Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Фесака Дмитра Володимировича*
(ПІБ)

академічної групи *121-18-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка кросплатформеного клієнт-серверного додатка для нотаток*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Приходченко С.Д.*			
розділів:				
спеціальний	*доц. Приходченко С.Д.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18-1* (група) *Фесака Д.В.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка кросплатформеного клієнт-серверного додатка для нотаток*

затверджена наказом ректора НТУ «ДП» від *18 травня 2022р.* № *268-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Фесак Д.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 85 с., 26 рис., 3 дод., 21 джерело.

Об'єкт розробки: кросплатформений клієнт-серверний додаток для нотаток.

Мета кваліфікаційної роботи: створення кросплатформеного додатка, який дозволятиме керувати синхронізованими нотатками, на різних пристроях.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні додатка, що дозволяє створювати, змінювати та видаляти нотатки на різних пристроях.

Актуальність даного програмного продукту визначається великим попитом на подібні розробки, що дозволяють ефективно працювати з користувальницькою інформацією на різних пристроях без втрати прогресу.

Список ключових слів: ДОДАТОК, ANDROID, IOS, WINDOWS, СМАРТФОН, КОМП’ЮТЕР, КЛІЄНТ, СЕРВЕР, ПРОЕКТУВАННЯ, АРХІТЕКТУРА, КРОСПЛАТФОРМЕНІСТЬ, ВЕБ.

ABSTRACT

Explanatory note: 85 p., 26 figs., 3 appx., 21 sources.

Object of development: cross-platform client-server application for notes.

Purpose of the qualification work: to create a cross-platform application that allows you to manage synchronized notes on different devices.

In the introduction it is considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the scope of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the subject branch is analyzed, the urgency of the task and purpose of development are defined, the statement of the task is formulated, requirements to software realization, technologies and software are specified.

The second section analyzes the existing solutions, selected platforms for development, designed and developed the program, describes the work of the program, algorithm and structure of its operation, as well as calling and loading the program, determines the input and output data, describes the parameters of technical means.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical value is to create an application that allows you to create, update and delete notes on different devices.

The relevance of this software product is determined by the high demand for such developments that allow you to work effectively with user information on different devices without losing progress.

List of keywords: APPLICATION, ANDROID, IOS, WINDOWS, SMARTPHONE, COMPUTER, CLIENT, SERVER, DESIGN, ARCHITECTURE, CROSS-PLATFORM, WEB.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

БД – база даних.

ОС – операційна система.

ЕОМ – електронна обчислювана машина.

CLR – common language runtime.

DTO – data transfer object.

ORM – object-relational mapping.

MVVM – Model - View - ViewModel.

UI – user interface.

UWP – universal windows platform.

JWT – JSON web token.

ВСТУП

Темою даної кваліфікаційної роботи є розробка кросплатформеного клієнт-серверного додатка для нотаток.

Метою даної кваліфікаційної роботи є вивчення інструментальних засобів Visual Studio, платформи .Net Core, набуття навичок проектування клієнт-серверної архітектури за протоколом REST та тісна взаємодія з системою управління базами даних MySql. Ці знання дозволять будувати ефективні і якісні програмні продукти, для яких важливі такі риси як масштабованість, кросплатформеність, відмовостійкість та прозорість архітектури.

В реаліях сьогодення, людина постійно отримує величезну кількість інформації з різних джерел. Звичайно, це дозволяє їй бути продуктивнішою, більш обізнаною та ефективною, але, в одночас, така кількість інформації ускладнює її засвоєння та раціональне використання. Цю проблему люди звикли вирішувати веденням щоденника або записами найважливішого у блокнот, але зараз різносторонньої інформації буває настільки багато, що записувати все на папері досить проблематично, та й блокнота під рукою може і не бути. Але є річ, яку людина зазвичай точно має при собі – смартфон. Тож мати замітки у смартфоні це вже дійсно необхідно, а спеціальний, зручний додаток, який був розроблений саме з цією метою стає на користь.

Окрім цього, бувають і такі ситуації, коли через втрату або заміну телефона втрачається доступ до програм, які використовувались на минулому пристрої. Дуже прикро втрачати всю інформацію, яка ретельно нотувалася протягом місяців, або навіть років. Саме для того, щоб уникнути таких ситуацій і потрібно, щоб додаток був кросплатформеним та у разі необхідності дозволяв продовжити роботу над нотатками на смартфоні з іншою ОС, або навіть на ПК без втрати прогресу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Клієнт-серверна архітектура – це обчислювана модель, в якій сервер керує більшою частиною ресурсів та послуг, які споживає клієнт. У цьому типі архітектури клієнтські пристрої підключаються до центрального сервера через інтернет. Сервер розміщує та надає клієнту високоякісні послуги з інтенсивними обчисленнями за запитом. Ці послуги можуть включати доступ до програм, зберігання, спільний доступ до файлів, або прямий доступ до обчислювальної потужності сервера.

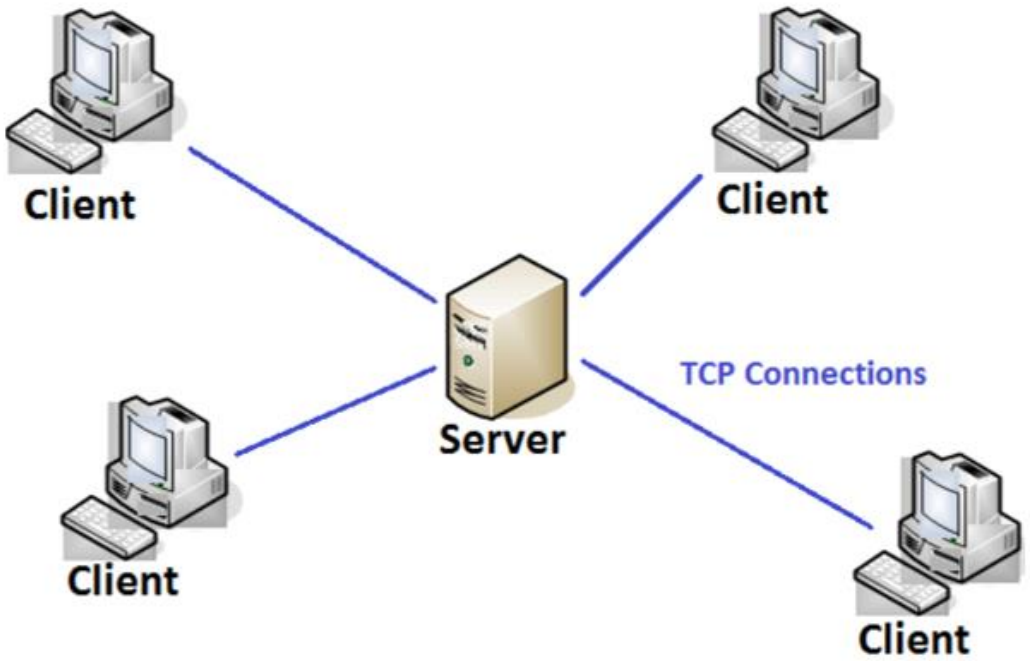

Рис.1.1. Зображення клієнт-серверної архітектури

Під час свого розвитку модель клієнт/сервер досить добре служила для того, що дехто називає веб 2.0, де Інтернет повільно став функціональним віртуальним простором для користувачів. Він надав усталену та передбачувану модель того, як проходитимуть сеанси користувачів і як постачальники доставляли ресурси на основі запитів на пакети даних та інші ресурси.

Характеристики клієнт-серверної архітектури:

– Клієнтським і серверним ЕОМ необхідна різна кількість апаратних і програмних ресурсів

– Горизонтальна масштабованість (збільшення кількості клієнтських машин).

– Вертикальна масштабованість (міграція на більш потужний сервер або багатосерверне рішення).

– Клієнтський або серверний додаток безпосередньо взаємодіє з протоколом транспортного рівня, щоб встановити зв'язок та надіслати чи отримати інформацію.

– Один комп'ютер серверного класу може надавати кілька послуг одночасно, проте для кожної служби потрібна окрема серверна програма.

Зазвичай, клієнт-серверна архітектура має три рівні:

– Рівень презентації, тобто клієнт, з яким взаємодіє користувач.
– Рівень бізнес логіки – програмне забезпечення, яке запускається на сервері та містить бізнес-логіку.
– Рівень даних – менеджер ресурсів, який зберігає дані.

Така архітектура має ряд переваг, таких як керований та безпечний обмін даними, масштабованість, спільні ресурси для різних платформ, простота обслуговування та інкапсуляція бізнес-логіки.

Класичним протоколом для передачі даних у клієнт-серверній моделі є протокол HTTP. Клієнт надсилає запит (request) до сервера, сервер обробляє запит, виконує необхідні маніпуляції з даними та надсилає клієнту відповідь (response). HTTP є протоколом без стану, тобто сервер не має ніякого зв'язку між двома різними запитами, які послідовно виконуються в одному з'єднанні.

HTTP request складається з наступних елементів:

1. Метод – вказує бажану дію для даного ресурсу, наприклад GET.
2. Шлях до ресурсу – URL-адреса.
3. Опціональні заголовки, які несуть додаткову інформацію для сервера
4. Тіло запиту, для деяких методів, наприклад POST.

В свою ж чергу, HTTP response складається з елементів:

1. Код статусу – вказує був запит успішним, чи ні, і чому.
2. Повідомлення про статус – короткий опис коду статусу.
3. Заголовки, як і у запита.
4. Опціональне тіло, яке містить отриманий ресурс.

Звичайно, при розмові про клієнт-серверну архітектуру в поєднанні з HTTP, неможливо не сказати про RESTful API. Це архітектурний стиль, який має свої принципи та обмеження, яким API має слідувати для того, щоб називатися RESTful. Важливо розуміти, що API – це інтерфейс, який дозволяє клієнтському додатку обмінюватися інформацією з серверним, при цьому як саме реалізовувати такий інтерфейс кожний розробник може вирішувати сам, але, звичайно, існують так звані найкращі практики, які призначені для того, щоб уніфікувати досвід багатьох розробників заради створення консистентного, розширюваного та зрозумілого програмного забезпечення.

Основними принципами REST є:

1. Уніфікований інтерфейс, а саме для його досягнення вводяться такі обмеження як однозначна ідентифікація кожного ресурса, який приймає участь у взаємодії між клієнтом і сервером; маніпулювання ресурсами за допомогою представлень, які часто називають DTO. Представлення для однакових запитів не можуть змінюватися, повинні нести в собі достатньо інформації для опису того, як їх потрібно обробляти; ну і на останок, клієнт повинен мати лише початкову адресу програми, а клієнтська програма керуватиме всіма іншими взаємодіями та ресурсами за допомогою гіперпосилань.

2. Звичайно ж, клієнт-серверний шаблон проєктування, який забезпечує розділення відповідальності, відокремлюючи проблеми клієнта від проблем зберігання даних і покращуючи переносимість на різні платформи та масштабованість у цілому.

3. Відсутність стану – кожен запит має містити всю інформацію, яка необхідна серверу для його виконання. Ніяка контекстна інформація про

попередній запит, чи клієнта не може бути збережена та використана для обробки іншого запита. Тому саме клієнтське ПЗ повинно зберігати стан сеансу.

4. Обмеження кешування, яке вимагає, щоб у відповіді сервер позначав чи може вона бути кешована, а її дані перевикористані пізніше для еквівалентних запитів.

5. Стиль багаторівневої системи – дозволяє системі складатися з ієрархічних шарів, обмежуючих поведінку компонентів. Кожен окремий шар не може дістатися інших шарів, окрім тих, з якими він безпосередньо взаємодіє.

6. Ну і на останок, REST також дозволяє розширити функціональність клієнта шляхом завантаження та виконання коду у вигляді скриптів.

Інша важлива складова RESTful архітектури це HTTP методи. Є такі загальноприйняті методи, як GET, POST, PUT, DELETE і зазвичай вони використовуються для того, щоб отримати, створити, змінити, та видалити інформацію про певний об’єкт відповідно, але якщо розробник вирішить змінити призначення якогось методу, це також не буде суперечити з REST. Головне, щоб призначення методів були консистентними для всієї програми і мали чіткий опис сценарію свого використання. Так само і клієнт повинен мати повне уявлення, якими методами необхідно скористатися щоб отримати передбачувану відповідь від сервера.

В результаті ми отримуємо просте, неперевантажене та швидке серверне програмне забезпечення, яке не має ніяких обмежень, пов’язаних з платформою, або користувальницьким інтерфейсом клієнтського застосунку.

1.2. Призначення розробки та галузь застосування

Розробка кросплатформеного клієнт-серверного додатка для нотаток.

Для розробки серверного програмного забезпечення було вирішено обрати платформу .NET Core. Це кросплатформений, високопродуктивний фреймворк з відкритим вихідним кодом від компанії Microsoft. Програма, розроблена на .NET Core може запускатися на ОС Windows, MacOS, або Linux. .NET Core побудований

на основі технології CLR, що означає, що його можна використовувати з будь-якою мовою .NET уніфікованим способом. До списку таких мов відносяться C#, F# та Visual Basic. На платформі .NET Core можна розробляти та запускати різні типи додатків: мобільні, десктопні, веб, хмарні, IoT, машинне навчання, мікросервіси, ігри тощо. Основна мета .NET Core – створення єдиного фреймворку, який працює на всіх платформах, але при цьому залишається консистентним, тобто використовується спільна мова програмування, загальні кодові конвенції та практики.

В ході роботи я неодмінно розширю свох знання платформи та покращу практичні навички її використання, що стане чудовим підгрунтям для подальшого професійного росту. Розроблений додаток дозволить користувачам забути про турботи, пов’язані з тим, що вони записали щось важливе десь в телефоні і не знають де тепер це знайти. Кросплатформений застосунок надасть можливість швидко переключатися між різними пристроями зі збереженням всіх заміток без будь-яких проблем.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином підставами для розробки (виконанням кваліфікаційної роботи) є:

– освітня програма спеціальності 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022 р;

– завдання на кваліфікаційну роботу на тему «Розробка кросплатформеного клієнт-серверного додатка для нотаток».

1.4. Постановка завдання

Завданням даної роботи є розробка серверного додатка, у вигляді RESTful Web API та кросплатформеного клієнта.

В результаті, сервер повинен мати наступний функціонал:

– Слідувати стилю REST;

– Реєстрація нового користувача;

– Авторизація за допомогою JWT токену;

– Можливість створення, зберігання, редагування нотаток;

– Реалізація смітника для нотаток з можливістю їх відновлення або остаточного видалення.

Клієнт, в свою чергу має реалізовувати функціонал, який пропонує сервер для роботи на платформах Android, iOS та Windows.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

– інтуїтивно зрозумілий інтерфейс користувача;

– RESTful архітектура

– ACID транзакції бази даних

– дані повинні вводитися користувачем на екрані;

– дані повинні виводитися користувачу на екран;

– дані можуть зберігатися, змінюватися і видалятися користувачем.

1.5.2. Вимоги до інформаційної безпеки

База даних повинна мати набір властивостей ACID. Пароль користувача повинен відповідати певним вимогам надійності і зберігатися в БД у захешованому надійним алгоритмом вигляді з додаванням до нього так званої “солі”. Кожне значення, яке потрапляє до бази даних повинно спочатку пройти валідацію. Авторизація повинна відбуватися за допомогою JWT токену і повністю обмежувати користувачу доступ до нотаток інших користувачів.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для підтримки стабільної роботи сервера, слід дотримуватися таким технічним вимогам:

– ОС Windows 10 або Linux;

– MySql Server;

– 2 гігабайти оперативної пам'яті;

– процесор x64 з тактовою частотою 2,5 ГГц;

– 10 гігабайт вільного місця на диску

Рекомендовані технічні характеристики для стабільної роботи клієнта:

– операційна система Android 8.1 (API 27) або вище, Windows 10, iOS 11 або вище;

– доступно не менше 1 ГБ вільного місця

– доступ до мережі інтернет;

1.5.4. Вимоги до інформаційної та програмної сумісності

Як основна мова програмування використовувався C#. Також використовується СУБД MySQL, база даних розроблена засобами Entity Framework Core, підхід Code First. Додаток було розроблено в Visual Studio Community 2022.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути серверна програма, яка реалізовує RESTful Web API та містить весь необхідний для роботи з нотатками функціонал, а також клієнтський додаток, який може бути встановлений на такі операційні системи, як Windows, Android та iOS.

Основний функціонал додатка полягає в можливості створення, редагування та видалення нотаток на різних пристроях без втрати прогресу, використовуючи один обліковий запис.

Також, для забезпечення кращого користувальницького досвіду додаток має реалізовувати наступні функції:

– зберігання JWT токена для авторизації в локальному сховищі. Це необхідно для того, щоб користувачу не потрібно було вводити свої дані від облікового запису при кожному вході в клієнтський додаток;

– зміна зовнішнього вигляду нотаток, щоб користувач міг швидко їх відрізняти;

– окрема директорія-смітник, в якій зберігаються видалені нотатки з можливістю їх відновлення або остаточного видалення;

– можливість відкатити зміни, які були зроблені під час редагування.

2.2. Опис застосованих математичних методів

При проєктуванні та розробці даного додатка використовувалися лише прості арифметичні дії. Математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проєктування

Серверна частина (BackEnd) дотримується архітектурного стиля REST та має багатошарову архітектуру.

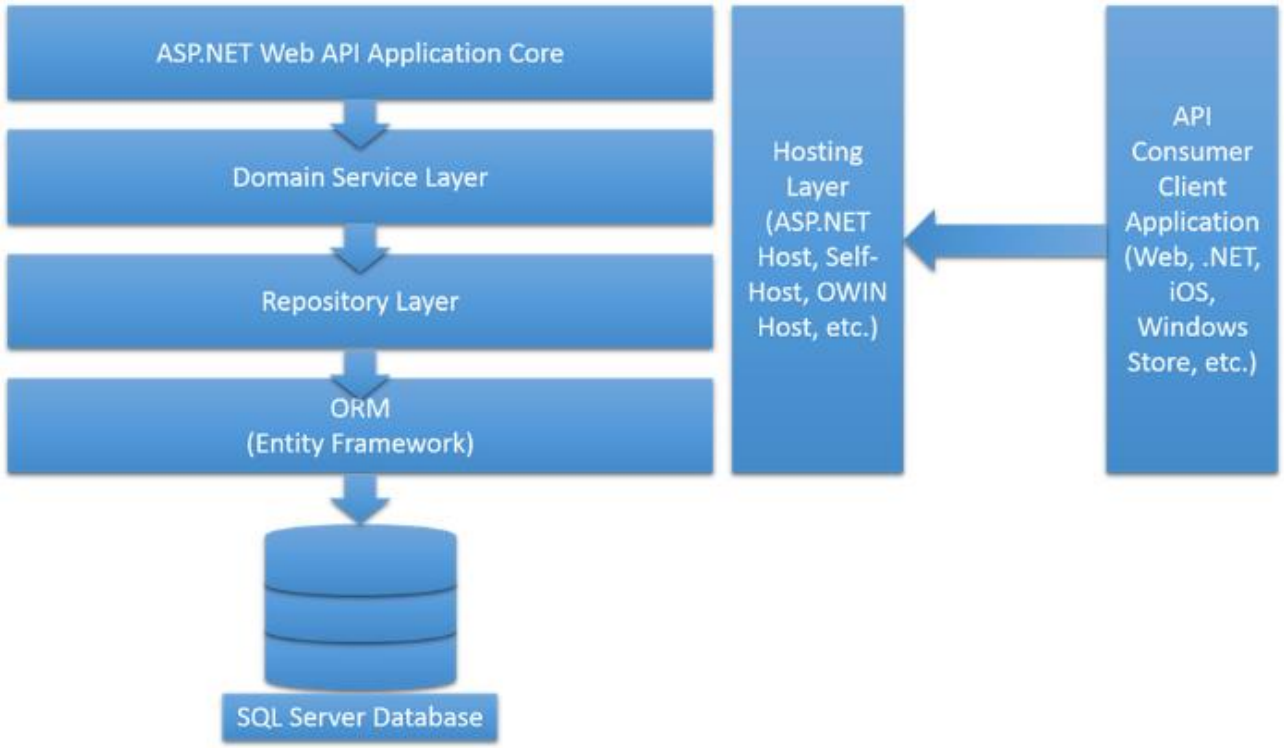

Рис.2.1. Графічне представлення багатошарової архітектури

У верхньому шарі зазвичай знаходяться контролери, які обробляють запити клієнтів. Другий рівень містить усі DTO, допоміжні класи та сервіси, які реалізовують бізнес логіку. Третій рівень містить моделі та репозиторії. Репозиторій – це посередник між ORM та сервісом бізнес логіки. Він відокремлює логіку доступу до даних і зіставляє моделі з відповідними DTO. Таким чином бізнес логіка працює тільки з DTO, а репозиторій працює безпосередньо з моделями даних, та віддає рівню бізнес логіки DTO, приховуючи від нього деталі доступу до даних. Репозиторії взаємодіють не напряму з базою даних, а з EntityFramework. EntityFramework – це ORM, яка дає змогу розробникам працювати з даними, використовуючи об’єкти специфічних для домену класів, не зосереджуючи увагу на таблицях і стовпцях бази даних, де ці дані зберігаються.

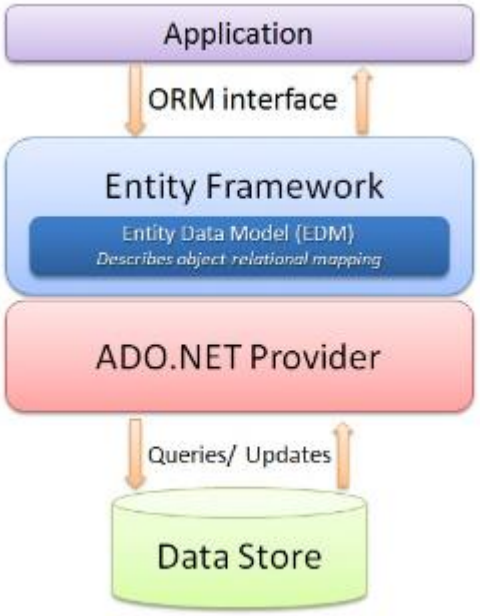

Рис.2.2. Схема роботи EntityFramework

Така архітектура дозволяє поділити проєкт на незалежні модулі, які можна легко заміняти та покривати тестами.

Клієнтська частина (FrontEnd) розроблена за патерном MVVM. MVVM дозволяє відокремити логіку програми від візуальної частини (UI). Це також може значно покращити можливості повторного використання коду та дозволяє розробникам і дизайнерам інтерфейсу користувача легше співпрацювати під час розробки відповідних частин програми. Цей патерн є архітектурним, тобто він задає загальну архітектуру програми.

Трохи детальніше про компоненти MVVM:

– Model – описує дані, які використовуються у програмі, може також містити логіку, яка безпосередньо пов’язана з даною моделлю.

– View – описує UI та може містити специфічну логіку, пов’язану з користувальницьким інтерфейсом, яку важко реалізувати у ViewModel.

– ViewModel – містить основну логіку, а також пов’язує модель та уявлення за допомогою механізму прив’язки даних, що дозволяє змінювати UI при зміні моделі, хоча напряму вони і не пов’язані.

2.4. Опис використаних технологій та мов програмування

Серверна частина була розроблена за допомогою таких технологій:

- ASP.NET Core Web API;
- C#;
- БД MySql;
- EntityFramework;
- JSON.
- JWT.
- Swagger.

Клієнтська частина була розроблена за допомогою таких технологій:

- Xamarin.Forms;
- C#;
- FreshMVVM;
- JSON.
- JWT.

ASP.NET Core Web API – це фреймворк для створення HTTP сервісів, які можуть використовуватись будь-яким клієнтом включаючи мобільні пристрої, ідеальний варіант для розробки кросплатформеного додатка з RESTful архітектурою.

C# – C-подібна об'єктно-орієнтована мова програмування з суворою типізацією для платформи .NET. Компанія Microsoft вирішила створити свій власний аналог Java – мову, для якої корпорація буде повноцінним власником. Ця новостворена мова отримала назву C#. Вона успадкувала від Java концепції віртуальної машини (середовище .NET), байт-коду (MSIL) і більшої безпеки вихідного коду програм, а також врахувала досвід використання програм на Java. Нововведенням C# стала можливість легшої взаємодії, порівняно з мовами-попередниками, з кодом програм, написаних на інших мовах, що є важливим при створенні великих проєктів. Якщо програми на різних мовах виконуються на

платформі .NET, саме платформа бере на себе клопіт щодо сумісності програм. Станом на сьогодні C# визначено флагманською мовою корпорації Microsoft, бо вона найповніше використовує нові можливості .NET. Решта мов програмування, хоч і підтримуються, але визнані такими, що мають спадкові прогалини щодо використання .NET.

MySql – це реляційна база даних з відкритим вихідним кодом, вона безкоштовна і проста в налаштуванні, тому є ідеальним варіантом.

JSON – це стандартний текстовий формат для представлення структурованих даних. Він використовується для передачі даних (наприклад, надсилання деяких даних із сервера клієнту, або навпаки).

Swagger – це набір інструментів, які допомагають описувати API. Завдяки йому користувачі краще розуміють можливості RESTful API без доступу до коду. За допомогою Swagger можна швидко створити документацію та надіслати її іншим розробникам чи клієнтам. В проєкті використаний підхід автогенерації документації на основі коду. Таким чином весь функціонал сервера задокументований і наочний, що дозволяє зручно тестувати його роботу.

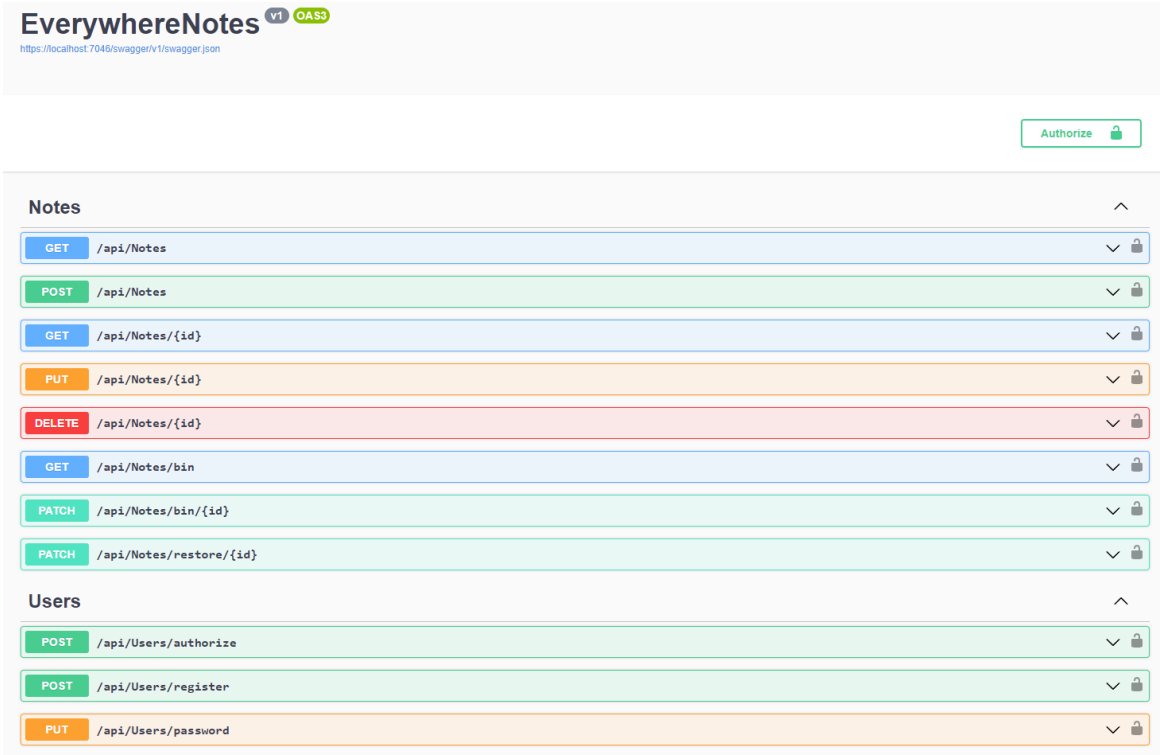

Рис.2.3. Сторінка Swagger

Xamarin.Forms – це кросплатформний фреймворк з відкритим вихідним кодом від Microsoft для створення додатків iOS, Android і Windows за допомогою .NET зі спільною кодовою базою. Xamarin.Forms дозволяє розробникам створювати користувацькі інтерфейси в XAML із кодом на C#. Ці інтерфейси на кожній платформі відображаються як нативні елементи керування. Під час виконання Xamarin.Forms використовує засоби візуалізації платформи для перетворення кросплатформених елементів інтерфейсу користувача у нативні елементи керування на Xamarin.Android, Xamarin.iOS та UWP. Це дозволяє розробникам отримати оригінальний вигляд, користувальницький досвід та продуктивність, усвідомлюючи переваги спільного використання коду між платформами. Програми Xamarin.Forms зазвичай складаються із спільної бібліотеки .NET Standard та окремих проектів для кожної платформи. Спільна бібліотека містить представлення XAML або C# і будь-яку бізнес-логіку, наприклад сервіси, моделі чи інший код. Проекти платформ містять будь-яку специфічну для платформи логіку або пакети, які потрібні додатку.

FreshMVVM – це надлегкий, на відміну від того ж MVVMCross, фреймворк MVVM, розроблений спеціально для Xamarin.Forms. Саме це і є одною з його найбільших переваг, адже фреймворки як MVVMCross були розроблені ще до виходу Xamarin.Forms, отже вони з самого початку були більше заточені під нативні проєкти.

Основні можливості FreshMVVM:

– Навігація від ViewModel до ViewModel.

– Автоматична прив’язка View до ViewModel.

– Автоматична прив’язка до подій View (наприклад поява на екрані).

– Базові методи ViewModel, за допомогою яких можна передавати об’єкти між в’юмоделями при навігації та організовувати взаємодію з користувачем.

– Вбудований IoC контейнер, який дозволяє ін’єкцію залежностей в конструкторі в’юмоделі.

– Вбудовані контейнери для простої навігації, з вкладками або з боковим меню.

JWT – це відкритий стандарт, який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити й довіряти, оскільки вона містить цифровий підпис. Підписані токени можуть перевірятися за значеннями, що містяться в них. Коли токени підписуються за допомогою пар відкритий/приватний ключ, підпис також засвідчує, що підписала його лише сторона, яка володіє приватним ключем. Авторизація – найчастіший варіант використання JWT. Щойно користувач увійде в систему, кожен наступний запит включатиме JWT, що дозволить йому отримати доступ до маршрутів, служб і ресурсів, які дозволені цим токеном.

2.5. Опис структури програми та алгоритми її функціонування

Для зображення сценаріїв використання системи користувачами використовуються діаграми варіантів використання (use case diagrams). Сценарії використання визначають очікувану поведінку (що), а не точний спосіб її здійснення (як). Випадки використання після вказівки можуть бути позначені як текстовим, так і візуальним представленням. Ключова концепція моделювання варіантів використання полягає в тому, що воно допомагає нам проектувати систему з точки зору кінцевого користувача. Це ефективна техніка для передачі поведінки системи в термінах користувача шляхом визначення всієї зовнішньої поведінки системи. Зазвичай такі діаграми досить прості. Вони не показують жодних деталей реалізації, а лише узагальнюють деякі зв’язки між варіантами використання, акторами та системами. Вони також не показують порядок, у якому виконуються кроки для досягнення цілей кожного варіанта використання.

Варіанти використання представляють лише функціональні вимоги системи. Інші вимоги, такі як бізнес-правила, вимоги до якості обслуговування та обмеження реалізації, повинні бути представлені окремо.

Основні цілі use case діаграми:

- Визначення контексту системи.
- Врахування вимог системи.

– Перевірка архітектури системи.

– Підтримка виконання та створення тест-кейсів для подальшого тестування.

Символом людини на цій діаграмі позначаються актори, еліпсом – варіанти використання, а прямокутник – це система. В даному проєкті актором може виступати новий, або вже зареєстрований користувач, а системою – серверний та клієнтський додатки.

Для того, щоб отримати доступ до створення, редагування та перегляду заміток користувачу потрібно зареєструватися. До цього йому буде доступний лише функціонал, дозволяючий створити новий обліковий запис або увійти в існуючий. Після авторизації користувач також може змінити свій пароль.

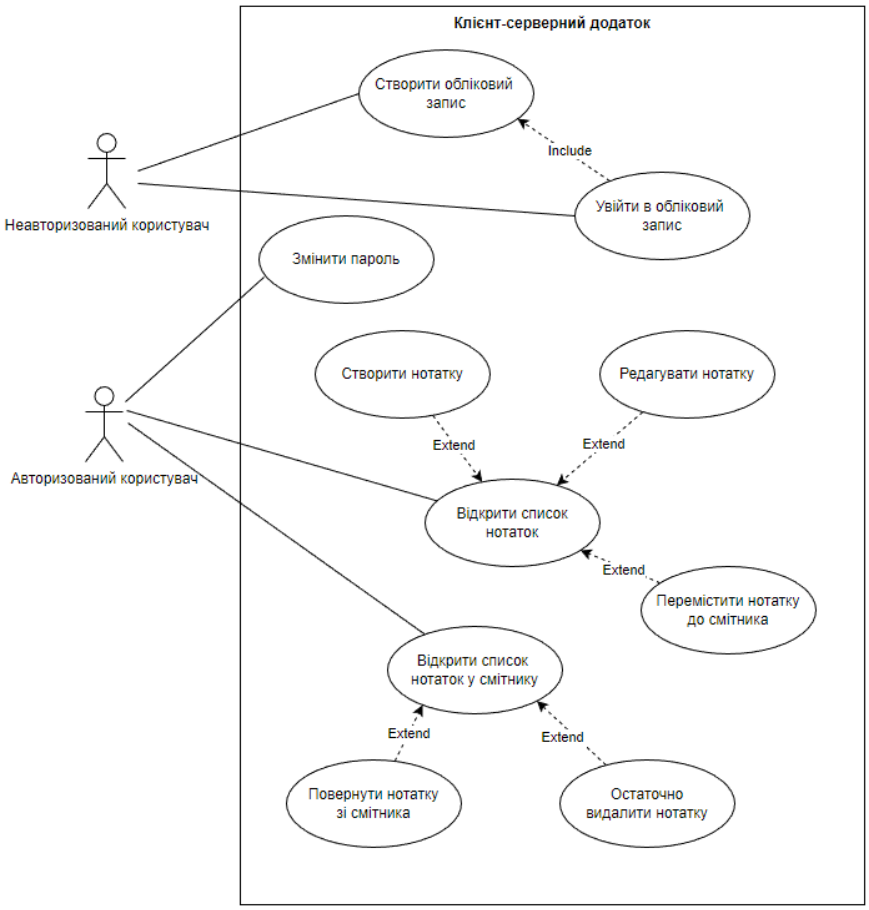

Рис.2.4. Діаграма варіантів використання

Діаграма активності є ще однією важливою діаграмою поведінки в UML для опису динамічних аспектів системи. Діаграма активності, по суті, є розширеною версією блок-схеми, яка моделює перехід від однієї діяльності до іншої. Діаграми діяльності описують, як діяльність координується для надання послуги, яка може бути на різних рівнях абстракції. Як правило, подія повинна бути досягнута деякими операціями, особливо якщо операція призначена для досягнення ряду різних речей, які потребують координації, або як події в одному випадку використання пов’язані одна з одною, зокрема, випадки використання, коли дії можуть перетинатися і вимагати координації. Вона також підходить для моделювання того, як набір варіантів використання координується для представлення робочих процесів бізнесу.

Основні переваги Activity діаграми:

– Продемонструвати логіку алгоритму.

– Опис кроків, які виконуються у варіанті використання.

– Ілюстрація робочого процесу між користувачами та системою.

– Спрощення і покращення будь-якого процесу, прояснивши складні варіанти використання.

Рис.2.5. Діаграма активності: реєстрація

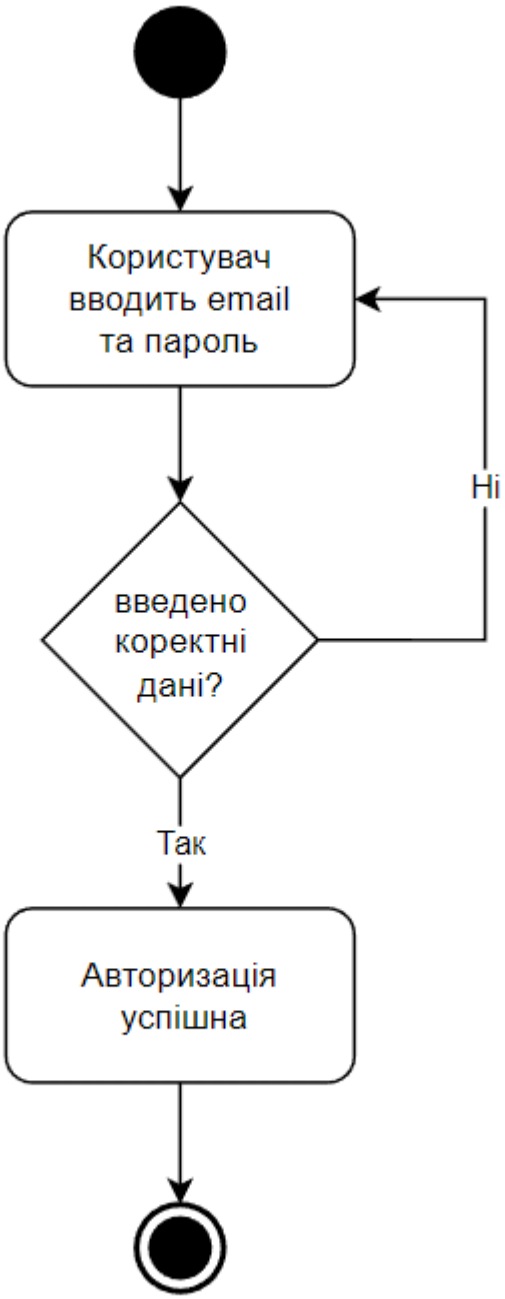

Рис.2.6. Діаграма активності: авторизація

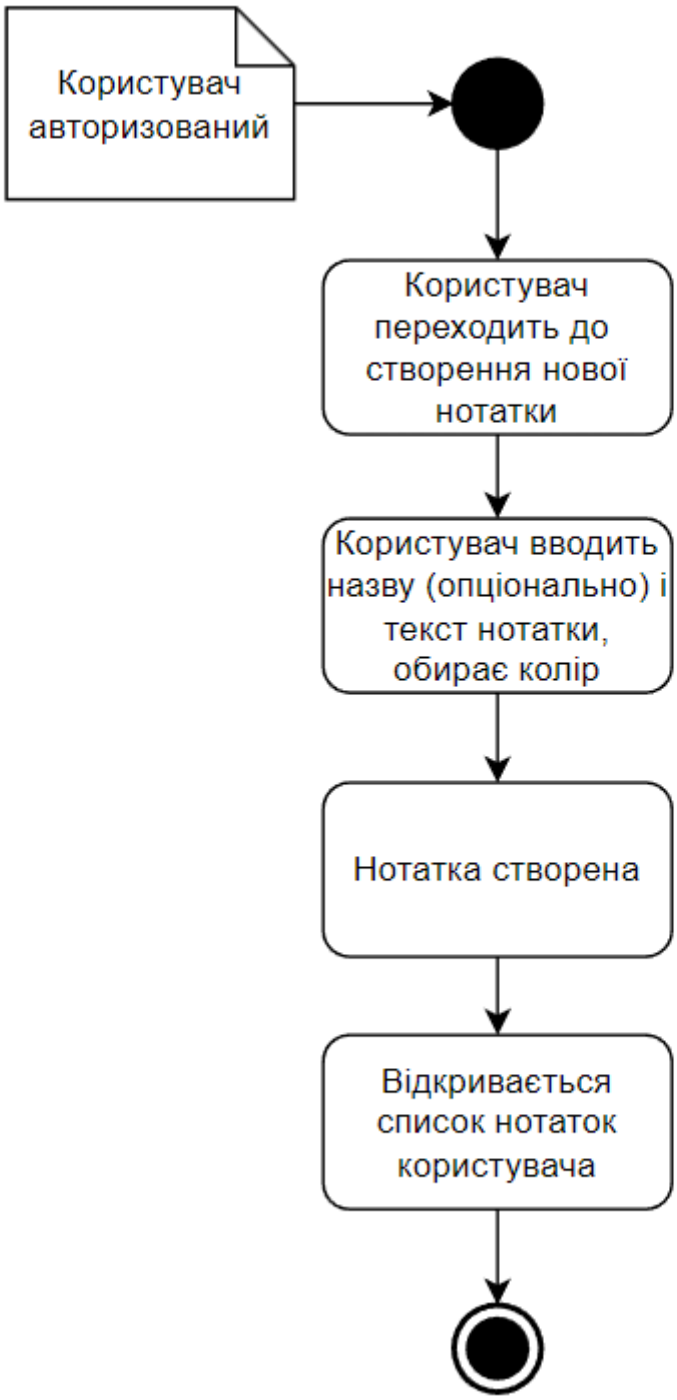

Рис.2.7. Діаграма активності: створення нотатки

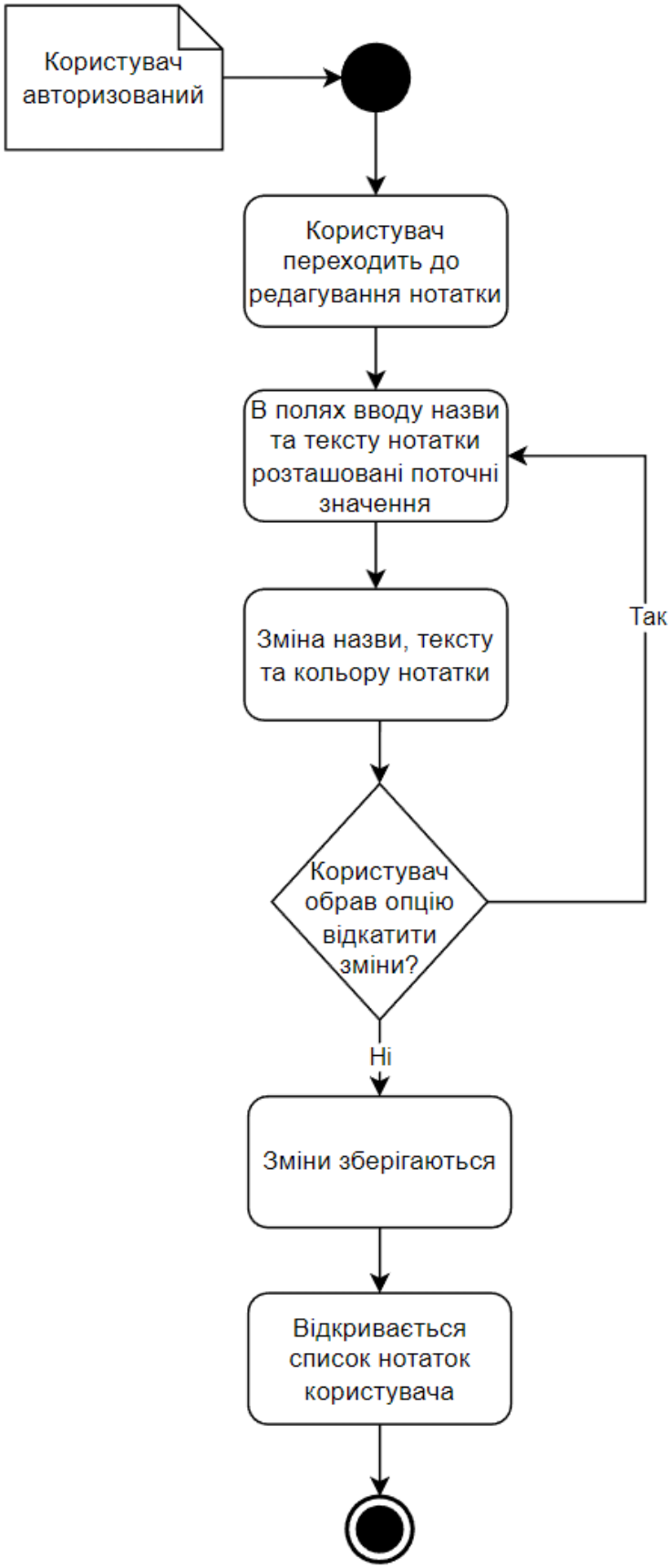

Рис.2.8. Діаграма активності: редагування нотатки

Рис.2.9. Діаграма активності: остаточне видалення нотатки

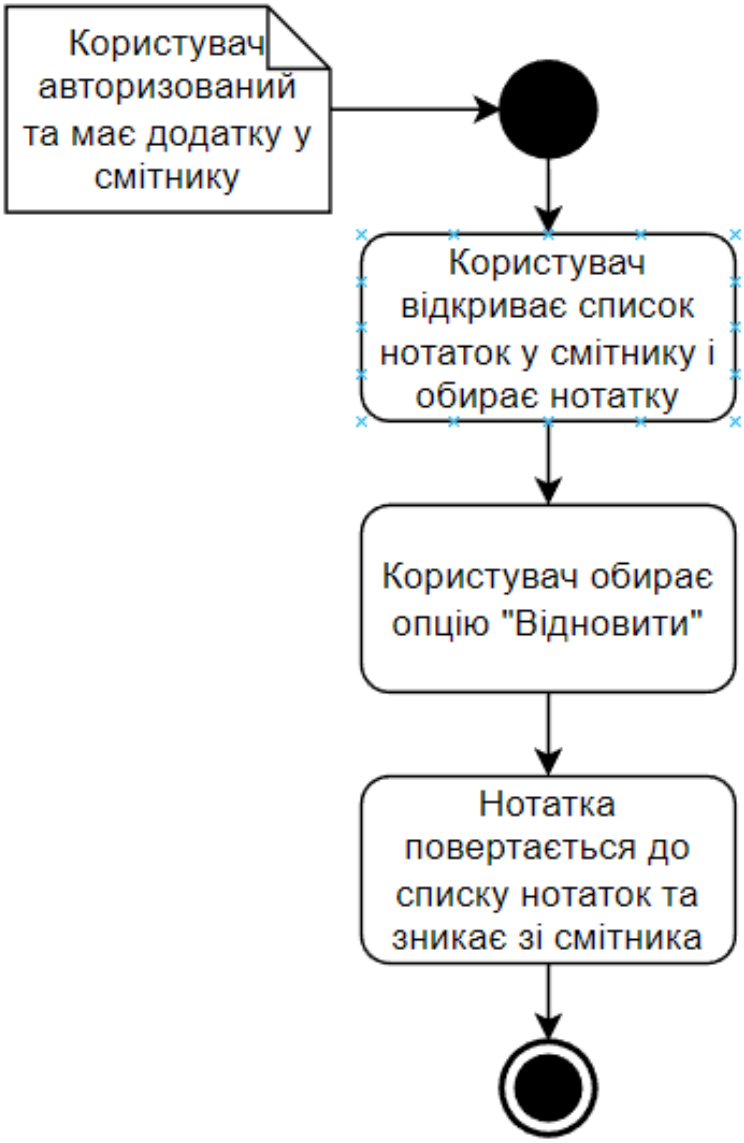

Рис.2.10. Діаграма активності: відновлення нотатки

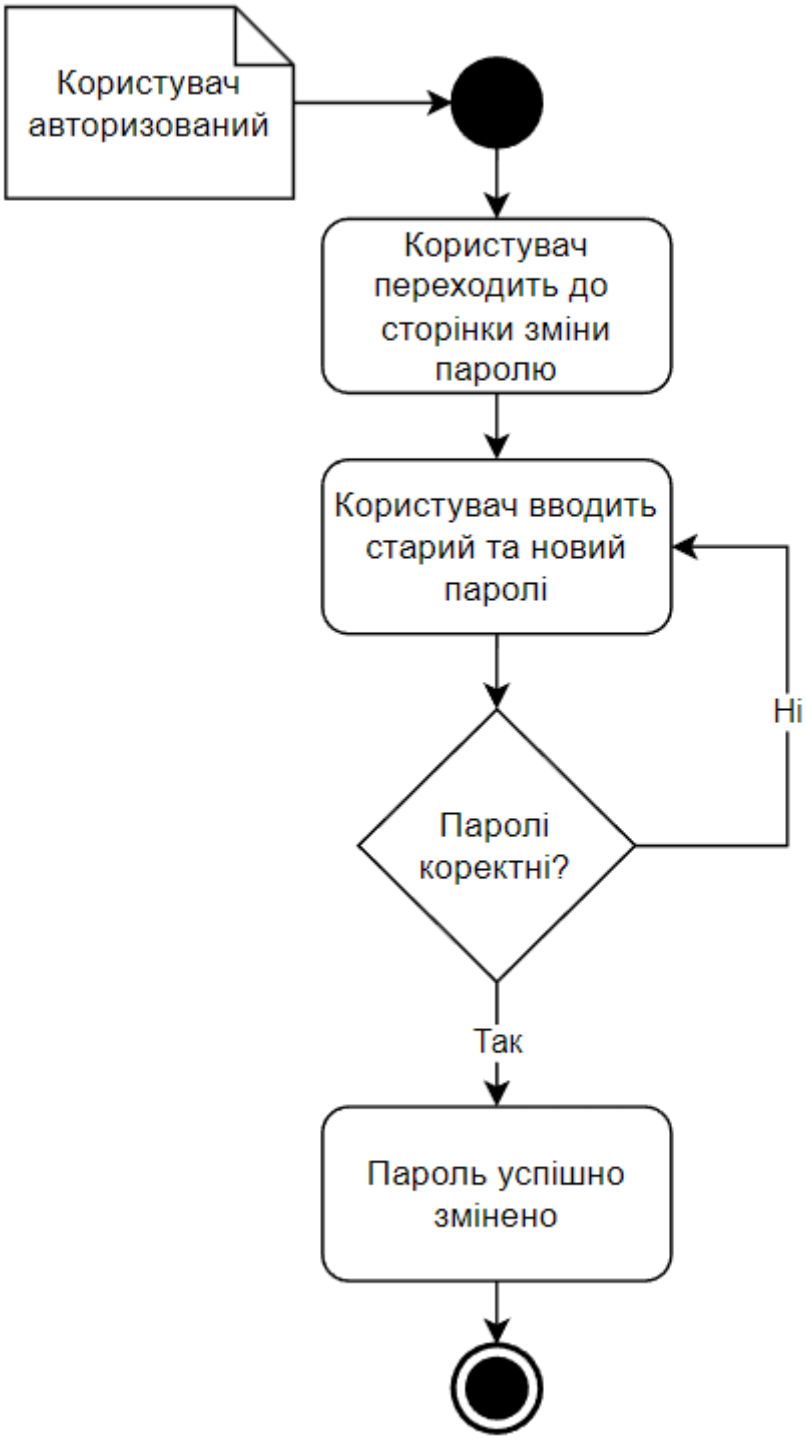

Рис.2.11. Діаграма активності: зміна паролю

Діаграма взаємовідносин сутностей (Entity Relationship Diagram) – це візуальне представлення різних сутностей у системі та їх взаємозв’язку. Вона широко використовується для проектування реляційних баз даних. Оскільки сутності в ER діаграмі можна використовувати для візуалізації таблиць бази даних

та їх зв’язків, вони також зазвичай використовуються для усунення несправностей бази даних.

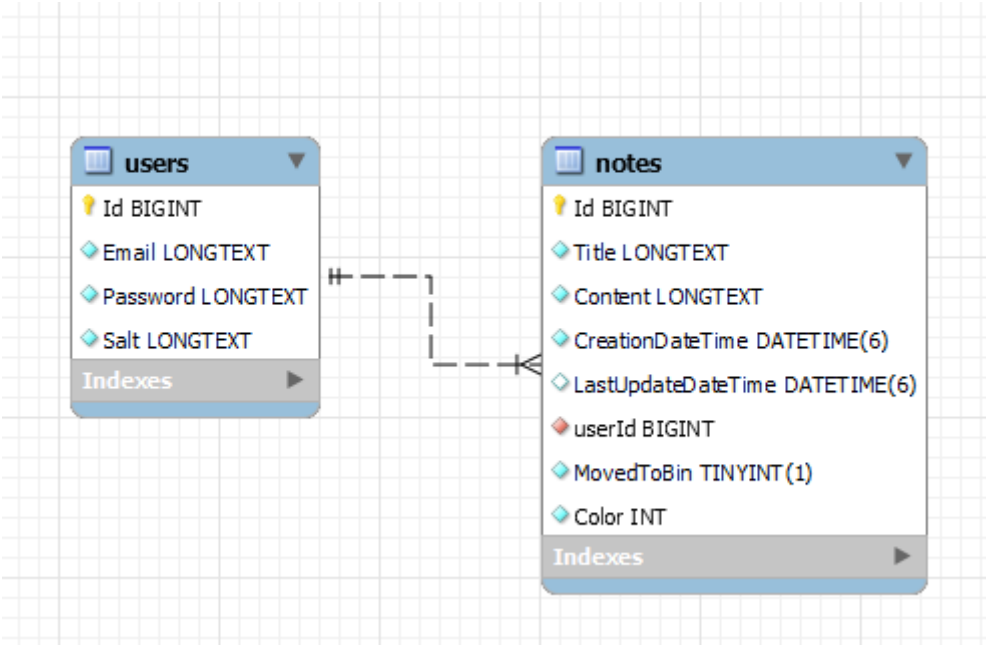

Рис.2.12. ER діаграма бази даних

2.6. Обґрунтування та організація вхідних та вихідних даних програми

В даній кваліфікаційній роботі вхідними даними можуть бути особисті дані клієнтів, такі як електронна пошта та пароль від облікового запису. При цьому пароль дуже важливо зберігати таким чином, щоб забезпечити неможливість його викрадення, адже отримавши доступ до паролю користувача, зловмисник зможе отримати доступ до всіх його нотаток, що неприпустимо. Тому необхідно використати кілька спеціальних методів для захисту пароля.

Основним з них є хешування пароля. Хешування – це проходження пароля через певний алгоритм хешування, щоб перетворити відкритий текст у незрозумілий ряд цифр і букв. Це важливо для базової гігієни безпеки, оскільки в разі порушення безпеки будь-які скомпрометовані паролі будуть незрозумілими для зловмисника. Як наслідок, крадіжка цієї інформації значно ускладнюється.

Але, попри все, підбір хешованого пароля все ще можливий. Так, це займе значно більше часу, але при наявності у зловмисника хеш-таблиць він може

спробувати знайти співпадіння хешу з хешованим паролем та в результаті отримати пароль користувача. Щоб звести вирогідність такого підбору до мінімума використовується сіль (salt). В цьому випадку сіль – це значення, створене криптографічно захищеною функцією, яке додається до хешованого пароля для створення унікальних хешів, незалежно від того, чи унікальний цей пароль. Сіль робить хеш-функцію недетермінованою, тобто навіть два однакових пароля матимуть різні хеші.

З інших вхідних даних можна виділити різні дані про нотатку, такі як її назва, основний текст та вибраний колір. Також сервер автоматично зберігає інформацію про дату та час створення і останнього редагування нотатки.

Вихідними даними є інформація про замітки, яка зберігається у БД, у зручному для графічного представлення вигляді.

2.7. Опис розробленого програмного продукту

Для функціонування додатка, серверна програма разом з сервером бази даних повинна бути розгорнута на певному хостингу, щоб клієнтський додаток міг робити запити і отримувати відповідь.

2.7.1. Використані технічні засоби

При розробці ПЗ була використана персональна ЕОМ з наступними характеристиками:

– Процесор AMD Ryzen 7 5800X;

– Відеопроцесор Nvidia GeForce RTX 2080 Super (8 ГБ GDDR6);

– Оперативна пам'ять 32 ГБ DDR4-3600.

– SSD M.2 на 1 ТБ.

– Материнська плата з підтримкою віртуалізації.

2.7.2. Використані програмні засоби

Для розробки серверного та клієнтського застосунків використовувались такі програмні засоби:

- Visual Studio 2022;
- Git, GitHub;
- Visual Studio Emulator for Android;
- MySql Workbench;

2.7.3. Виклик та завантаження програми

Після того, як серверна частина успішно запущена, можна запускати клієнтський додаток. Так як клієнтський застосунок є кросплатформеним, його можна встановити і запустити на будь-якій з трьох платформ: Android, iOS та Windows. Клієнтський додаток буде надсилати запити серверу на відповідну URL-адресу, яка є стандартизованою та доступною в мережі інтернет. Більш того, так як сервер представляє собою WEB API, він не має жодних обмежень щодо типу клієнта. Тому, окрім розробленого додатка для перелічених вище платформ, завжди можна розробити й інші, наприклад для веб-браузерів. Також, для тестування роботи сервера окремо від клієнта можна використовувати Swagger, який включено до проєкту, або Postman.

2.7.4. Опис інтерфейсу користувача

Інтерфейс клієнтського додатка є зручним та інтуїтивно зрозумілим. Після запуску програми, користувач, якщо він ще не авторизований, бачить сторінку авторизації (рис. 2.13), з якої можна перейти до реєстрації (рис. 2.14) або, у разі успішної авторизації, на головну сторінку додатка (рис. 2.15). В залежності від платформи клієнта (Desktop чи Mobile) ця сторінка може відрізнятися. Для мобільної версії реалізоване бокове меню (рис. 2.16), яке знаходиться зліва і

містить опції для переходу між головною сторінкою, смітником (рис. 2.18) та налаштуваннями облікового запису (рис. 2.19). У версії для Windows замість бокового меню присутні вкладки у верхній частині інтерфейсу (рис. 2.17). Натискаючи кнопку “new (+)” користувач переходить на сторінку створення нової нотатки (рис. 2.20), при цьому у правому верхньому куті є кнопка для вибору кольору для нотатки. Після внесення бажаних змін можна повернутися на головну сторінку, нова нотатка з’явиться у списку. При виборі нотатки зі списку відкривається сторінка редагування нотатки (рис. 2.21). Вона схожа на сторінку створення нової нотатки, але з певними відмінностями. По перше, поля для вводу назви та тексту нотатки будуть заповнені поточними значеннями. А по друге, у правому верхньому куті з’явиться ще одна кнопка, яка відкриває опції, серед яких переміщення нотатки до смітника та відкат змін. Якщо ж вибрати нотатку з тих, що знаходяться у смітнику, відкривається схожа сторінка, але робити зміни в ній неможна, а з опцій є видалення на завжди та відновлення нотатки.

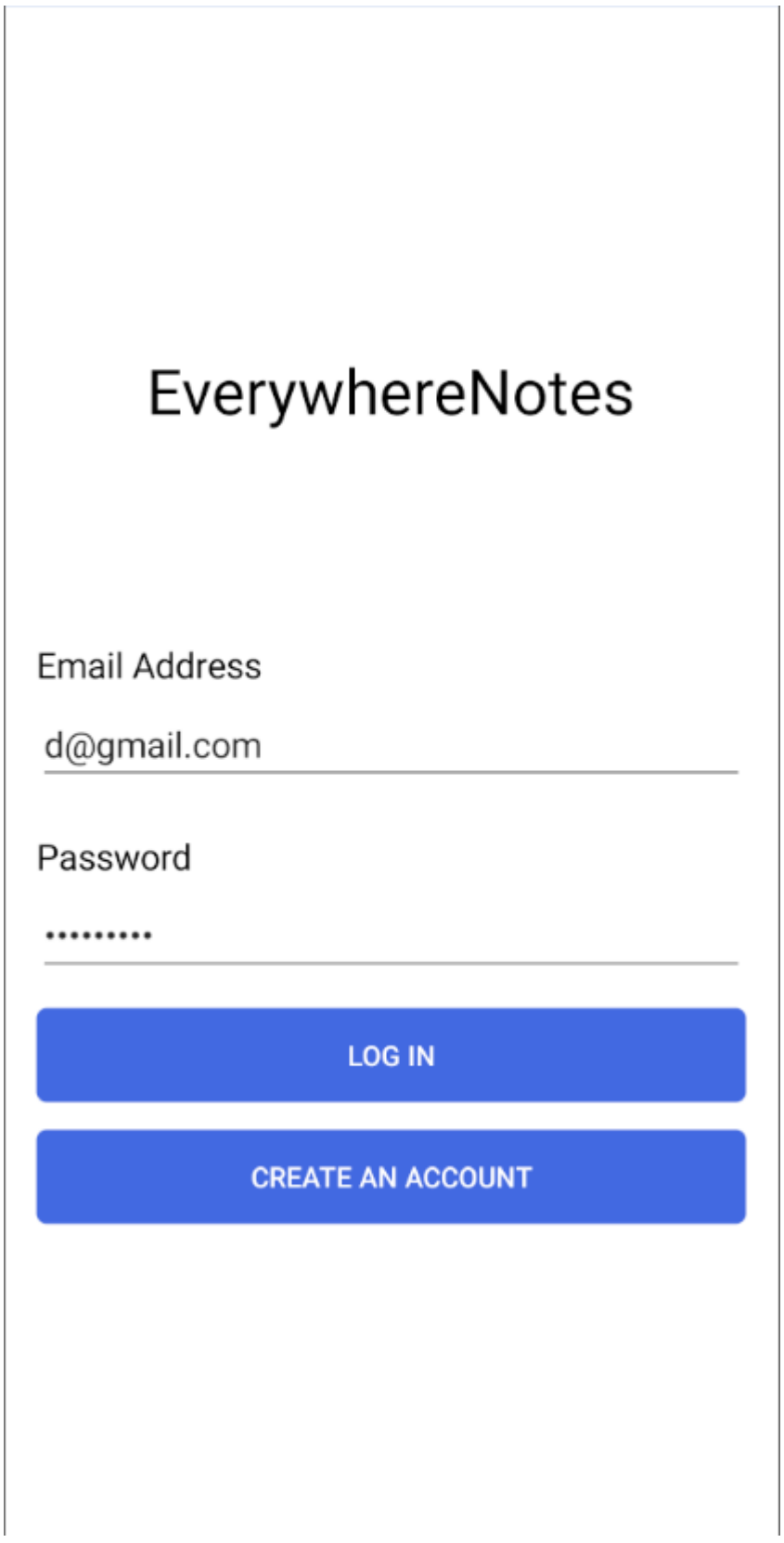

Рис.2.13. Сторінка авторизації

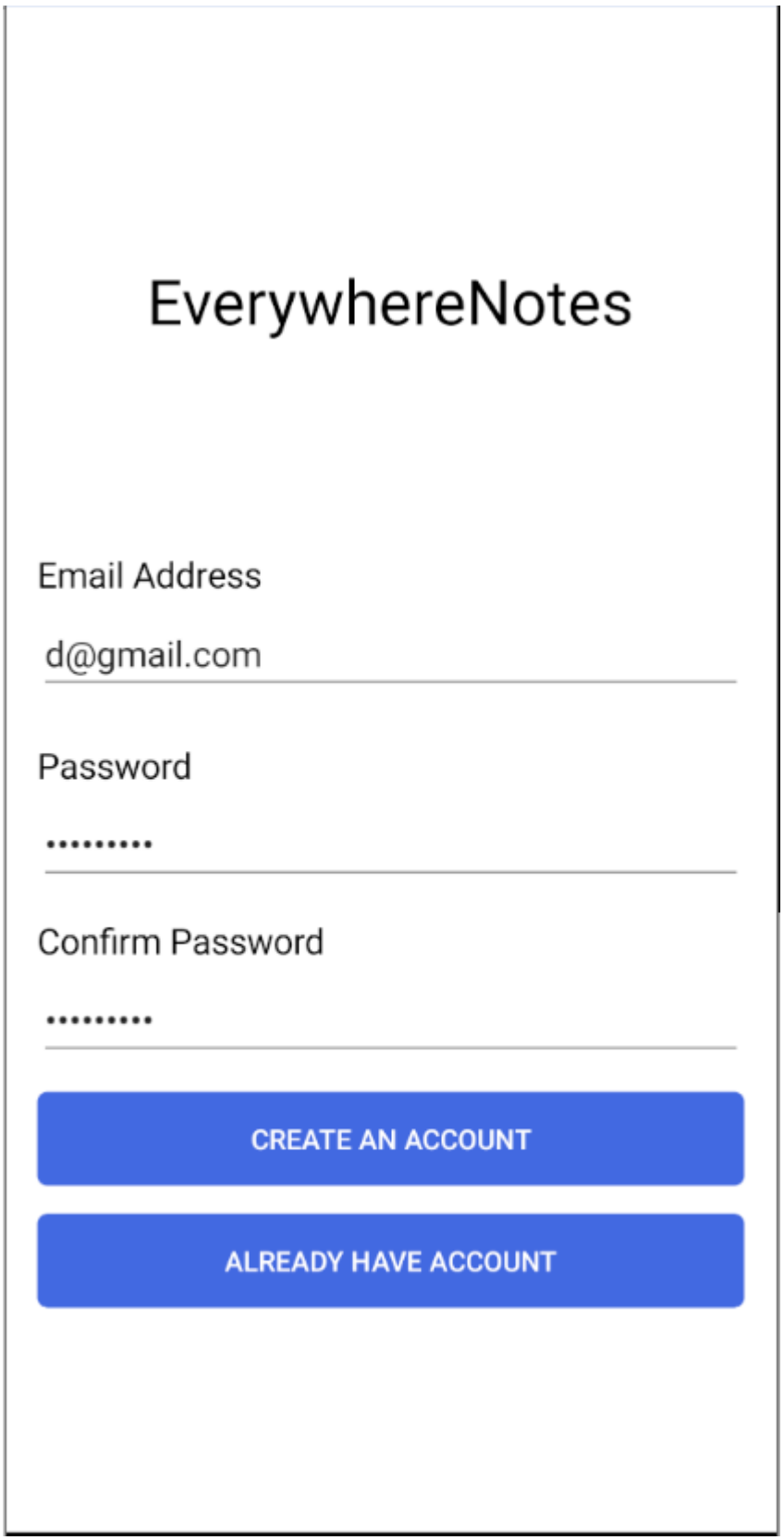

Рис.2.14. Сторінка реєстрації

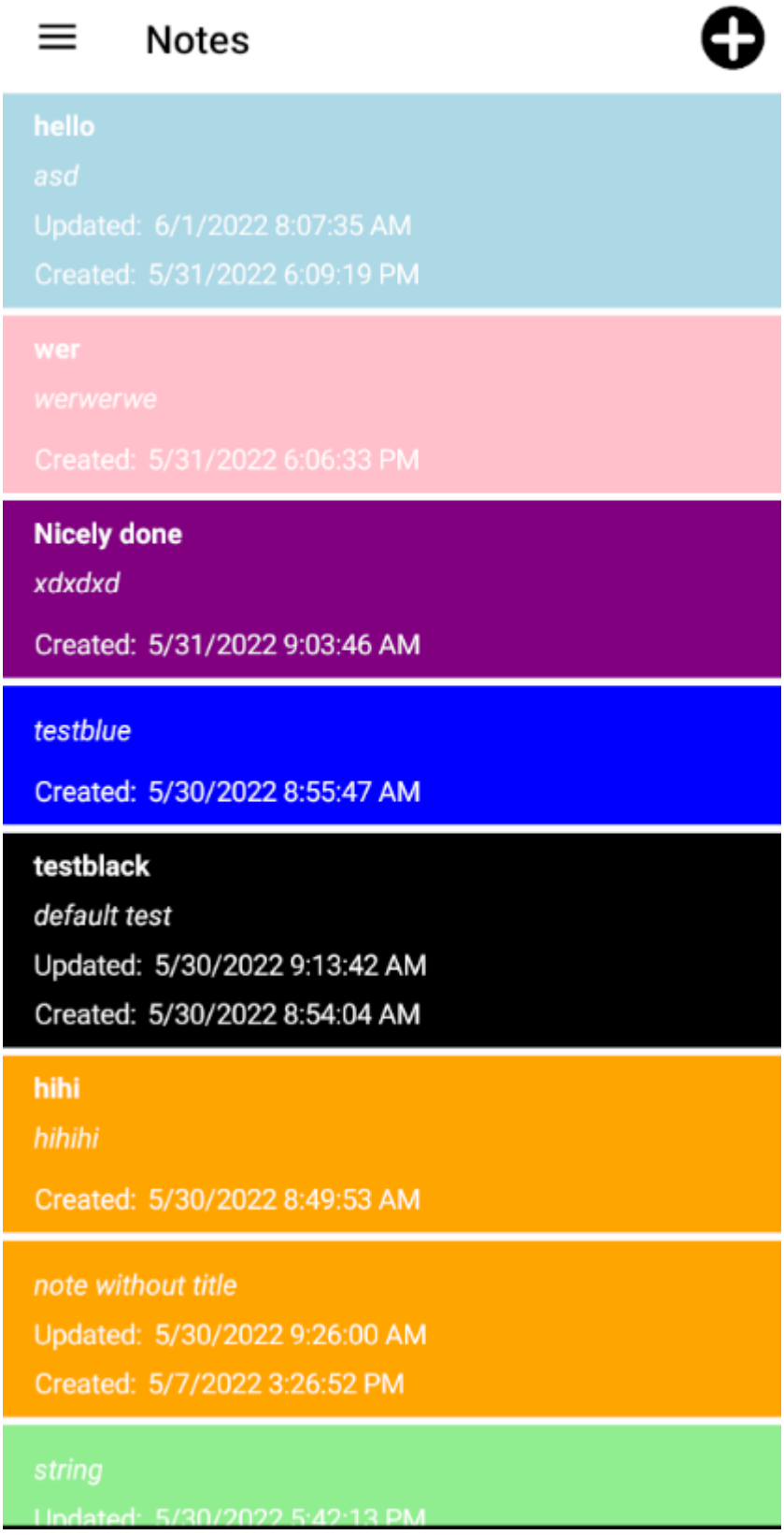

Рис.2.15. Головна сторінка

Рис.2.16. Бокове меню в мобільній версії додатка

Рис.2.17. Вкладки у версії додатка для ОС Windows

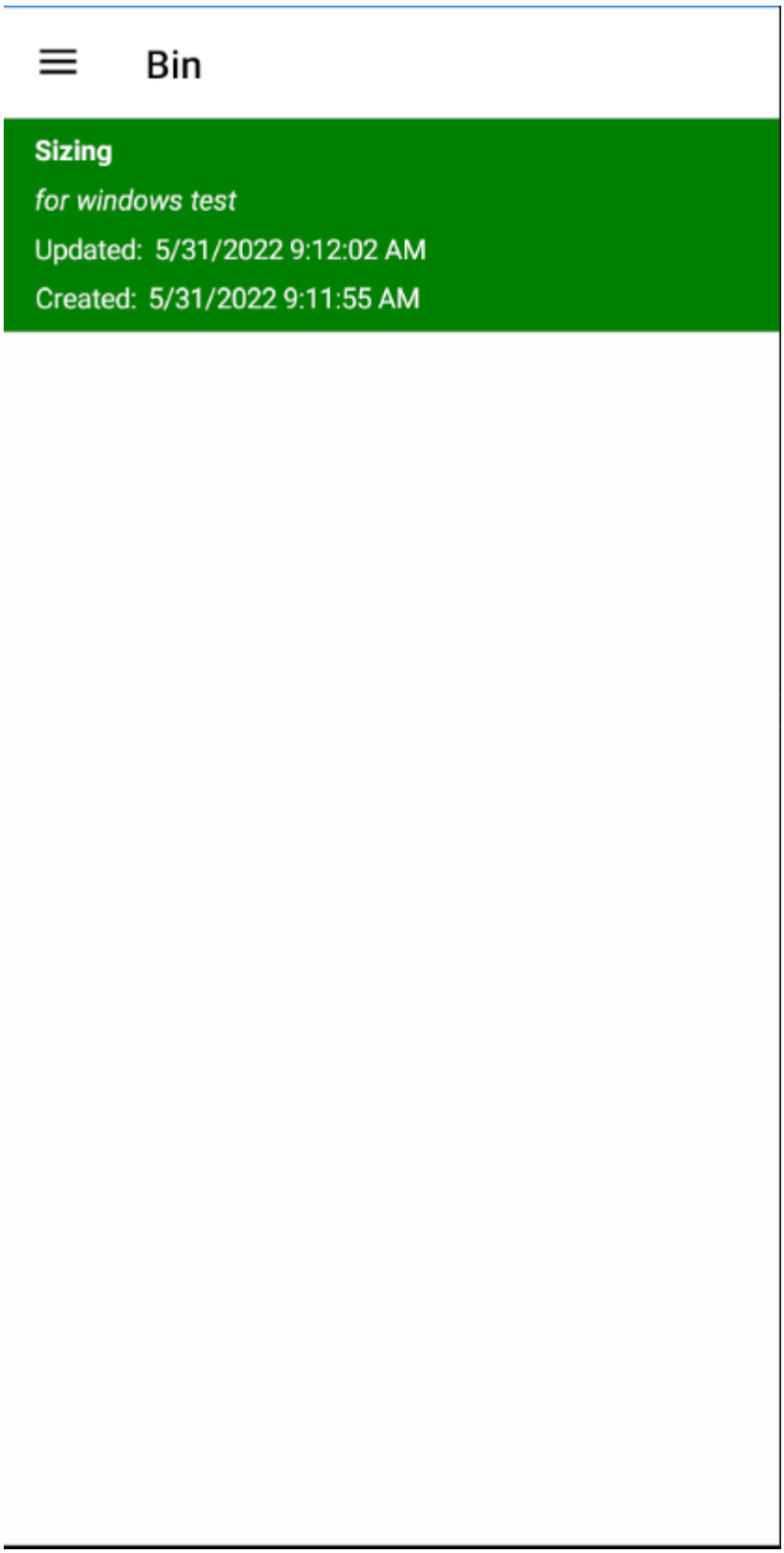

Рис.2.18. Сторінка смітника

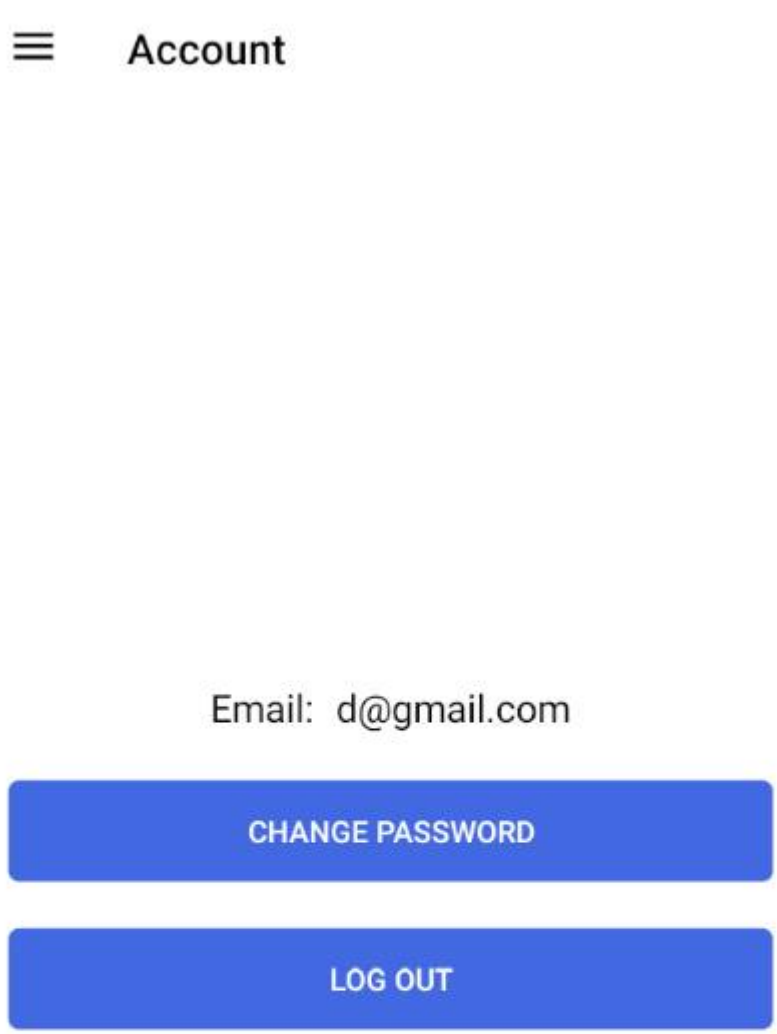

Рис.2.19. Сторінка налаштувань облікового запису

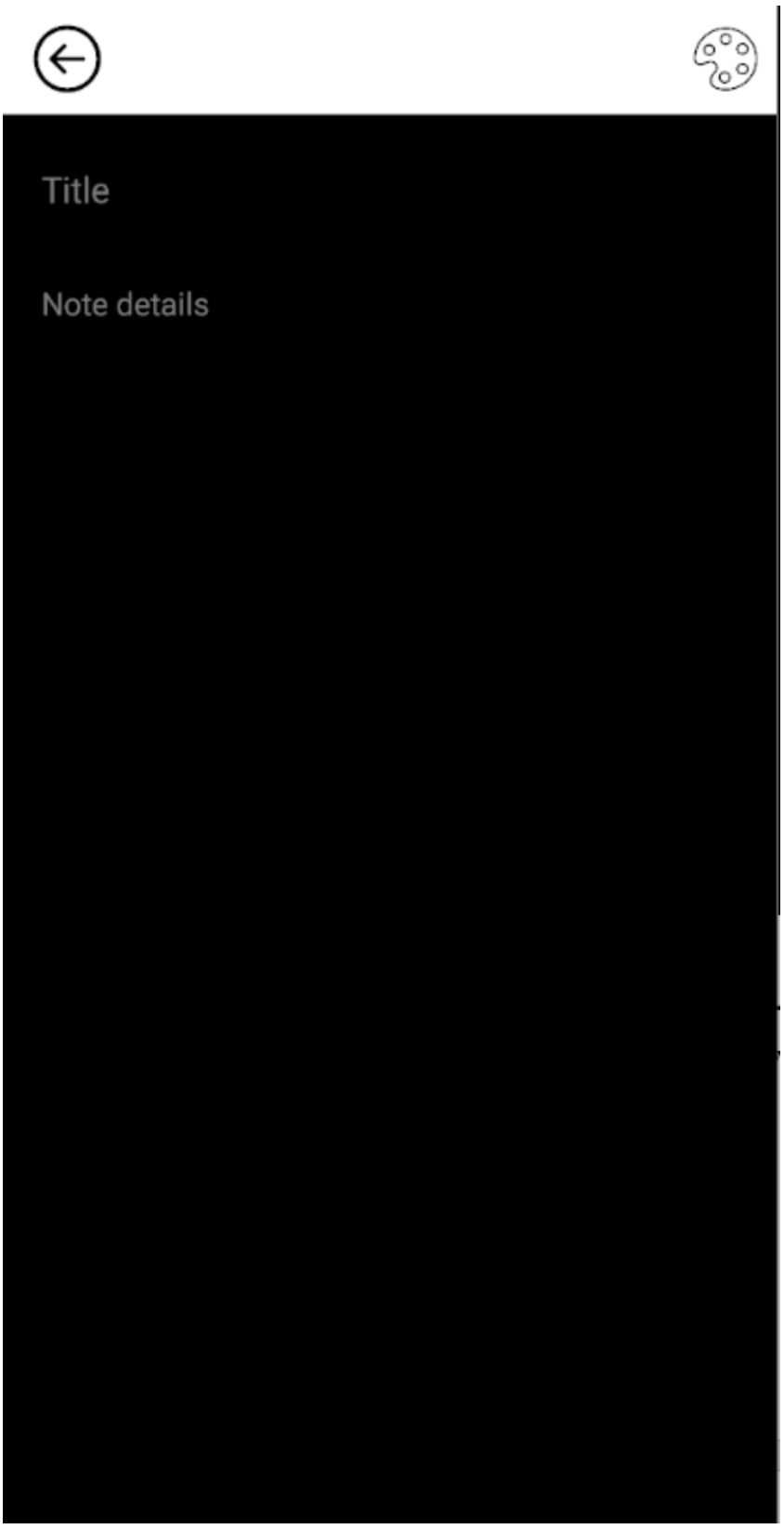

Рис.2.20. Сторінка створення нової нотатки

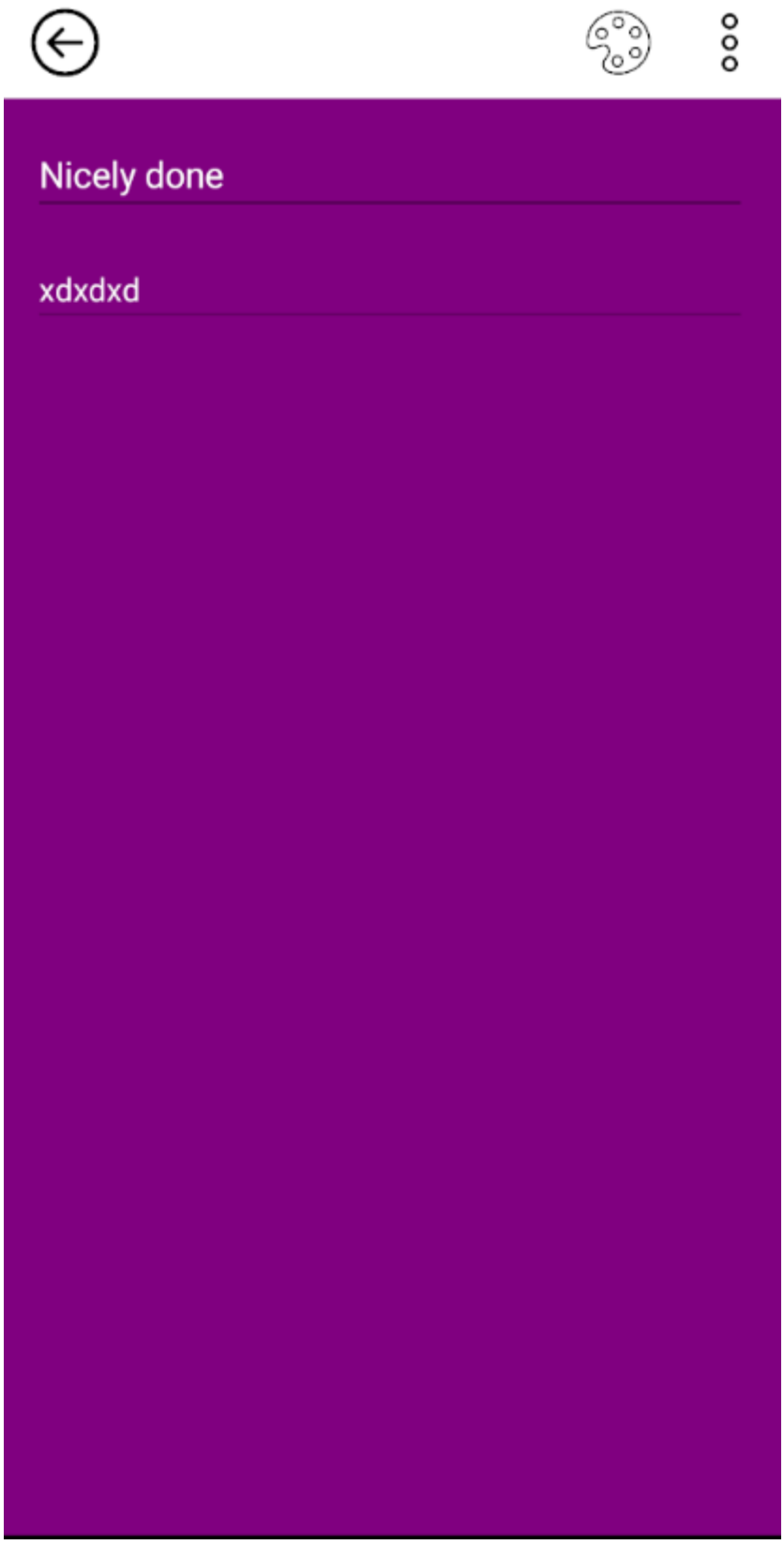

Рис.2.21. Сторінка редагування нотатки

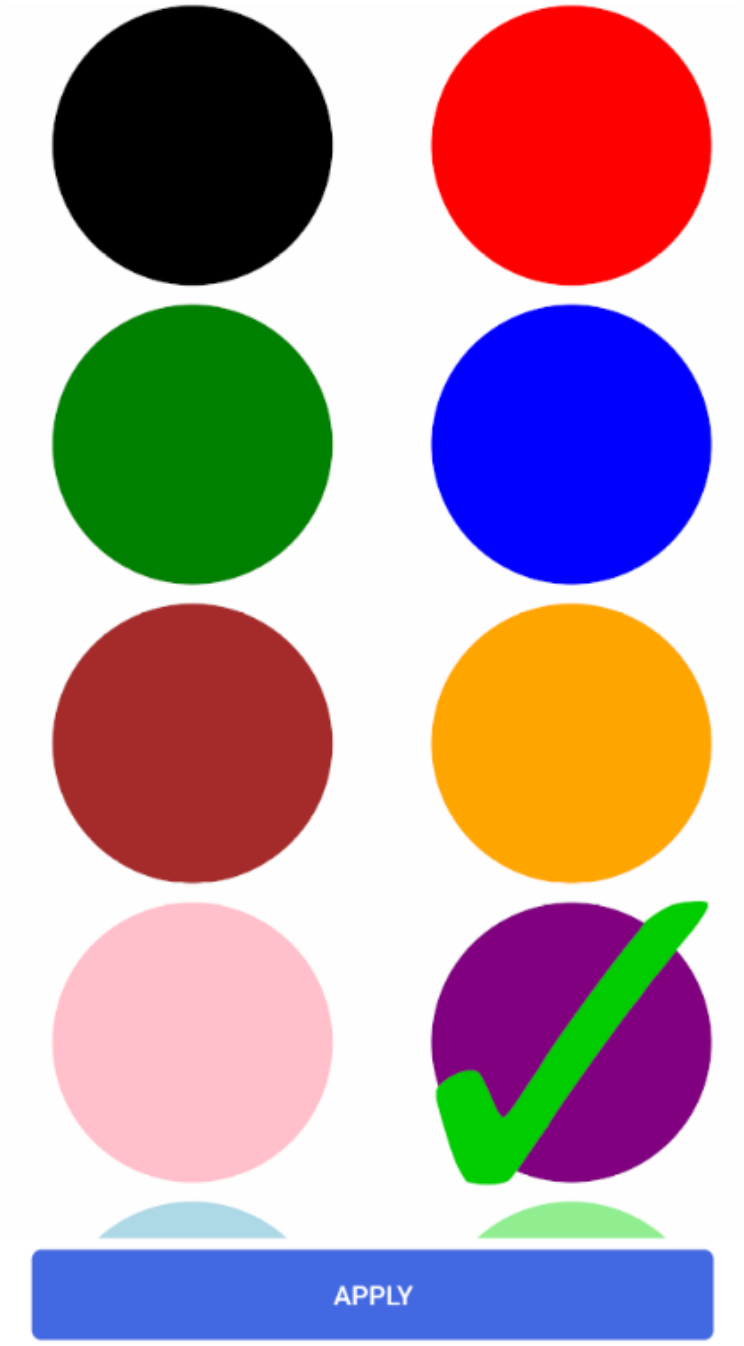

Рис.2.22. Сторінка вибору кольора для нотатки

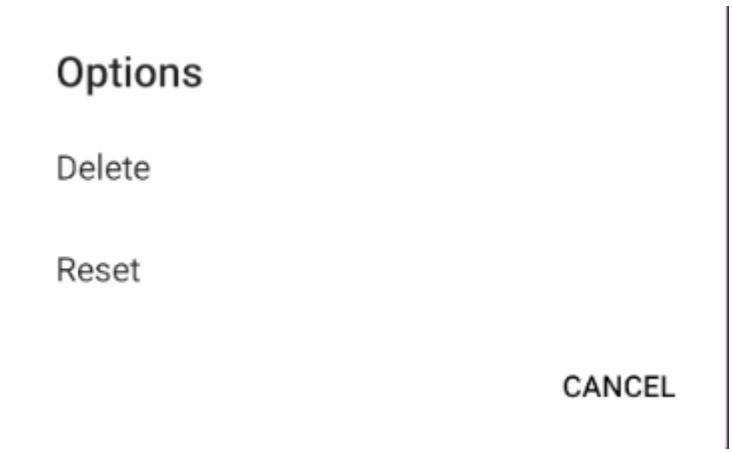

Рис.2.23. Опції на сторінці редагування нотатки

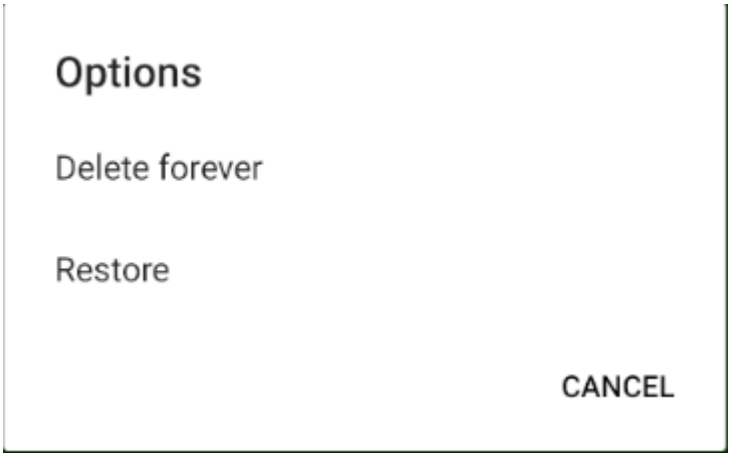

Рис.2.24. Опції для нотатки в смітнику

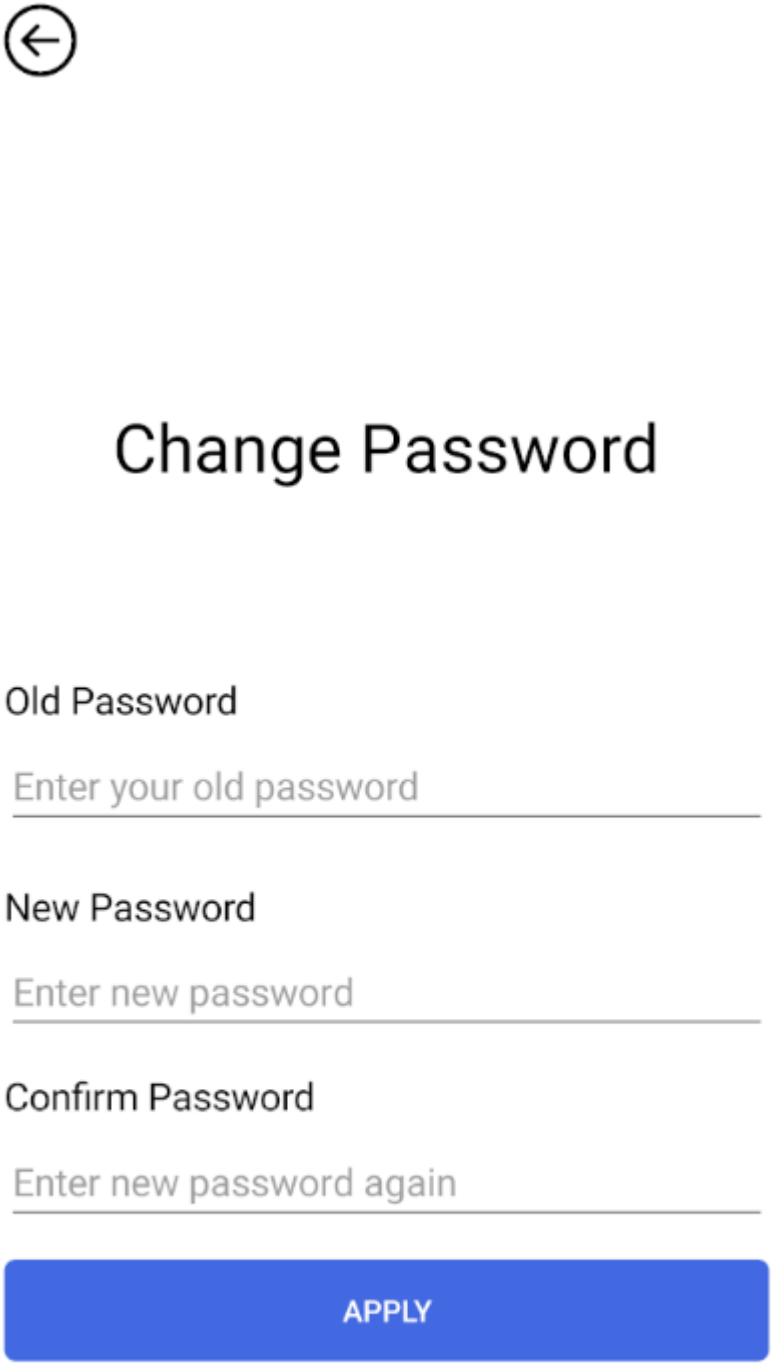

Рис.2.25. Сторінка зміни паролю

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1800;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,04;
4. годинна заробітна плата програміста – 169 грн/год;

Годинна заробітна плата C#/.Net розробника була вирахувана виходячи з даних «Української спільноти програмістів (DOU)». Середньоукраїнська заробітна плата такого розробника з досвідом роботи близько року дорівнює 1000 доларів США у місяць. При курсі валют НБУ на початок червня 2022 року один американський долар дорівнює 29,25 грн, тому середня зарплата в гривнях дорівнює 29750 грн. При восьмигодинному робочому дні (176 робочих годин в місяць в середньому) середня заробітна плата за годину буде становити 169 грн.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності – 0,8;
7. Кількість виконавців – 1;
8. вартість машино-години ЕОМ – 13 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ – витрати праці на дослідження алгоритму рішення задачі;

$t_а$ – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

$$Q = 1800 \cdot 1{,}4 \cdot (1 + 0{,}04) = 2620.8;$$

Витрати праці на вивчення опису задачі $t_и$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_и = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{2620.8 \cdot 1,3}{78 \cdot 0,8} = 54,6, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K}; \quad (3.4)$$

$$t_a = \frac{2620.8}{24 \cdot 0,8} = 136,5, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K}; \quad (3.5)$$

$$t_n = \frac{2620.8}{25 \cdot 0,8} = 131,04, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5) \cdot K}; \quad (3.6)$$

$$t_{\text{отл}} = \frac{2620.8}{5 \cdot 0,8} = 655,2, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{\text{к}} = 1,2 \cdot t_{\text{отл}}; \quad (3.7)$$

$$t_{\text{отл}}^{\text{к}} = 1,2 \cdot 655,2 = 786,24, \text{ людино-годин,}$$

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}; \qquad (3.8)$$

де $t_{\partial p}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_{\partial p} = \frac{Q}{(15...20)\cdot K}; \qquad (3.9)$$

$$t_{\partial p} = \frac{2620.8}{20 \cdot 0{,}8} = 163{,}8, \text{ людино-годин,}$$

де $t_{\partial o}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{\partial o} = 0{,}75 \cdot t_{\partial p}; \qquad (3.10)$$

$$t_{\partial o} = 0{,}75 \cdot 163{,}8 = 122{,}85, \text{ людино-годин.}$$

$$t_{\partial} = 163{,}8 + 122{,}85 = 286{,}65, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 54,6 + 136,5 + 131,04 + 655,2 + 286,64 = 1313,98, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1313,98 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1313{,}98 \cdot 169 = 222062{,}62, \text{ грн.}$$

Вартість машинного часу $З_{МВ,}$ необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot С_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 655{,}2 \cdot 13 = 8517{,}6 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 222062{,}62 + 8517{,}6 = 230580{,}22 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \qquad (3.14)$$

де $В_к$ – число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

Витрати на створення программного продукту:

$$T = \frac{1313{,}98}{1 \cdot 176} = 7{,}47 \text{ міс.}$$

Висновки: Даний кросплатформений клієнт-серверний додаток має вартість 230580,22 грн. Ймовірний очікуваний час розробки – 7,47 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Цей термін включає в себе час для дослідження та розробку алгоритму розв’язання задачі, розробку дизайну і створення документації. На розробку додатка буде витрачено 1313,98 людино-годин.

ВИСНОВКИ

В процесі виконання даної кваліфікаційної роботи був розроблений кросплатформений клієнт-серверний додаток для нотаток. Цей додаток дозволить швидко і зручно працювати з нотатками на будь-якому девайсі на ОС Android, iOS або Windows, якщо він відповідає мінімальним вимогам до системи. До того ж, застосунок дозволяє змінювати девайс без жодних втрат даних.

Актуальність даного програмного продукту визначається великим попитом на подібні розробки, що дозволяють ефективно працювати з користувальницькою інформацією на різних пристроях без втрати прогресу.

Під час виконання даного кваліфікаційної роботи були виконані наступні задачі:

- виконано аналіз предметної області;
- з’ясовані вимоги та підстави для розробки даного додатка;
- обрано раціональну архітектуру та інструменти розробки;
- описані варіанти використання додатка.

Окрім цього у кваліфікаційній роботі було визначено вартість створення даного додатка (230580,22 грн) та розраховано час, необхідний для його розробки (7,5 місяці).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jeffrey Richter. CLR via C#, 4th Edition. –2012. – 896 p.
2. Adam Freeman. Pro ASP.NET Core 3, 8th Edition. –2020. – 1109 p.
3. Can Bilgin. Mobile Development with .NET: Build cross-platform mobile applications with Xamarin.Forms 5 and ASP.NET Core 5, 2nd Edition. –2021. – 572 p.
4. Client Server Architecture. URL: https://cio-wiki.org/wiki/Client_Server_Architecture
5. An overview of HTTP. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
6. What is REST. URL: https://restfulapi.net/
7. REST API (RESTful API). URL: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API
8. Defining resource methods for RESTful applications. URL: https://www.ibm.com/docs/en/was-nd/8.5.5?topic=applications-defining-resource-methods-restful
9. Overview to ASP.NET Core. URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0
10. ACID Transactions. URL: https://databricks.com/glossary/acid-transactions
11. Repository Design Pattern In ASP.NET MVC. URL: https://www.c-sharpcorner.com/article/repository-design-pattern-in-asp-net-mvc/
12. Creating Web API With Repository Pattern And Dependency Injection. URL: https://www.c-sharpcorner.com/article/creating-web-api-with-repository-pattern-and-dependency-injection/
13. The Model-View-ViewModel Pattern. URL: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm
14. Working with JSON. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

15. What is Xamarin.Forms? URL: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms
16. FreshMvvm – A Mvvm Framework designed for Xamarin.Forms. URL: https://michaelridland.com/xamarin/freshmvvm-mvvm-framework-designed-xamarin-forms/
17. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction
18. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
19. What is Activity Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/
20. Зарплати розробників в Україні. URL: https://jobs.dou.ua/salaries/?period=2021-12&position=Junior%20SE&technology=C#/.NET
21. How to use FluentValidation in ASP.NET Core (.NET 6) URL: https://christian-schou.dk/how-to-use-fluentvalidation-in-asp-net-core/

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг Program.cs

```
using EverywhereNotes.Database;
using EverywhereNotes.Helpers;
using EverywhereNotes.Options;
using EverywhereNotes.Repositories;
using EverywhereNotes.Services;
using FluentValidation.AspNetCore;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using System.Reflection;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddHttpContextAccessor();

builder.Services.AddControllers()
    .AddFluentValidation(options =>
    {
        options.ImplicitlyValidateChildProperties = true;
        options.ImplicitlyValidateRootCollectionElements = true;
        options.RegisterValidatorsFromAssembly(Assembly.GetExecutingAssembly());
    });

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddAutoMapper(typeof(Program));

builder.Services.AddDbContext<DataContext>(options =>
{
    string mySqlConnectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
    options.UseMySql(mySqlConnectionString, ServerVersion.AutoDetect(mySqlConnectionString));
});

var jwtSettings = new JwtSettings();
builder.Configuration.Bind(nameof(jwtSettings), jwtSettings);
builder.Services.AddSingleton(jwtSettings);

builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
```

```
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
    .AddJwtBearer(x =>
    {
        x.SaveToken = true;
        x.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(jwtSettings.Secret)),
            ValidateIssuer = false,
            ValidateAudience = false,
            RequireExpirationTime = false,
            ValidateLifetime = true
        };
    });

builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "EverywhereNotes",
        Version = "v1"
    });
    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        In = ParameterLocation.Header,
        Description = "Please insert JWT with Bearer into field",
        Name = "Authorization",
        Type = SecuritySchemeType.ApiKey
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement {
   {
     new OpenApiSecurityScheme
     {
       Reference = new OpenApiReference
       {
         Type = ReferenceType.SecurityScheme,
         Id = "Bearer"
       }
      },
      new string[] { }
    }
  });
});

builder.Services.AddScoped<ICurrentUserService, CurrentUserService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<INotesService, NotesService>();

builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<INotesRepository, NotesRepository>();
```

```
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

builder.Services.AddScoped<TokenHelper>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Лістинг NotesController.cs

```
using EverywhereNotes.Contracts.Requests;
using EverywhereNotes.Extensions;
using EverywhereNotes.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EverywhereNotes.Controllers
{
    [Authorize]
    [Route("api/[controller]")]
    [ApiController]
    public class NotesController : ControllerBase
    {
        private readonly INotesService _notesService;

        public NotesController(INotesService notesService)
        {
            _notesService = notesService;
        }

        [HttpGet]
        public async Task<IActionResult> GetAsync()
        {
            var notes = await _notesService.GetByUserIdAsync();

            return notes.ToActionResult();
        }
```

```
[HttpGet("{id}")]
public async Task<IActionResult> GetByIdAsync(long id)
{
   var note = await _notesService.GetByIdAsync(id);

   return note.ToActionResult();
}

[HttpPost]
public async Task<IActionResult> PostAsync(NoteRequest request)
{
   var note = await _notesService.AddAsync(request);

   return note.ToActionResult();
}

[HttpPut("{id}")]
public async Task<IActionResult> PutAsync(long id, NoteRequest request)
{
   var note = await _notesService.UpdateAsync(id, request);

   return note.ToActionResult();
}

[HttpGet("bin")]
public async Task<IActionResult> GetBinByUserIdAsync()
{
   var notes = await _notesService.GetBinByUserIdAsync();

   return notes.ToActionResult();
}

[HttpPatch("bin/{id}")]
public async Task<IActionResult> MoveToBinAsync(long id)
{
   var note = await _notesService.MoveToBinAsync(id);

   return note.ToActionResult();
}

[HttpPatch("restore/{id}")]
public async Task<IActionResult> MoveFromBinAsync(long id)
{
   var note = await _notesService.RestoreFromBinAsync(id);

   return note.ToActionResult();
}

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteAsync(long id)
{
```

```
            var note = await _notesService.DeleteAsync(id);

            return note.ToActionResult();
        }
    }
}
```

Лістинг NotesService.cs

```
using EverywhereNotes.Contracts.Requests;
using EverywhereNotes.Contracts.Responses;
using EverywhereNotes.Models.Enums;
using EverywhereNotes.Models.ResultModel;
using EverywhereNotes.Repositories;

namespace EverywhereNotes.Services
{
    public class NotesService : INotesService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly ICurrentUserService _currentUserService;

        public NotesService(IUnitOfWork unitOfWork, ICurrentUserService currentUserService)
        {
            _unitOfWork = unitOfWork;
            _currentUserService = currentUserService;
        }

        public async Task<Result<NoteResponse>> AddAsync(NoteRequest note)
        {
            try
            {
                if (note == null)
                {
                    return Result<NoteResponse>.GetError(ErrorCode.ValidationError, "Note is null!");
                }

                var noteResponse = await _unitOfWork.NotesRepository.AddAsync(note);
                await _unitOfWork.CommitAsync();

                return Result<NoteResponse>.GetSuccess(noteResponse);
            }
            catch
            {
                _unitOfWork.Rollback();

                return Result<NoteResponse>.GetError(ErrorCode.InternalServerError, "Internal error");
            }
        }

        public async Task<Result<NoteResponse>> DeleteAsync(long id)
        {
            try
```

```
        {
            var foundNote = await _unitOfWork.NotesRepository.GetByIdAsync(id);

            if (foundNote == null)
            {
                return Result<NoteResponse>.GetError(ErrorCode.NotFound, "Note with this id was not
found!");
            }

            if (foundNote.userId != _currentUserService.UserId)
            {
                return Result<NoteResponse>.GetError(ErrorCode.Forbidden, "Only owner can reach
the note!");
            }

                        if (!foundNote.MovedToBin)
            {
                return Result<NoteResponse>.GetError(ErrorCode.Forbidden, "Note can be deleted only
from bin!");
            }

            await _unitOfWork.NotesRepository.DeleteAsync(id);
            await _unitOfWork.CommitAsync();

            return Result<NoteResponse>.GetSuccess(foundNote);
        }
        catch
        {
            _unitOfWork.Rollback();

            return Result<NoteResponse>.GetError(ErrorCode.InternalServerError, "Internal error");
        }
    }

    public async Task<Result<List<NoteResponse>>> GetBinByUserIdAsync()
    {
        var userId = _currentUserService.UserId;

        var foundNotes = await _unitOfWork.NotesRepository.GetBinByUserIdAsync(userId);

        if (foundNotes == null || foundNotes.Count == 0)
        {
            return Result<List<NoteResponse>>.GetError(ErrorCode.NotFound, "Bin is empty!");
        }

        return Result<List<NoteResponse>>.GetSuccess(foundNotes);
    }

    public async Task<Result<NoteResponse>> GetByIdAsync(long id)
    {
        var foundNote = await _unitOfWork.NotesRepository.GetByIdAsync(id);
```

```
            if (foundNote == null)
            {
                return Result<NoteResponse>.GetError(ErrorCode.NotFound, "Note with this id was not
found!");
            }

            if (foundNote.userId != _currentUserService.UserId)
            {
                return Result<NoteResponse>.GetError(ErrorCode.Forbidden, "Only owner can reach the
note!");
            }

            return Result<NoteResponse>.GetSuccess(foundNote);
        }

        public async Task<Result<List<NoteResponse>>> GetByUserIdAsync()
        {
            var userId = _currentUserService.UserId;

            var foundNotes = await _unitOfWork.NotesRepository.GetByUserIdAsync(userId);

            if (foundNotes == null || foundNotes.Count == 0)
            {
                return Result<List<NoteResponse>>.GetError(ErrorCode.NotFound, "Notes with this user
id were not found!");
            }

            return Result<List<NoteResponse>>.GetSuccess(foundNotes);
        }

        public async Task<Result<NoteResponse>> MoveToBinAsync(long id)
        {
            try
            {
                var foundNote = await _unitOfWork.NotesRepository.GetByIdAsync(id);

                if (foundNote == null)
                {
                    return Result<NoteResponse>.GetError(ErrorCode.NotFound, "Note with this id was not
found!");
                }

                if (foundNote.userId != _currentUserService.UserId)
                {
                    return Result<NoteResponse>.GetError(ErrorCode.Forbidden, "Only owner can reach
the note!");
                }

                if (foundNote.MovedToBin)
                {
                    return Result<NoteResponse>.GetError(ErrorCode.Conflict, "Note with this id is already
in bin!");
```

```
            }

            foundNote.MovedToBin = true;

            await _unitOfWork.NotesRepository.UpdateAsync(foundNote);
            await _unitOfWork.CommitAsync();

            return Result<NoteResponse>.GetSuccess(foundNote);
        }
        catch
        {
            _unitOfWork.Rollback();

            return Result<NoteResponse>.GetError(ErrorCode.InternalServerError, "Internal error");
        }
    }

    public async Task<Result<NoteResponse>> RestoreFromBinAsync(long id)
    {
        try
        {
            var foundNote = await _unitOfWork.NotesRepository.GetByIdAsync(id);

            if (foundNote == null)
            {
                return Result<NoteResponse>.GetError(ErrorCode.NotFound, "Note with this id was not
found!");
            }

            if (foundNote.userId != _currentUserService.UserId)
            {
                return Result<NoteResponse>.GetError(ErrorCode.Forbidden, "Only owner can reach
the note!");
            }

            if (!foundNote.MovedToBin)
            {
                return Result<NoteResponse>.GetError(ErrorCode.Conflict, "Note with this id is not in
bin!");
            }

            foundNote.MovedToBin = false;

            await _unitOfWork.NotesRepository.UpdateAsync(foundNote);
            await _unitOfWork.CommitAsync();

            return Result<NoteResponse>.GetSuccess(foundNote);
        }
        catch
        {
            _unitOfWork.Rollback();
```

```
            return Result<NoteResponse>.GetError(ErrorCode.InternalServerError, "Internal error");
        }
    }

    public async Task<Result<NoteResponse>> UpdateAsync(long id, NoteRequest note)
    {
        try
        {
            if (note == null)
            {
                return Result<NoteResponse>.GetError(ErrorCode.NotFound, "Note request is null!");
            }

            var foundNote = await _unitOfWork.NotesRepository.GetByIdAsync(id);

            if (foundNote == null)
            {
                return Result<NoteResponse>.GetError(ErrorCode.NotFound, "Note was not found!");
            }

            if (foundNote.userId != _currentUserService.UserId)
            {
                return Result<NoteResponse>.GetError(ErrorCode.Forbidden, "Only owner can reach
the note!");
            }

            foundNote.Title = note.Title;
            foundNote.Content = note.Content;
            foundNote.Color = note.Color;

            await _unitOfWork.NotesRepository.UpdateAsync(foundNote);
            await _unitOfWork.CommitAsync();

            return Result<NoteResponse>.GetSuccess(foundNote);
        }
        catch
        {
            _unitOfWork.Rollback();

            return Result<NoteResponse>.GetError(ErrorCode.InternalServerError, "Internal error");
        }
    }
  }
}
```

Лістинг NotesRepository.cs

```
using AutoMapper;
using EverywhereNotes.Contracts.Requests;
using EverywhereNotes.Contracts.Responses;
using EverywhereNotes.Database;
using EverywhereNotes.Models.Entities;
using EverywhereNotes.Services;
```

```
using Microsoft.EntityFrameworkCore;

namespace EverywhereNotes.Repositories
{
   public class NotesRepository : INotesRepository
   {
      private DataContext _dataContext;
      private readonly IMapper _mapper;
      private readonly ICurrentUserService _currentUserService;

      public NotesRepository(DataContext dataContext, ICurrentUserService currentUserService,
IMapper mapper)
      {
         _dataContext = dataContext;
         _currentUserService = currentUserService;
         _mapper = mapper;
      }

      public async Task<NoteResponse> AddAsync(NoteRequest note)
      {
         var createdNote = new Note
         {
            Title = note.Title,
            Content = note.Content,
            Color = note.Color,
            CreationDateTime = DateTime.Now,
            userId = _currentUserService.UserId
         };

         await _dataContext.Notes.AddAsync(createdNote);

         return _mapper.Map<NoteResponse>(createdNote);
      }

      public async Task DeleteAsync(long id)
      {
         var noteToDelete = await _dataContext.Notes.FirstOrDefaultAsync(x => x.Id == id);

         if (noteToDelete != null)
         {
            _dataContext.Notes.Remove(noteToDelete);
         }
      }

      public async Task<List<NoteResponse>> GetBinByUserIdAsync(long userId)
      {
         var notes = await _dataContext.Notes.Where(x => x.userId == userId &&
x.MovedToBin).ToListAsync();

         return _mapper.Map<List<NoteResponse>>(notes);
      }
```

```
        public async Task<NoteResponse?> GetByIdAsync(long id)
        {
            var note = await _dataContext.Notes.FirstOrDefaultAsync(x => x.Id == id);

            return _mapper.Map<NoteResponse>(note);
        }

        public async Task<List<NoteResponse>> GetByUserIdAsync(long userId)
        {
            var notes = await _dataContext.Notes.Where(x => x.userId == userId &&
!x.MovedToBin).ToListAsync();

            return _mapper.Map<List<NoteResponse>>(notes);
        }

        public async Task UpdateAsync(NoteResponse note)
        {
            var noteToUpdate = await _dataContext.Notes.FirstOrDefaultAsync(x => x.Id == note.Id);

            if (noteToUpdate == null)
            {
                return;
            }

            noteToUpdate.Title = note.Title;
            noteToUpdate.Content = note.Content;
            noteToUpdate.Color = note.Color;
            noteToUpdate.MovedToBin = note.MovedToBin;

            noteToUpdate.LastUpdateDateTime = DateTime.Now;

            _dataContext.Notes.Update(noteToUpdate);
        }
    }
}
```

Лістинг PasswordHelper.cs

```
using System.Text;
using System.Security.Cryptography;

namespace EverywhereNotes.Helpers
{
    public static class PasswordHelper
    {
        /// <summary>
        /// At least eight characters, at least one uppercase letter, one lowercase letter one number and
special character
        /// </summary>
        public const string PasswordPattern = @"^(?=.{8,})(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$_!%^&-
+()*=])(?=.*[0-9]).*$";

        public static string GenerateSalt()
```

```
        {
            return Guid.NewGuid().ToString();
        }

        public static string HashPassword(string password, string salt)
        {
            byte[] data = Encoding.Default.GetBytes(password + salt);
            using SHA256 mySHA256 = SHA256.Create();
            var result = mySHA256.ComputeHash(data);

            return BitConverter.ToString(result).Replace("-", "").ToLower();
        }
    }
}
```

Лістинг TokenHelper.cs

```
using EverywhereNotes.Models.Entities;
using EverywhereNotes.Options;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace EverywhereNotes.Helpers
{
    public class TokenHelper
    {
        private readonly JwtSettings _jwtSettings;

        public TokenHelper(JwtSettings jwtSettings)
        {
            _jwtSettings = jwtSettings;
        }

        public string GenerateToken(User user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_jwtSettings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                    new Claim(JwtRegisteredClaimNames.Sub, user.Email),
                    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
                    new Claim("Email", user.Email),
                    new Claim("Id", user.Id.ToString())
                }),
                Expires = DateTime.UtcNow.AddDays(5),
                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
            };
```

```
            var token = tokenHandler.CreateToken(tokenDescriptor);

            return tokenHandler.WriteToken(token);
        }
    }
}
```

Лістинг App.xaml.cs

```
using Xamarin.Forms;
using FreshMvvm;
using EverywhereNotesClient.ViewModels;
using EverywhereNotesClient.Services;

namespace EverywhereNotesClient
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            FreshIOC.Container.Register<IHttpService, HttpService>();
            FreshIOC.Container.Register<ILoginService, LoginService>();

            HttpClientProvider.InitializeHttpClient();

            var loginPage = FreshPageModelResolver.ResolvePageModel<LoginViewModel>();
            var loginNavigation = new FreshNavigationContainer(loginPage,
NavigationStacks.LoginNavigationStack);

            if (Device.Idiom == TargetIdiom.Phone)
            {
                var masterNavigation = new
FreshMasterDetailNavigationContainer(NavigationStacks.MainAppStack);
                masterNavigation.Init("Menu");
                masterNavigation.AddPage<MainViewModel>("Notes", null);
                masterNavigation.AddPage<BinViewModel>("Bin", null);
                masterNavigation.AddPage<AccountViewModel>("Account", null);
                MainPage = masterNavigation;
            }
            else
            {
                var masterNavigation = new
FreshTabbedNavigationContainer(NavigationStacks.MainAppStack);
                masterNavigation.AddTab<MainViewModel>("Notes", null);
                masterNavigation.AddTab<BinViewModel>("Bin", null);
                masterNavigation.AddTab<AccountViewModel>("Account", null);
                MainPage = masterNavigation;
            }
        }
```

```
        protected override void OnStart()
        {
        }

        protected override void OnSleep()
        {
        }

        protected override void OnResume()
        {
        }
    }

    public static class NavigationStacks
    {
        public static string LoginNavigationStack = "LoginNavigationStack";
        public static string MainAppStack = "MainAppStack";
    }
}
```

Лістинг HttpService.cs

```
using EverywhereNotesClient.Models.DTO;
using EverywhereNotesClient.Models.Requests;
using EverywhereNotesClient.Models.Responses;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace EverywhereNotesClient.Services
{
    public class HttpService : IHttpService
    {
        public async Task<Result> RegisterAsync(UserRegistrationRequest request)
        {
            var result = new Result();
            var json = JsonConvert.SerializeObject(request);
            var content = new StringContent(json, Encoding.UTF8, "application/json");

            try
            {
                var response = await HttpClientProvider.Client.PostAsync("users/register", content);

                result.StatusCode = response.StatusCode;

                if (response.IsSuccessStatusCode)
                {
                    var responseString = await response.Content.ReadAsStringAsync();
                    var deserializedObject =
JsonConvert.DeserializeObject<AuthSuccessResponse>(responseString);
```

```
                result.Data = deserializedObject;
            }
        }
        catch (Exception ex)
        {
        }

        return result;
    }

    public async Task<Result> AuthorizeAsync(UserAuthorizationRequest request)
    {
        var result = new Result();
        var json = JsonConvert.SerializeObject(request);
        var content = new StringContent(json, Encoding.UTF8, "application/json");

        try
        {
            var response = await HttpClientProvider.Client.PostAsync("users/authorize", content);

            result.StatusCode = response.StatusCode;

            if (response.IsSuccessStatusCode)
            {
                var responseString = await response.Content.ReadAsStringAsync();
                var deserializedObject =
JsonConvert.DeserializeObject<AuthSuccessResponse>(responseString);
                result.Data = deserializedObject;
            }
        }
        catch (Exception ex)
        {
        }

        return result;
    }

    public async Task<Result> ChangePasswordAsync(ChangePasswordRequest request)
    {
        var result = new Result();
        var json = JsonConvert.SerializeObject(request);
        var content = new StringContent(json, Encoding.UTF8, "application/json");

        try
        {
            var response = await HttpClientProvider.Client.PutAsync("users/password", content);

            result.StatusCode = response.StatusCode;
        }
        catch (Exception ex)
        {
        }
```

```
    return result;
}

public async Task<Result> GetNotesAsync()
{
    var result = new Result();

    try
    {
        var response = await HttpClientProvider.Client.GetAsync("notes");

        result.StatusCode = response.StatusCode;

        if (response.IsSuccessStatusCode)
        {
            var responseString = await response.Content.ReadAsStringAsync();
            var deserializedObjects = JsonConvert.DeserializeObject<List<Note>>(responseString);
            result.Data = deserializedObjects;
        }
    }
    catch (Exception ex)
    {
    }

    return result;
}

public async Task<Result> GetNoteByIdAsync(long id)
{
    var result = new Result();

    try
    {
        var response = await HttpClientProvider.Client.GetAsync($"notes/{id}");

        result.StatusCode = response.StatusCode;

        if (response.IsSuccessStatusCode)
        {
            var responseString = await response.Content.ReadAsStringAsync();
            var deserializedObject = JsonConvert.DeserializeObject<Note>(responseString);
            result.Data = deserializedObject;
        }
    }
    catch (Exception ex)
    {
    }

    return result;
}
```

```
public async Task<Result> CreateNoteAsync(NoteRequest request)
{
    var result = new Result();
    var json = JsonConvert.SerializeObject(request);
    var content = new StringContent(json, Encoding.UTF8, "application/json");

    try
    {
        var response = await HttpClientProvider.Client.PostAsync("notes", content);

        result.StatusCode = response.StatusCode;

        if (response.IsSuccessStatusCode)
        {
            var responseString = await response.Content.ReadAsStringAsync();
            var deserializedObject = JsonConvert.DeserializeObject<Note>(responseString);
            result.Data = deserializedObject;
        }
    }
    catch (Exception ex)
    {
    }

    return result;
}

public async Task<Result> UpdateNoteAsync(long id, NoteRequest request)
{
    var result = new Result();
    var json = JsonConvert.SerializeObject(request);
    var content = new StringContent(json, Encoding.UTF8, "application/json");

    try
    {
        var response = await HttpClientProvider.Client.PutAsync($"notes/{id}", content);

        result.StatusCode = response.StatusCode;

        if (response.IsSuccessStatusCode)
        {
            var responseString = await response.Content.ReadAsStringAsync();
            var deserializedObject = JsonConvert.DeserializeObject<Note>(responseString);
            result.Data = deserializedObject;
        }
    }
    catch (Exception ex)
    {
    }

    return result;
}
```

```
        public async Task<Result> GetNotesFromBinAsync()
        {
            var result = new Result();

            try
            {
                var response = await HttpClientProvider.Client.GetAsync("notes/bin");

                result.StatusCode = response.StatusCode;

                if (response.IsSuccessStatusCode)
                {
                    var responseString = await response.Content.ReadAsStringAsync();
                    var deserializedObjects = JsonConvert.DeserializeObject<List<Note>>(responseString);
                    result.Data = deserializedObjects;
                }
            }
            catch (Exception ex)
            {
            }

            return result;
        }

        public async Task<Result> MoveToBinAsync(long id)
        {
            var result = new Result();

            try
            {
                var httpRequest = new HttpRequestMessage(new HttpMethod("PATCH"),
$"notes/bin/{id}");

                var response = await HttpClientProvider.Client.SendAsync(httpRequest);

                result.StatusCode = response.StatusCode;

                if (response.IsSuccessStatusCode)
                {
                    var responseString = await response.Content.ReadAsStringAsync();
                    var deserializedObject = JsonConvert.DeserializeObject<Note>(responseString);
                    result.Data = deserializedObject;
                }
            }
            catch (Exception ex)
            {
            }

            return result;
        }

        public async Task<Result> RestoreFromBinAsync(long id)
```

```
        {
            var result = new Result();

            try
            {
                var httpRequest = new HttpRequestMessage(new HttpMethod("PATCH"),
$"notes/restore/{id}");

                var response = await HttpClientProvider.Client.SendAsync(httpRequest);

                result.StatusCode = response.StatusCode;

                if (response.IsSuccessStatusCode)
                {
                    var responseString = await response.Content.ReadAsStringAsync();
                    var deserializedObject = JsonConvert.DeserializeObject<Note>(responseString);
                    result.Data = deserializedObject;
                }
            }
            catch (Exception ex)
            {
            }

            return result;
        }

        public async Task<Result> DeleteNoteAsync(long id)
        {
            var result = new Result();

            try
            {
                var response = await HttpClientProvider.Client.DeleteAsync($"notes/{id}");

                result.StatusCode = response.StatusCode;

                if (response.IsSuccessStatusCode)
                {
                    var responseString = await response.Content.ReadAsStringAsync();
                    var deserializedObject = JsonConvert.DeserializeObject<Note>(responseString);
                    result.Data = deserializedObject;
                }
            }
            catch (Exception ex)
            {
            }

            return result;
        }
    }
}
```

Лістинг MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="EverywhereNotesClient.Pages.MainPage"
             BackgroundColor="White"
             Title="Notes">
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="New" IconImageSource="Add.png" Command="{Binding
CreateNoteCommand}"/>
    </ContentPage.ToolbarItems>

    <StackLayout>
        <CollectionView ItemsSource="{Binding Notes}"
                        SelectionMode="Single"
                        SelectedItem="{Binding SelectedItem, Mode=OneWayToSource}"
                        SelectionChangedCommand="{Binding ItemSelectedCommand}"
                        ItemSizingStrategy="MeasureAllItems">
            <CollectionView.ItemsLayout>
                <GridItemsLayout Orientation="Vertical" VerticalItemSpacing="4"></GridItemsLayout>
            </CollectionView.ItemsLayout>
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <Grid Padding="16, 0, 16, 8"
                          BackgroundColor="{Binding DrawingColor}">
                        <Grid.RowDefinitions>
                            <RowDefinition Height="Auto" />
                            <RowDefinition Height="Auto" />
                            <RowDefinition Height="Auto" />
                            <RowDefinition Height="Auto" />
                        </Grid.RowDefinitions>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="Auto" />
                        </Grid.ColumnDefinitions>
                    <Label Grid.Row="1"
                      IsVisible="{Binding HasTitle}"
                      Text="{Binding Title}"
                      FontAttributes="Bold"
                      TextColor="White"/>
                    <Label Grid.Row="2"
                      Text="{Binding Content}"
                      FontAttributes="Italic"
                      TextColor="White"/>
                    <StackLayout Orientation="Horizontal"
                                 HorizontalOptions="Start"
                                 Grid.Row="3">
                        <Label
                          IsVisible="{Binding HasUpdateDateTime}"
                          Text="Updated:"
                          FontAttributes="None"
                          TextColor="White"
                          HorizontalTextAlignment="End"/>
```

```
                <Label
                    IsVisible="{Binding HasUpdateDateTime}"
                    Text="{Binding LastUpdateDateTime}"
                    FontAttributes="None"
                    TextColor="White"
                    HorizontalTextAlignment="Start"/>
            </StackLayout>
            <StackLayout Orientation="Horizontal"
                         HorizontalOptions="Start"
                         Grid.Row="4">
                <Label
                    Text="Created:"
                    FontAttributes="None"
                    TextColor="White"
                    HorizontalTextAlignment="End"/>
                <Label
                    Text="{Binding CreationDateTime}"
                    FontAttributes="None"
                    TextColor="White"
                    HorizontalTextAlignment="Start"/>
            </StackLayout>

            </Grid>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
</StackLayout>

</ContentPage>
```

Лістинг MainViewModel.cs

```
using EverywhereNotesClient.Models;
using EverywhereNotesClient.Models.DTO;
using EverywhereNotesClient.Services;
using FreshMvvm;
using MvvmHelpers.Commands;
using MvvmHelpers.Interfaces;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Net;
using System.Threading.Tasks;

namespace EverywhereNotesClient.ViewModels
{
    public class MainViewModel : FreshBasePageModel
    {
        private readonly ILoginService _loginService;
        private readonly IHttpService _httpService;

        public MainViewModel(ILoginService loginService, IHttpService httpService)
```

```
{
    _loginService = loginService;
    _httpService = httpService;

    Notes = new ObservableCollection<NoteItemModel>();

    ItemSelectedCommand = new AsyncCommand(ItemSelectedAsync);
    CreateNoteCommand = new AsyncCommand(CreateNoteAsync);
}

public ObservableCollection<NoteItemModel> Notes { get; set; }

public NoteItemModel SelectedItem { get; set; }

public IAsyncCommand ItemSelectedCommand { get; }

public IAsyncCommand CreateNoteCommand { get; }

protected override async void ViewIsAppearing(object sender, EventArgs e)
{
    base.ViewIsAppearing(sender, e);

    var isLoggedIn = await _loginService.LoginAsync();

    if (isLoggedIn)
    {
        await GetNotesAsync();
    }
    else
    {
        CoreMethods.SwitchOutRootNavigation(NavigationStacks.LoginNavigationStack);
    }
}

private async Task GetNotesAsync()
{
    var result = await _httpService.GetNotesAsync();

    if (result.StatusCode == HttpStatusCode.OK)
    {
        var notes = (List<Note>)result.Data;
        notes.Reverse();

        Notes.Clear();

        foreach (var note in notes)
        {
            var noteItem = new NoteItemModel(note);
            Notes.Add(noteItem);
        }
    }
    else if (result.StatusCode == HttpStatusCode.NotFound)
```

```
            {
                Notes.Clear();
            }
            else
            {
                _loginService.Logout();
                CoreMethods.SwitchOutRootNavigation(NavigationStacks.LoginNavigationStack);
            }
        }

        private async Task ItemSelectedAsync()
        {
            if (SelectedItem == null)
            {
                return;
            }

            await CoreMethods.PushPageModel<NoteDetailsViewModel>(SelectedItem, true);
        }

        private async Task CreateNoteAsync()
        {
            await CoreMethods.PushPageModel<NoteDetailsViewModel>(null, true);
        }
    }
}
```

Лістинг NoteDetailsPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="EverywhereNotesClient.Pages.NoteDetailsPage"
             BackgroundColor="{Binding Color}">
    <NavigationPage.TitleView>
        <StackLayout Orientation="Horizontal"
                     HorizontalOptions="FillAndExpand"
                     VerticalOptions="Center">
            <Image Source="Back.png"
                   HeightRequest="34"
                   WidthRequest="34"
                   Margin="0, 0, 8, 0"
                   BackgroundColor="Transparent"
                   HorizontalOptions="Start">
                <Image.GestureRecognizers>
                    <TapGestureRecognizer
                Command="{Binding CloseCommand}"
                NumberOfTapsRequired="1" />
                </Image.GestureRecognizers>
            </Image>
            <StackLayout Orientation="Horizontal"
                         VerticalOptions="CenterAndExpand"
                         HorizontalOptions="EndAndExpand">
```

```
            <Image Source="Colors.png"
              HeightRequest="34"
              WidthRequest="34"
              Margin="8, 0, 8, 0"
              BackgroundColor="Transparent">
               <Image.GestureRecognizers>
                  <TapGestureRecognizer
            Command="{Binding PickColorCommand}"
            NumberOfTapsRequired="1" />
               </Image.GestureRecognizers>
            </Image>
            <Image Source="Options.png"
              IsVisible="{Binding IsEditing}"
              HeightRequest="34"
              WidthRequest="34"
              Margin="8, 0, 8, 0"
              BackgroundColor="Transparent">
               <Image.GestureRecognizers>
                  <TapGestureRecognizer
               Command="{Binding ShowOptionsCommand}"
               NumberOfTapsRequired="1" />
               </Image.GestureRecognizers>
            </Image>
         </StackLayout>
      </StackLayout>
   </NavigationPage.TitleView>
   <ContentPage.Content>
      <StackLayout BackgroundColor="Transparent">
         <Entry Margin="16, 16, 16, 8"
               Placeholder="Title"
               Text="{Binding Title}"
               BackgroundColor="Transparent"
               FontSize="18"
               PlaceholderColor="Gray"
               TextColor="White"/>
         <Editor
               Margin="16, 0, 16, 16"
               Placeholder="Note details"
               Text="{Binding Content}"
               BackgroundColor="Transparent"
               FontSize="16"
               AutoSize="TextChanges"
               PlaceholderColor="Gray"
               TextColor="White"/>
      </StackLayout>
   </ContentPage.Content>
</ContentPage>
```

Лістинг NoteDetailsViewModel.cs

```
using EverywhereNotesClient.Extensions;
using EverywhereNotesClient.Models;
using EverywhereNotesClient.Models.Enums;
```

```
using EverywhereNotesClient.Models.Requests;
using EverywhereNotesClient.Services;
using FreshMvvm;
using MvvmHelpers.Commands;
using MvvmHelpers.Interfaces;
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace EverywhereNotesClient.ViewModels
{
   public class NoteDetailsViewModel : FreshBasePageModel
   {
      private readonly IHttpService _httpService;
      private NoteColor _noteColor;
      private bool _isEditing;
      private NoteItemModel _item;

      public NoteDetailsViewModel(IHttpService httpService)
      {
         _httpService = httpService;

         PickColorCommand = new AsyncCommand(PickColorAsync);
         CloseCommand = new AsyncCommand(CloseAsync);
         ShowOptionsCommand = new AsyncCommand(ShowOptionsAsync);
      }

      public string Title { get; set; }

      public string Content { get; set; }

      public Color Color { get; set; }

      public bool IsEditing => _isEditing;

      public NoteColor NoteColor
      {
         get => _noteColor;
         set
         {
            _noteColor = value;
            Color = _noteColor.ConvertToColor();
         }
      }

      public IAsyncCommand PickColorCommand { get; }

      public IAsyncCommand CloseCommand { get; }

      public IAsyncCommand ShowOptionsCommand { get; }

      public override void Init(object initData)
```

```
{
    if (initData is NoteItemModel item)
    {
        _item = item;
        _isEditing = true;
        Title = _item.Title;
        Content = _item.Content;
        _noteColor = _item.Color;
        Color = _item.DrawingColor;
    }
}

public override void ReverseInit(object returnedData)
{
    NoteColor = (NoteColor)returnedData;
}

protected override void ViewIsAppearing(object sender, EventArgs e)
{
    base.ViewIsAppearing(sender, e);

    if (!_isEditing)
    {
        Color = NoteColor.ConvertToColor();
        Title = string.Empty;
    }
}

private async Task SaveNoteAsync()
{
    if (string.IsNullOrEmpty(Content))
    {
        return;
    }

    var request = new NoteRequest
    {
        Title = Title,
        Content = Content,
        Color = NoteColor
    };

    if (_isEditing)
    {
        if (request.Title == _item.Title &&
        request.Content == _item.Content &&
        request.Color == _item.Color)
        {
            return;
        }

        await _httpService.UpdateNoteAsync(_item.Id, request);
```

```
            }
            else
            {
                await _httpService.CreateNoteAsync(request);
            }
        }

        private async Task PickColorAsync()
        {
            await CoreMethods.PushPageModel<ColorPickerViewModel>(_noteColor, true);
        }

        private async Task ShowOptionsAsync()
        {
            var result = await CoreMethods.DisplayActionSheet("Options", "Cancel", null, new[] {
"Delete", "Reset" });

            switch (result)
            {
                case "Delete":
                    if (_isEditing)
                    {
                        await _httpService.MoveToBinAsync(_item.Id);
                        await CoreMethods.PopPageModel(true);
                    }
                    else
                    {
                        await CoreMethods.PopPageModel(true);
                    }
                    break;
                case "Reset":
                    Title = _item.Title;
                    Content = _item.Content;
                    NoteColor = _item.Color;
                    break;
            }
        }

        private async Task CloseAsync()
        {
            await SaveNoteAsync();

            await CoreMethods.PopPageModel(true);
        }
    }
}
```

Лістинг NoteItemModel.cs

```
using EverywhereNotesClient.Extensions;
using EverywhereNotesClient.Models.DTO;
using EverywhereNotesClient.Models.Enums;
using System;
```

```
using Xamarin.Forms;

namespace EverywhereNotesClient.Models
{
   [PropertyChanged.AddINotifyPropertyChangedInterface]
   public class NoteItemModel
   {
      private Note _note;

      public NoteItemModel(Note note)
      {
         _note = note;
      }

      public long Id => _note.Id;

      public string Title => _note.Title;

      public string Content => _note.Content;

      public NoteColor Color => _note.Color;

      public Color DrawingColor => Color.ConvertToColor();

      public DateTime CreationDateTime => _note.CreationDateTime;

      public DateTime? LastUpdateDateTime => _note.LastUpdateDateTime;

      public bool HasTitle => !string.IsNullOrEmpty(Title);

      public bool HasUpdateDateTime => LastUpdateDateTime != null;
   }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна робота Фесак.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Фесак.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Фесак.rar	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Фесак.ppt	Презентація кваліфікаційної роботи.

ВІДГУК

на кваліфікаційну роботу бакалавра
на тему:
"Розробка кросплатформеного клієнт-серверного додатка для нотаток"
студента групи 121-18-1 Фесака Дмитра Володимировича

В результаті виконання даної кваліфікаційної роботи було розроблено кросплатформений клієнт-серверний додаток для нотаток.

Актуальність даного програмного продукту визначається великим попитом на подібні розробки, що дозволяють ефективно працювати з користувальницькою інформацією на різних пристроях без втрати прогресу.

Розроблене програмне забезпечення може бути використане як основа для більш комплексних кросплатформених клієнт-серверних додатків в багатьох сферах. Для реалізації програмного забезпечення була використана мова програмування C# та фреймворки ASP.NET Core Web API, EntityFramework, Xamarin.Forms та FreshMVVM.

Працездатність представленої програми підтверджена налагоджувальними випробуваннями та тестуванням програми.

В економічному розділі визначено трудомісткість розробленого додатку, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра за напрямом підготовки 121 «Інженерія програмного забезпечення».

Оформлення пояснювальної записки до кваліфікайційної роботи проекту виконано відповідно до стандартів на програмну документацію.

Кваліфікаційна робота виконана самостійно та заслуговує оцінки 95 балів «відмінно», а студент Фесак Дмитро Володимирович присвоєння йому кваліфікації бакалавра за спеціальністю «фахівець в галузі інформаційних технологій».

Керівник дипломного проекту
доцент каф. ПЗКС, к.т.н. **С.Д. Приходченко**

РЕЦЕНЗІЯ
на кваліфікаційну роботу бакалавра
на тему:
"Розробка кросплатформеного клієнт-серверного додатка для нотаток"
студента групи 121-18-1 Фесака Дмитра Володимировича

Кваліфікаційна робота на тему «розробка кросплатформеного клієнт-серверного додатка для нотаток» виконана в повному обсязі, відповідно до технічного завдання.

Мета кваліфікаційної роботи: створення кросплатформеного додатка, який дозволятиме керувати синхронізованими нотатками, на різних пристроях.

У пояснювальній записці розглянуто необхідність створення і сфера застосування розробленого програмного забезпечення, виконано постановку завдання, опис вхідних і вихідних даних, розроблено логічну структуру програми, розроблено інформаційне забезпечення системи, наведені загальні відомості про додаток та його функціональне призначення, зазначені використовувані технічні засоби, визначені джерела, використані при розробці.

Для реалізації програмного забезпечення була використана мова програмування C# та фреймворки ASP.NET Core Web API, EntityFramework, Xamarin.Forms та FreshMVVM.

Вважаю завдання і зміст кваліфікаційної роботи відповідним для перевірки ступеня підготовленості Фесака Д. В. за напрямом 121 «Інженерія програмного забезпечення».

Список літератури, наведений в роботі, налічує більше 20 джерел, що свідчить про вміння автора працювати з літературою та іншими джерелами інформації. Якість оформлення кваліфікаційної роботи можна визнати задовільною, вона супроводжується достатньою кількістю малюнків. В роботі присутні заключні висновки. Працездатність цього програмного забезпечення підтверджується експлуатаційними випробуваннями.

Рівень теоретичної та практичної підготовки автора, логіка і стиль викладу матеріалу в цілому відповідає кваліфікаційним вимогам.

Кваліфікаційна робота виконана самостійно та заслуговує оцінки 95 балів «відмінно», а студент Фесак Дмитро Володимирович присвоєння йому кваліфікації бакалавра за спеціальністю «Інженерія програмного забезпечення».

Рецензент дипломного проекту