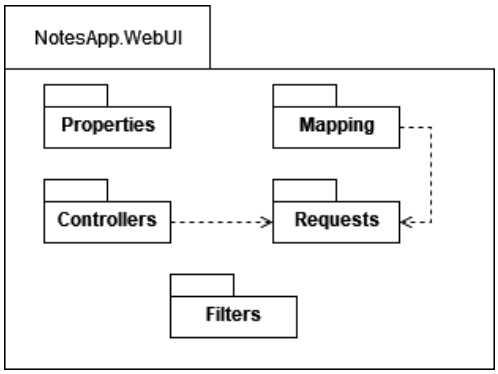


Рисунок 3.16 – Діаграма пакетів шару WebUI

Діаграма пакетів всієї системи(рис. 3.17) показує як шари взаємодіють між собою.

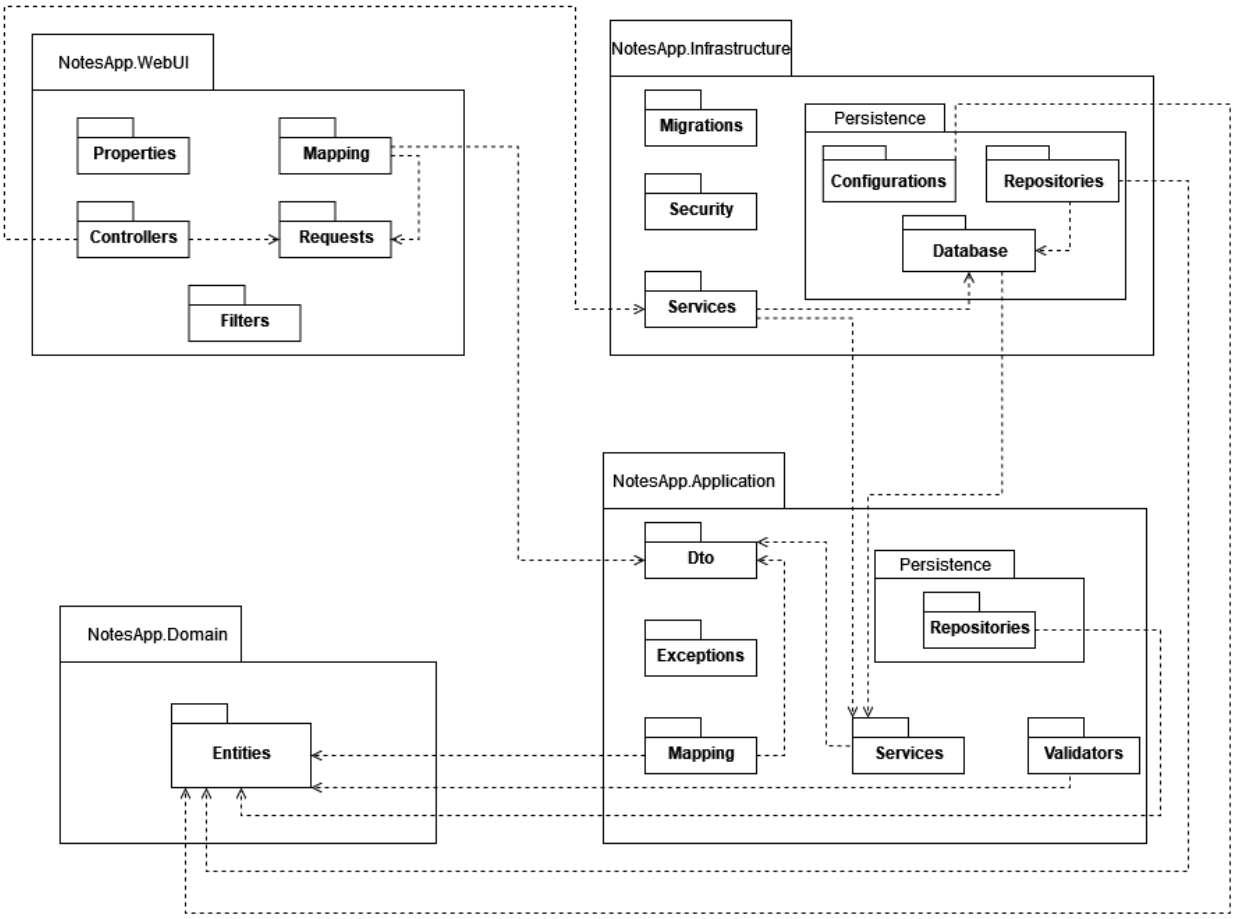


Рисунок 3.17 – Діаграма пакетів системи.

Архітектура передбачає велику кількість класів, які будуються за однаковим принципом, але містять методи та поля, які потрібні певним сутностям. На діагармі класів системи(рис. 3.18) зображено класи, які спроєктовані для сутності «User».

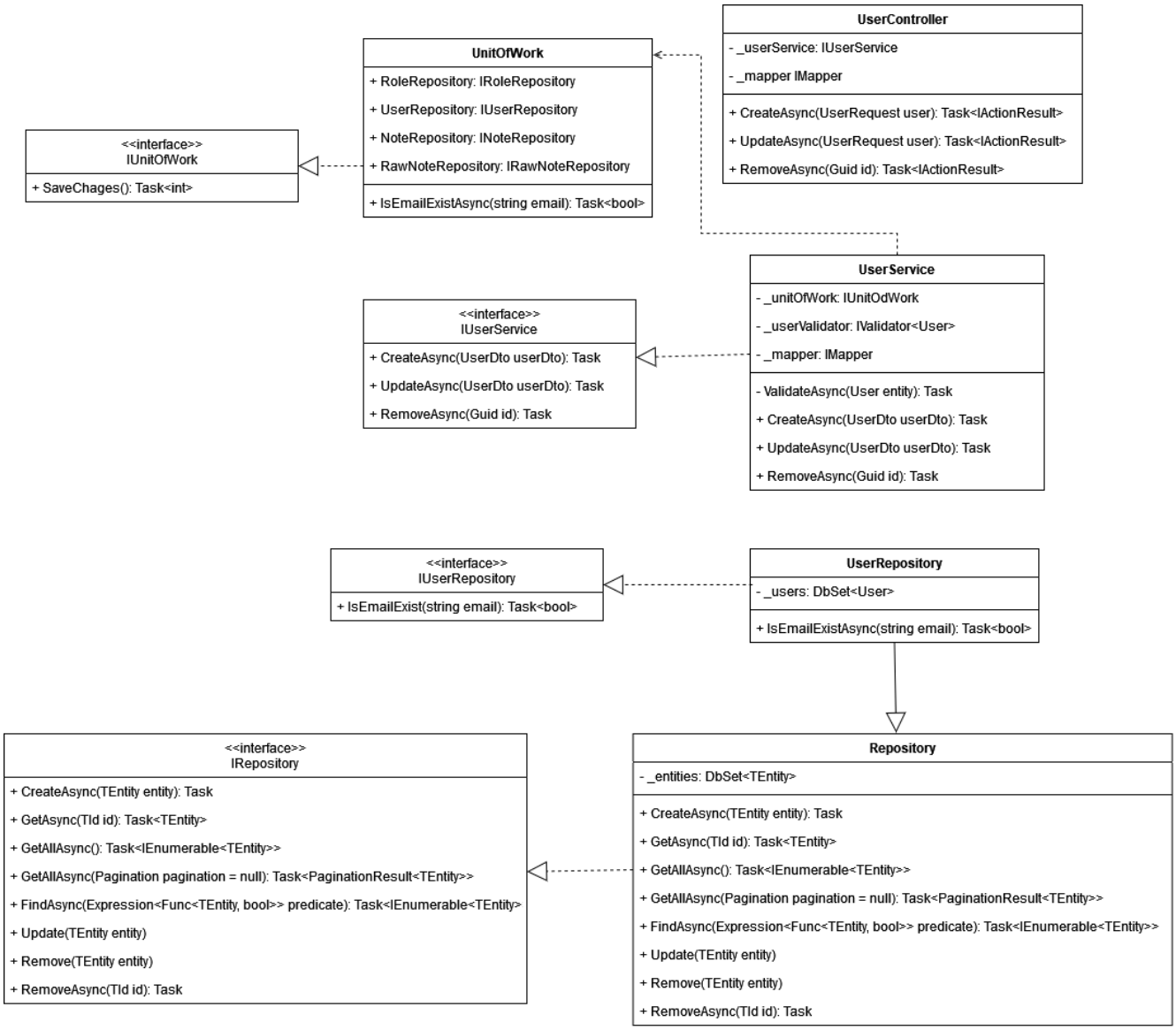


Рисунок 3.18 – Діаграма класів

3.6 Проектування інтерфейсу

Для веб-сервісу ведення нотаток необхідно спроєктувати наступні сторінки:

– Головна сторінка

– Акаунт користувача

– Сторінка реєстрації та авторизації

– Сторінка створення та редагування нотатки

На головній сторінці будуть відображатися всі публічні нотатки користувачів. Також на ній будуть присутні кнопки переходу до реєстрації або на акаунт користувача.

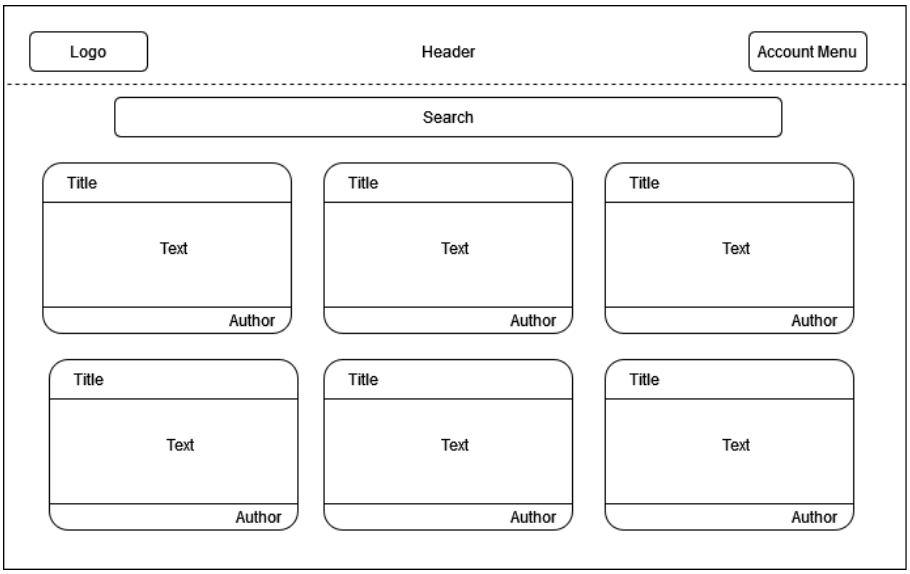


Рисунок 3.19 – Макет головної сторінки сайту

Для відображення нотаток буде використовуватися безкінечна прокрутка(infinite scroll) - це бібліотека JavaScript, яка автоматично додає наступну сторінку, позбавляючи користувачів від повного завантаження сторінки[32]. Тобто при перегляді нотаток, коли користувач буде досягати кінця сторінки буде автоматично завантажуватися наступний блок нотаток.

Кнопка «Account Menu» матиме різний вміст в залежності від того авторизований користувач чи ні. Вміст меню можна побачити на діаграмі станів меню(рис. 3.20).

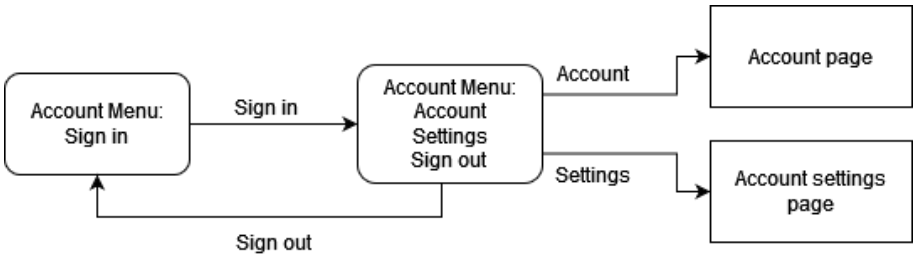


Рисунок 3.20 – Діаграма станів меню

На сторінці акаунта користувача відображається інформація користувача та всі нотатки, які йому належать. Також на сторінці повинні буди кнопки створення нової нотатки та редагування вже створених.

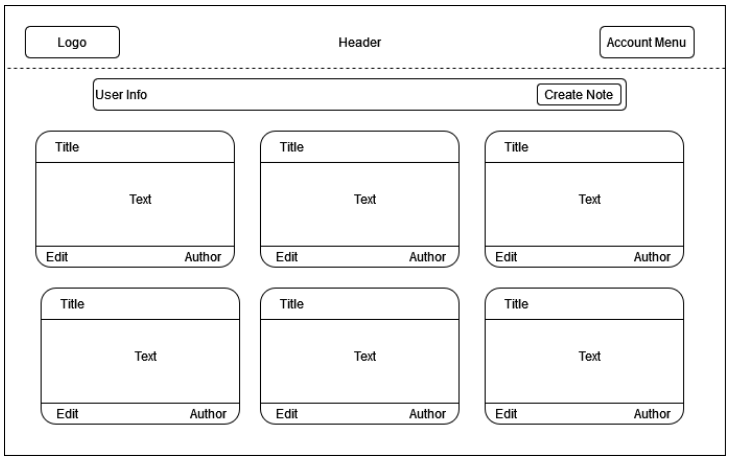


Рисунок 3.21 – Макет сторінки акаунту користувача

Для реєстрації користувача необхідно передавати на сервер наступну інформацію:

– Ім'я
– Прізвище
– Псевдонім
– Електронна пошта
– Пароль

Для передачі такої інформацій необхідно створити форму із відповідними полями.

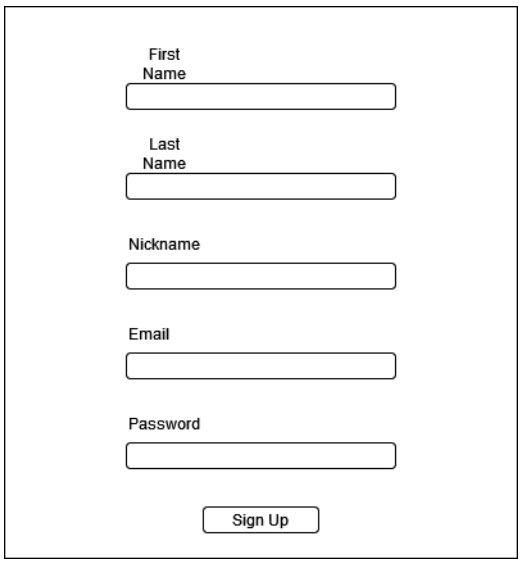


Рисунок 3.22 – Макет сторінки реєстрації користувача

На сторінці створення нотатки повинні бути присутні відповідні поля для передачі інформації на сервер для створення нотатки. Нотатку можна форматувати як завгодно. Для форматування було використано бібліотеку CKEditor 5 [33].

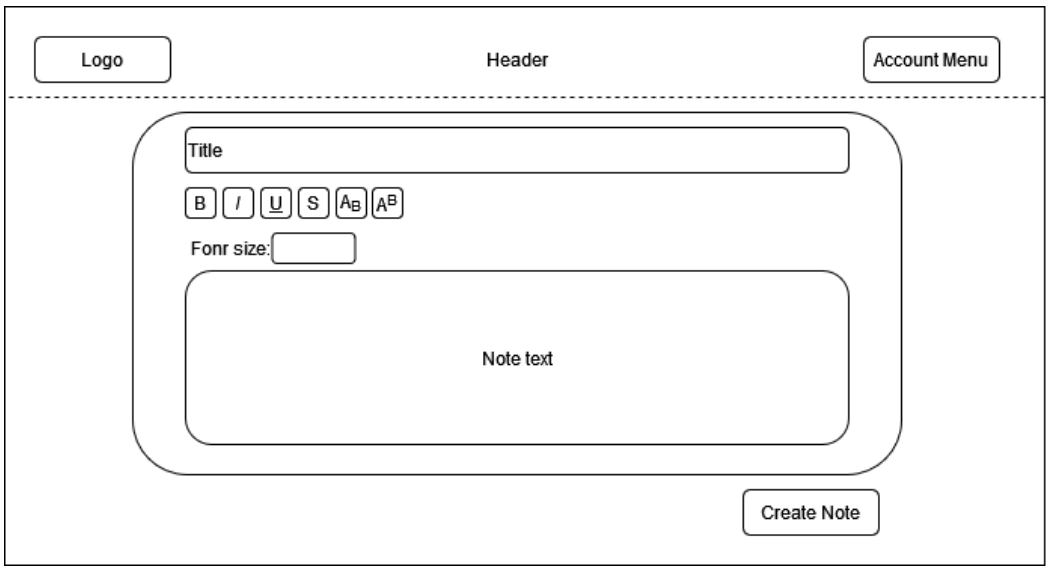


Рисунок 3.23 – Макет сторінки створення нотатки

3.7 Безпека даних

Безпека паролів здійснюється за допомогою хеш-функції bcrypt яка заснована на шифрі Blowfish. Bcrypt використовує так звану «сіль» для захисту від райдужних таблиць, крім цього, алгоритм є адаптивним, тобто час виконання легко налаштувати та ускладнити brute force атаку на хешовані дані.

Передача даних між клієнтом та серверами здійснюється у зашифрованому вигляді за допомогою асинхронного алгоритму шифрування RSA. Клієнт отримує публічні ключі, котрими, у разі потреби, користується для шифрування певної інформації. Після цього зашифровані дані відправляються на сервер де, у разі валідності, відбувається розшифровка та подальша їх обробка.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ НОТАТОК

4.1 Створення бази даних за підходом Code First.

Entity Framework увів підхід Code First. У даному підході фокус залишається на додатку і все починається із створення класів для нього, а не з розроблення бази даних, потім створюються класи, що відповідають дизайну бази даних.

4.1.1 Реалізація та конфігурація моделей бази даних

У доменному шарі створюємо моделі бази даних. Модель виглядає наступним чином(для прикладу було взято модель Note)

```
public class Note : AuditableEntity
{
    public Guid Id { get; set; }
    2 usages
    public string Title { get; set; }
    2 usages
    public string? Description { get; set; }
    2 usages
    public string Text { get; set; }

    2 usages
    public AccessType AccessType { get; set; } = AccessType.Public;
    public DateTime? LifeTime { get; set; } = DateTime.UtcNow.AddHours(24);

    2 usages
    public Guid UserId { get; set; }
    1 usage
    public User User { get; set; }
}
```

Рисунок 4.1 – Приклад моделі Note

На зображенні можна побачити набір полів. Поле Id має кожна модель і являється первинним ключем для майбутньої таблиці у бази даних. Entity Framework Core може сам автоматично встановити ключі, але для уникнення

помилкових зв’язків необхідно створити класи конфігурації, в яких також можна налаштувати обмеження для кожного поля.

```
public class NoteConfigurations : IEntityTypeConfiguration<Note>
{
    public void Configure(EntityTypeBuilder<Note> builder)
    {
        builder // EntityTypeBuilder<Note>
            .HasOne( navigationExpression: n :Note => n.User) // ReferenceNavigationBuilder<Note,User>
            .WithMany( navigationExpression: u :User => u.Notes)
            .HasForeignKey(n :Note => n.UserId);

        builder.Property(n :Note => n.Title).IsRequired().HasMaxLength(80);
        builder.Property(n :Note => n.Description).HasMaxLength(128);
        builder.Property(n :Note => n.Text).IsRequired();
        builder.Property(n :Note => n.AccessType).IsRequired();

        builder.Property(n :Note => n.Created).IsRequired();
        builder.Property(n :Note => n.CreatedBy).IsRequired().HasMaxLength(256);

        builder.Property(n :Note => n.LastModifiedBy).HasMaxLength(256);

    }
}
```

Рисунок 4.2 – Приклад класу-конфігурації моделі Note

На рисунку 4.2 зображений клас NoteConfiguration, у конструкторі якого можна побачити налаштування зв’язку типу «один до багатьох» та налаштування обмежень всіх полів: встановлення максимальної довжини та обов’язковість наявності даних.

4.1.2 Контекст бази даних

Основним класом, який керує функціональністю EF для даної моделі даних, є клас контекст бази даних: ApplicationDbContext. Цей клас створюється на основі класу Microsoft.EntityFrameworkCore.DbContext. Похідний клас від DbContext визначає, які сутності включені в модель даних.

За допомогою даного класу можна виконувати CRUD операції із базою даних. Колекції сутностей доменної області представленні у вигляді властивостей класу контексту:

```
1 usage
public DbSet<Role> Roles { get; set; }
1 usage
public DbSet<User> Users { get; set; }
1 usage
public DbSet<Note> Notes { get; set; }
1 usage
public DbSet<RawNote> RawNotes { get; set; }
```

Рисунок 4.3 – Колекція сутностей класу контексту

4.1.3 Міграції

Після описання всіх моделей та здійснення налаштувань полів та зв’язків між таблицями можна створювати фізичну базу даних за допомогою міграцій. У даному проєкті було вирішено додати до бази даних окремо всі таблиці, для кожної таблиці було створено окрему міграцію. Це було зроблено для подальшого легкого редагування та зміни структури у разі необхідності.

Для виконання міграції необхідно встановити бібліотеку Microsoft.EntityFrameworkCore.Tools. Після встановлення бібліотеки необхідно ввести відповідні команди у консолі, вказавши назву міграції та шлях. Після додавання нової міграції буде створено папку «Migrations», де і будуть зберігатися всі оновлення бази даних. На рисунку 4.4 можна побачити склад цієї папки на ресурсному сервері.

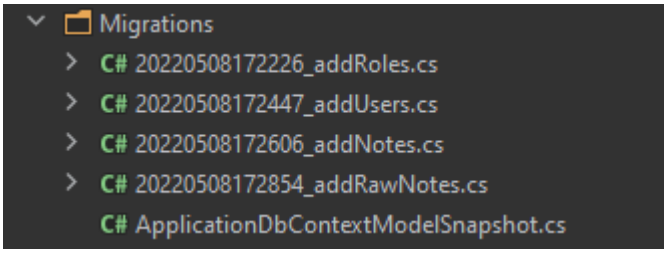


Рисунок 4.4 – Папка «Migrations»

4.2 Реалізація патерну Repository

Репозиторій - це шар абстракції, що інкапсулює в собі все, що відноситься до способу зберігання даних. Його призначення - розділ бізнес-логіки від деталей реалізації шару доступу до даних[34].

Додавання, видалення, оновлення та вибір елементів з цієї колекції здійснюється за допомогою низки простих методів, без необхідності мати справу з базами даних, і такими елементами, як зв'язки, команди, курсори або читачі.

Для використання патерну було створено інтерфейс IRepository(рис. 4.5.), де описані методи, які будуть однакові для всіх моделей – CRUD методи.

```
public interface IRepository<TEntity, TId>
{
    3 usages  1 implementation
    Task CreateAsync(TEntity entity);
    2 usages  1 implementation
    Task<TEntity> GetAsync(TId id);
    1 implementation
    Task<IEnumerable<TEntity>> GetAllAsync();
    2 usages  1 implementation
    Task<PaginationResult<TEntity>> GetAllAsync(Pagination pagination);
    1 implementation
    Task<IEnumerable<TEntity>> FindAsync(Expression<Func<TEntity, bool>> predicate);
    3 usages  1 implementation
    void Update(TEntity entity);
    1 implementation
    void Remove(TEntity entity);
    3 usages  1 implementation  ··· More
    Task RemoveAsync(TId id);
}
```

Рисунок 4.5 – Інтерфейс «IRepository»

Далі інтерфейс реалізується у абстрактному класі Repository. Для кожної моделі створюється окремий репозиторій, який успадковується від абстрактного. Приклад реалізації репозиторію користувачів можна побачити на рисунку 4.6.

```
public class UserRepository : Repository<User, Guid>, IUserRepository
{
    private readonly DbSet<User> _users;

    1 usage
    public UserRepository(ApplicationDbContext context) : base(context)
    {
        _users = context.Users;
    }

    0+1 usages
    public async Task<bool> IsEmailExistsAsync(string email)
    {
        return await _users.AnyAsync(u:User => u.Email == email);
    }
}
```

Рисунок 4.6 – Репозиторій користувача «UserRepository»

На рисунку можна побачити, що клас UserRepository успадковується від абстрактного репозиторію та власного інтерфейсу, де описані специфічні методи для користувачів.

Також клас містить приватне поле _users типу DbSet<User>. Таке поле присутнє у репозиторіях кожної моделі, але з іншою типізацією, що відповідає назві моделі. Через це поле і відбувається зв’язок із таблицями БД.

На рисунку 4.7 зображено діаграму класів, на якій наглядно видно зв’язок між класами та інтерфейсами всіх репозиторіїв.

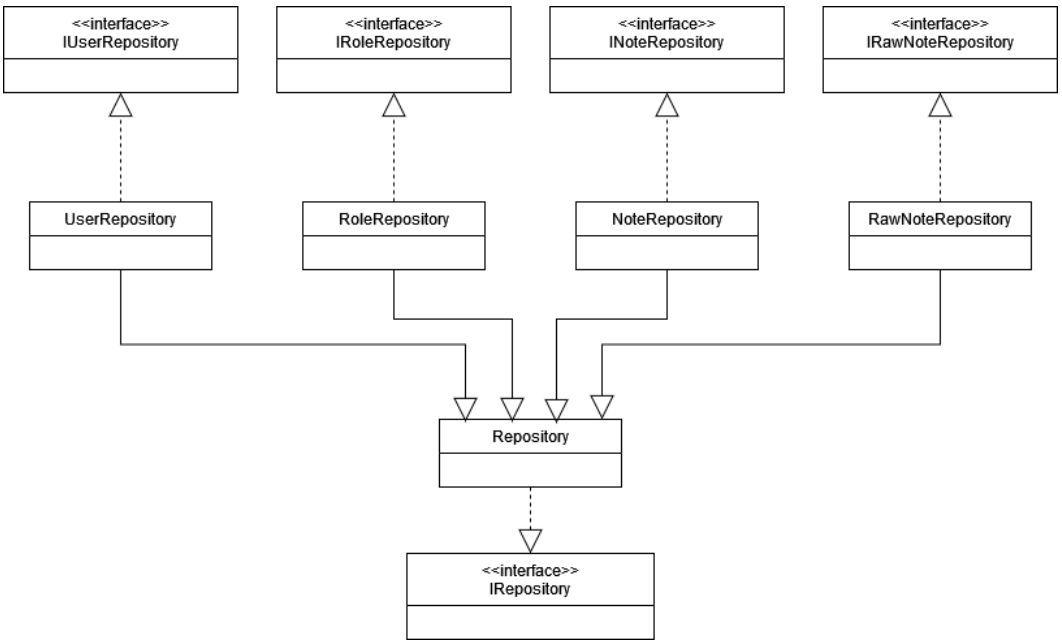


Рисунок 4.7 – Діаграма класів репозиторіїв сервісу

4.3 Релізація патерну UnitOfWork

Патерн Repository не передбачає наявність методу збереження, тому виникає необхідність у використанні патерну UnitOfWork.

UnitOfWork - це патерн, що визначає логічну транзакцію, тобто атомарну синхронізацію змін в об'єктах, поміщених в об'єкт UoW зі сховищем (базою даних)[35]. Він веде перелік об’єктів, на які впливають бізнес операції і координує зміни.

Для реалізації патерну описуємо інтерфейс IUnitOfWork(рис. 4.8), де присутні поля, через які здійснюється збереження змін та сам асинхронний метод SaveChangesAsync, який зберігає зміни.

Рисунок 4.8 – Інтерфейс IUnitOfWork

Далі було створено клас UnitOfWork, що реалізує інтерфейс IUnitOfWork(рис. 4.9).

```
public class UnitOfWork : IUnitOfWork
{
    1+1 usages
    public IRoleRepository RoleRepository { get; }
    1+6 usages
    public IUserRepository UserRepository { get; }
    1+7 usages
    public IRawNoteRepository RawNoteRepository { get; }
    1+10 usages
    public INoteRepository NoteRepository { get; }

    private readonly ApplicationDbContext _context;

    public UnitOfWork(ApplicationDbContext context)
    {
        _context = context;

        RoleRepository = new RoleRepository(context);
        UserRepository = new UserRepository(context);
        RawNoteRepository = new RawNoteRepository(context);
        NoteRepository = new NoteRepository(context);
    }

    0+9 usages
    public async Task<int> SaveChangesAsync()
    {
        return await _context.SaveChangesAsync();
    }
}
```

Рисунок 4.9 – Клас UnitOfWork

4.4 Реалізація сервісів

Обраною архітектурою передбачене реалізація сервісів для кожної моделі. Через сервіси буде здійснюватися зв’язок із репозиторіями.

Сервіси не будуть опрацьовувати сутності моделі, для обробки будуть використовуватися DTO-сутності. DTO – це об’єкт який переносить дані з одного шару програми в інший. У ньому можна скоротити кількість полів в залежності від необхідності наявності певних полів. Отже, для реалізації сервісів необхідно описати класи DTO для кожної моделі(рис. 4.10).

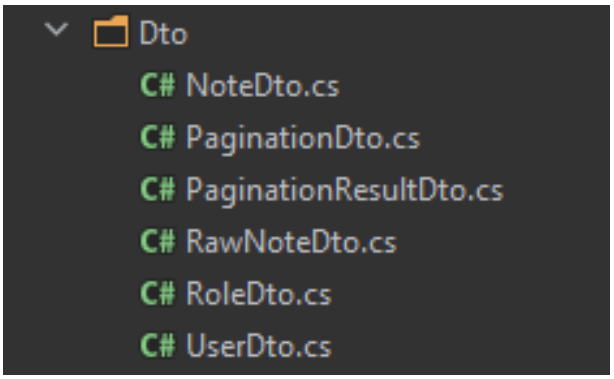


Рисунок 4.10 – Папка із класами DTO

Сервіси повинні реалізовувати такі ж самі методи як репозиторії. Методи повинні передбачати валідацію даних, де це є необхідним. Також у методах виконується перетворення за допомогою AutoMapper, після чого викликається відповідний метод із репозиторію, куди передається перетворена сутність.

```
public class UserService : IUserService
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IValidator<User> _userValidator;
    private readonly IMapper _mapper;
    public UserService(IUnitOfWork unitOfWork, IValidator<User> userValidator, IMapper mapper)
    {
        _unitOfWork = unitOfWork;
        _userValidator = userValidator;
        _mapper = mapper;
    }
    2 usages
    private async Task ValidateAsync(User entity)
    {
        var result = await _userValidator.ValidateAsync(entity);
        if (!result.IsValid)
            throw new ValidationException(result.Errors);
    }
    0+1 usages
    public async Task CreateAsync(UserDto userDto)
    {
        if (await _unitOfWork.UserRepository.IsEmailExistsAsync(userDto.Email))
            throw new ValidationException(new ValidationFailure(nameof(userDto.Email), errorMessage: $"Email '{userDto.Email}' already used."));
        var entity = _mapper.Map<User>(userDto);
        await ValidateAsync(entity);
        await _unitOfWork.UserRepository.CreateAsync(entity);
        await _unitOfWork.SaveChangesAsync();
    }
    0+1 usages
    public async Task UpdateAsync(UserDto userDto)
    {
        var entity = _mapper.Map<User>(userDto);
        await ValidateAsync(entity);
        _unitOfWork.UserRepository.Update(entity);
        await _unitOfWork.SaveChangesAsync();
    }
    public async Task RemoveAsync(Guid id)
    {
        await _unitOfWork.UserRepository.RemoveAsync(id);
        await _unitOfWork.SaveChangesAsync();
    }
}
```

Рисунок 4.11 – Реалізація сервісу UserService

4.4.1 Валідація даних

Кожен сервіс містить поле валідатор, що використовується у методі ValidateAsync. Для кожної моделі створюється клас-валідатор, що повинен успадковувати клас AbstractValidator. Цей клас відноситься до бібліотеки FluentValidation, яку необхідно установити за допомогою NuGet менеджеру.

FluentValidation — це бібліотека .NET для створення строго типізованих правил перевірки[36].

На рисунку 4.12 зображено приклад валідатору для моделі користувачів.

```
public class UserValidator : AbstractValidator<User>
{
    public UserValidator()
    {
        RuleFor( expression: u :User => u.FirstName).NotEmpty().Length(3, 48);
        RuleFor( expression: u :User => u.LastName).NotEmpty().Length(3, 48);
        RuleFor( expression: u :User => u.Nickname).NotEmpty().Length(3, 32);
        RuleFor( expression: u :User => u.Email).NotEmpty().Length(10, 128); //match
    }
}
```

Рисунок 4.12 – Клас UserValidator

4.4.2 Automapper

Для обробки DTO та передачі даних до репозиторіїв необхідно перетворити DTO об’єкт у об’єкт типу сутності. Для цього необхідно встановити бібліотеку AutoMapper за допомогою NuGet Package Manager.

Для налаштування перетворень для кожного виду сутностей створюється профіль. Приклад такого профілю можна побачити на рисунку 4.13.

```
public class UserProfile : Profile
{
    public UserProfile()
    {
        CreateMap<User, UserDto>().ReverseMap();
    }
}
```

Рисунок 4.13 – Приклад налаштувань перетворень

У кожному профілі викликається метод CreateMap та вказується із якого типу в який необхідно перетворити об’єкт(у прикладі із User в UserDto). У даному випадку всі поля в User та UserDto мають однакову назву, тому не потребують додаткових налаштувань, AutoMapper сам розуміє що вони відповідають один одному. Але у разі, якщо назви полів різні, необхідно указати у профілі які поля відповідають один одному. Метод ReverseMap означає налаштування перетворення в обидві сторони.

Також слід зазначити, що перетворення необхідно налаштувати не лише із звичайних моделей у DTO-моделі(та навпаки), а і із DTO-моделей в Request-модель(та навпаки). Request-моделі – це такі ж моделі, але вони містять лише ту інформацію, яка необхідна користувачу та буде відображатися на інтерфейсі.

Перетворення моделей зображено на діаграмі класів на рисунку 4.14.

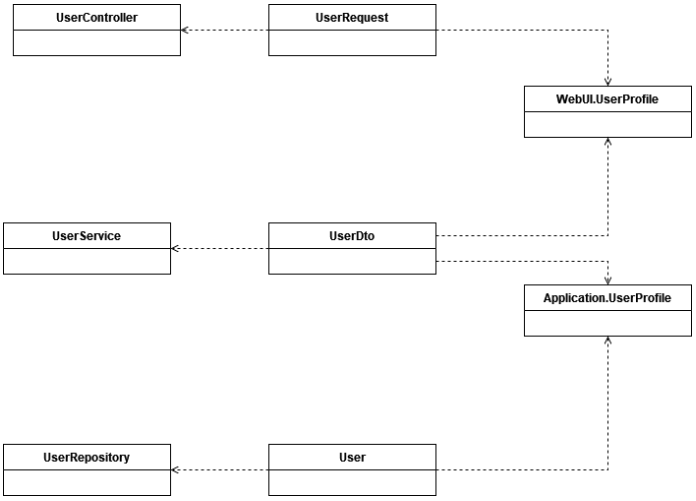


Рисунок 4.14 – Діаграма класів перетворень

4.5 Реалізація контролерів

Проєкт типу Веб-Апі на основі контролера складається із одного або декількох класів контролерів, що успадковані від класу ControllerBase. Цей клас містить велику кількість полів та методів, які зручні для обробки http-запитів.

Контролери створюються для того, щоб зробити сервіс доступним для користувачів. При створенні кожного класу контролера необхідно прописати конструктор, де додати відповідні сервіси(рис. 4.15)

```
[Route( template: "api/v1/roles")]
public class RoleController : ControllerBase
{
    private readonly IMapper _mapper;
    private readonly IRoleService _roleService;

    public RoleController(IMapper mapper,IRoleService roleService)
    {
        _mapper = mapper;
        _roleService = roleService;
    }
```

Рисунок 4.15 – Клас RoleController

На рисунку можна побачити поля та конструктор контролеру ролей користувача. Для його роботи створено поле сервісу ролей, методи якого використовуються у методах контролерів.

Деякі методи контролерів дозволено викликати тільки, якщо користувач авторизований. Для такого розмежування необхідно встановлювати атрибут Autorize, наприклад на рисунку 4.16 можна побачити метод Update, який доступний лише авторизованим користувачам, у яких роль – Admin.

```
[HttpPut( template: "{id}")]
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Update([FromRoute] string id, [FromBody] string role)
{
    await _userService.UpdateRole(id, role);

    return Ok();
}
```

Рисунок 4.16 – Метод контролера із атрибутом Autorize

4.6 Реалізація авторизації та аутентифікації користувачів

Реєстрація та аутентифікація користувачів відбувається на аутентифікаційному сервері, який побудований за такими ж архітектурними рішеннями як і ресурсний. Основна логіка серверу реалізована у сервісах.

На сервері присутні наступні функції у контролері:

– Реєстрація(SignUp)

– Аутентифікація(SignIn)

– Оновлення токена доступу(Refresh)

Більш детально розглянемо процес реєстрації. Коли користувач натискає кнопку «Sign up» формується запит на отримання публічного ключа, за допомогою якого шифрується пароль, та http-запит із введеними даними користувача по заданому маршруту, що відповідає реєстрації на аутентифікаційному сервері.

Далі дані передають до аутентифікаційного сервісу у метод реєстрації(рис.4.17).

```
public async Task<AuthResultDto> SignUp(SignUpRequestDto requestDto)
{
    requestDto.Password = Decrypt(requestDto.Password);
    var userDto = await _userService.CreateAsync( userDto: _mapper.Map<UserDto>(requestDto));

    return await GenerateTokensAsync(userDto);
}
```

Рисунок 4.17 – Метод реєстрації аутентифікаційного сервісу.

Спочатку здійснюється розшифровка паролю за допомогою методу Decrypt, який приймає пароль у вигляді Base64 рядку. Якщо пароль передається у невірному форматі , то метод повертає помилку «Incorrect password type».

Після розшифрування паролю викликається метод сервісу користувача(UserService)(рис.4.18), де спочатку валідуються дані, потім у разі

коректності даних хешується пароль і в результаті дані зберігаються та повертається об’єкт нового користувача.

```
public async Task<UserDto> CreateAsync(UserDto userDto)
{
    var entity = _mapper.Map<User>(userDto);
    await ValidateAsync(entity);

    entity.Password = _passwordHashService.HashPassword(entity.Password);

    await _unitOfWork.UserRepository.CreateAsync(entity);
    await _unitOfWork.SaveChangesAsync();

    return _mapper.Map<UserDto>(entity);
}
```

Рисунок 4.18 – Створення нового користувача

Після отримання об’єкту нового користувача генерується пара Access Token та Refresh Token з певними даними користувача. В кінці ця пара повертається на клієнтську частину.

При авторизації спочатку формується запит на ресурсний сервер, де перевіряється валідність токену та у разі його коректності користувачу надається певна роль, що встановлена на ресурсному сервері.

ВИСНОВКИ

У межах даної бакалаврської роботи проведено розробку веб-сервісу ведення нотаток з використанням мови програмування C#.

Актуальність даного проекту залежить в тому, що у наш час людей оточує велика кількість даних та фактів, все більше людей працюють та навчаються онлайн. Для того, щоб не забувати певну інформацію або донести її іншим людям, необхідний універсальний засіб для структурування та ефективної передачі даних, який буде доступний на будь-якому пристрої.

У роботі вирішено такі завдання:

– Проаналізовано та досліджено існуючі аналоги ведення нотаток, здійснено їх порівняльну характеристику.
– Досліджено програмні засоби створення веб-сервісів.
– Розроблено структуру веб-сервісу ведення нотаток.
– Реалізовано веб-сервіс ведення нотаток за допомогою мови програмування C#.

Робота пройшла апробацію на Науково-технічній конференції «Застосування програмного забезпечення в інфокомунікаційних технологіях». За результатами апробації опубліковано тези доповідей: «Криптографічні алгоритми у розробці програмного забезпечення»

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт веб-сервісу ведення нотаток «OneNote». URL: https://www.microsoft.com/uk-ua/microsoft-365/onenote/digital-note-taking-app
2. Офіційний сайт веб-сервісу ведення нотаток «Evernote». URL: https://evernote.com/intl/en
3. Список можливостей, які надають підписки сервісу «Evernote». URL: https://evernote.com/intl/en/compare-plans
4. Офіційний сайт веб-сервісу ведення нотаток «Keep». URL: https://keep.google.com/
5. What is .NET? Introduction and overview. URL: https://docs.microsoft.com/en-us/dotnet/core/introduction
6. Офіційна документація мови програмування C#. URL: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
7. Офіційна документація мови програмування F#. URL: https://docs.microsoft.com/en-us/dotnet/fsharp/what-is-fsharp
8. Офіційна документація мови програмування Visual Basic. URL: https://docs.microsoft.com/en-us/dotnet/visual-basic/whats-new/
9. Офіційна документація бібліотеки Vue.js. URL: https://vuejs.org/guide/introduction.html#what-is-vue
10. Офіційна документація бібліотеки React.js. URL: https://ru.reactjs.org/
11. Офіційна документація бібліотеки Angular. URL:https://angular.io/
12. Офіційний сайт середовища розробки Rider. URL: https://www.jetbrains.com/ru-ru/rider/
13. Система керування версіями. URL: https://uk.wikipedia.org/wiki/Система_керування_версіями
14. Git. URL: https://ru.wikipedia.org/wiki/Git
15. Документація СКВ Subversion. URL: https://tortoisesvn.net/
16. СКВ Mercurial. URL: https://ru.wikipedia.org/wiki/Mercurial
17. Бази даних. URL: https://uk.wikipedia.org/wiki/ База_даних

18. Реляційні бази даних. URL: https://uk.wikipedia.org/wiki/Реляційна_база_даних

19. Офіційна документація MySQL. URL: https://www.mysql.com/

20. Офіційний сайт PostgreSQL. URL: https://www.postgresql.org/

21. Офіційна документація MSSQL Server. URL: https://www.microsoft.com/en-us/sql-server/sql-server-2019

22. Симетричні криптосистеми. URL: https://ru.wikipedia.org/wiki/Симметричные_криптосистемы

23. Криптографічні алгоритми. Класифікація з точки зору кількості ключів. URL: https://habr.com/ru/post/336578/

24. RSA . URL: https://uk.wikipedia.org/wiki/RSA

25. Архітектура клієнт-сервер. URL: https://uk.wikipedia.org/wiki/ Клієнт-серверна_архітектура

26. UML діаграми. URL: https://evergreens.com.ua/ru/articles/uml-diagrams.html

27. Офіційна документація по проектуванню баз даних. URL: https://support.microsoft.com/uk-ua/office/ основи-розробки-баз-даних-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5

28. Основні поняття реляційних БД: нормалізація, зв’язок та ключі. URL: https://bondarenko.dn.ua/osnovni-ponyattya-relyatsijnih-bd-normalizatsiya-zv-yazok-ta-klyuchi/

29. Нормалізація. URL: https://uk.wikipedia.org/wiki/Нормалізація

30. Clean Architecture. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

31. Robert Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design - 1st Edition. – 431 ст.

32. Бібліотека Infinite Scroll. URL: https://infinite-scroll.com/

33. Бібліотека CKEditor 5. URL: https://ckeditor.com/ckeditor-5/

34. Патерн Repository. URL: https://deviq.com/design-patterns/repository-pattern

35. Патерн Unit of work. URL: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

36. Офіційна документація Fluent Validation. URL: https://docs.fluentvalidation.net/en/latest/

ДОДАТОК А Презентація

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ВЕБ- СЕРВІСУ ВЕДЕННЯ НОТАТОК З ОБЛАДНАННЯ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#

Виконавець: студент 4 курсу
групиПД-43 ЦатурянЕдуард Русланович
Керівник роботи
старший викладач Гаманюк Ігор Михайлович

Київ – 2022

АНАЛОГИ

Назва додатку	Основні переваги	Основні недоліки
OneNote	Можливість розбити нотатку на розділи Можливість працювати із зображеннями та документами Можливість встановлювати пароль на нотатку	Незрозумілий та перевантажений інтерфейс Синхронізаіція сервіс не завжди правильно завантажує необхідні дані.
Evernote	Встановлення нагадувань Можливість працювати із зображеннями та документами	Безкоштовна версія занадто обмежена Відсутність публічного доступу до нотаток
Google Keep	Пошук нотаток по тегам. Встановлення нагадувань Виділення нотаток кольорами.	Відсутня можливість надавати доступ до нотатки іншим користувачам Невеликий набір стилів форматування тексту нотатки

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення ведення нотаток за допомогою онлайн-засобів.

Об’єкт дослідження: ведення нотаток за допомогою онлайн засобів.

Предмет дослідження: веб-сервіс для ведення нотаток.

ТЕХНІЧНІ ЗАВДАННЯ

1. Реалізувати розмежування доступу.
2. Реалізувати створення та управління нотатками.
3. Реалізувати можливість встановлення доступу до нотатки.
4. Реалізувати можливість перегляду акаунтів та нотаток користувачів.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Платформа Net Core

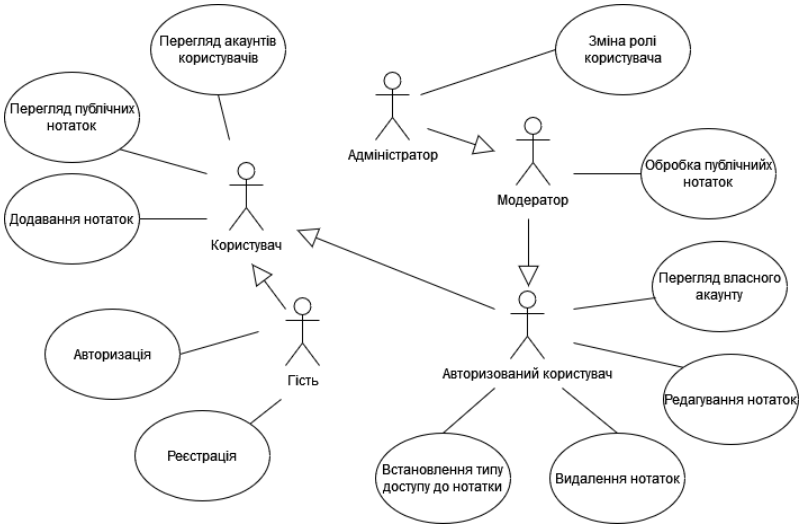
Мова програмування C#

ФреймворкAngular

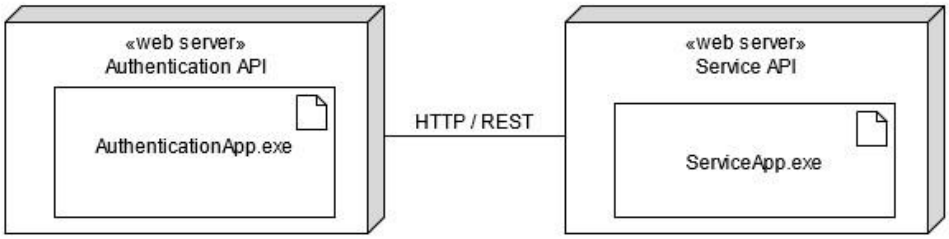
Система керування версіямиGit

СКБД MSSQL Server

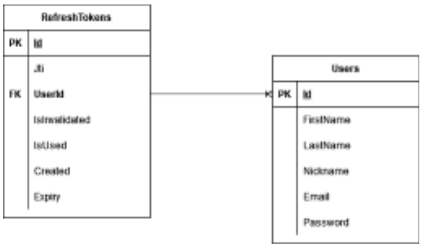
ДІАГРАМА ПРЕЦЕДЕНТІВ



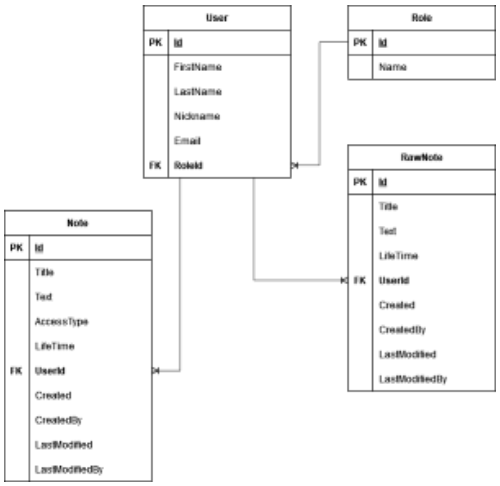
ДІАГРАМА РОЗГОРТАННЯ



СХЕМИ БАЗ ДАНИХ

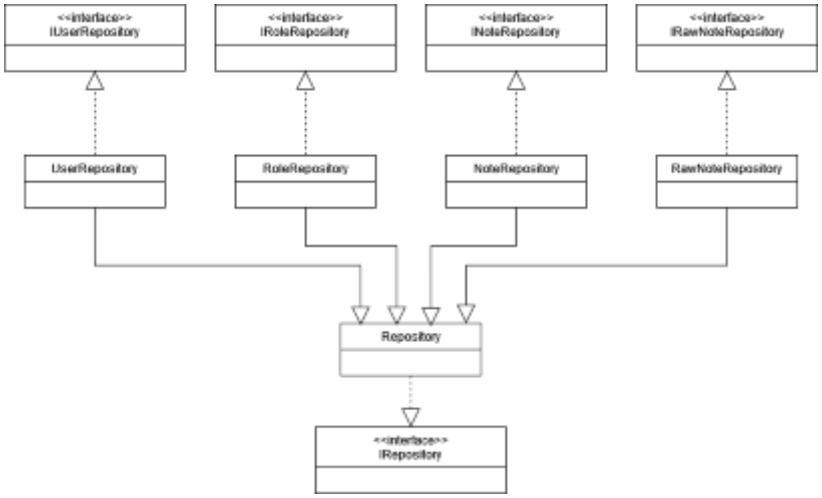


Аутентифікаційний сервер



Ресурсний сервер

ДІАГРАМА КЛАСІВ РЕПОЗИТОРІЇВ СЕРВІСУ



ЕКРАННІ ФОРМИ

Головна сторінка сервісу

ЕКРАННІ ФОРМИ

Сторінка створення нотатки

ЕКРАННІ ФОРМИ

Форма входу до аккаунту

Форма реєстрації нового користувача

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Цатурян Е.Р. Криптографічні алгоритми у розробці програмного забезпечення / Е.Р. Цатурян, Д.С. Коваленко // Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез. 22.04.2022, ДУТ, м. Київ — К.: ДУТ, 2022 — С. 53 - 54.

ВИСНОВКИ

1. Проаналізовано та досліджено існуючі аналоги.
2. Досліджено програмні засоби для створення веб-сервісів.
3. Визначено функціональні можливості.
4. Опрацьовано концептуальну модель бази даних.
5. Спроектовано архітектуру застосунку.
6. Реалізовано ресурсний та аутентифікаційний сервери та клієнтську частину веб-сервісу нотаток

ДЯКУЮ ЗА УВАГУ!