

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
ПЛАНУВАННЯ ТА ОРГАНІЗАЦІЇ СПРАВ PLANIFY
PLANIFY SOFTWARE DEVELOPMENT FOR PLANNING
AND ORGANIZING THE ACTIVITY

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

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

НІКУЛІНА А.С.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

НІКУЛІНОЇ Анастасії Сергіївни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення для планування та організації справ Planify.
Planify Software Development for Planning and Organizing the Activity.

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “24” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір мови програмування та інших технологій розробки. 3. Розробка архітектури програми. 4. Розробка прототипів інтерфейсів системи. 5. Розробка програми. 6. Тестування та експериментальне дослідження програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації (17 слайдів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КАПЛІЄНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

 Анастасія НІКУЛІНА
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

Тетяна КАПЛІЄНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 91 с., 11 табл., 43 рис., 2 дод., 20 джерел.

ПЛАНУВАЛЬНИК, МОБІЛЬНИЙ ОРГАНАЙЗЕР,
ВЕБЗАСТОСУНОК, МОБІЛЬНИЙ ЗАСТОСУНОК, API, ASP.NET, БАЗА
ДАНИХ, PATTERN MVVM

Об'єкт дослідження – мобільний та вебзастосунки, призначені для допомоги користувачам у веденні та організації своїх завдань, планів та подій.

Предмет дослідження – розробка застосунків, що забезпечать користувачів зручним інструментом для планування.

Мета роботи – створення надійного, зручного та функціонального програмного продукту, який допоможе користувачам ефективно керувати своїм часом та завданнями, а також створення інтуїтивно зрозумілого інтерфейсу, який забезпечить легку навігацію та зручність використання для різних категорій користувачів, таких як звичайний користувач чи студент.

Матеріали, методи та технічні засоби: мова програмування C#, середовище розробки Visual Studio, фреймворк ASP.NET, СКБД Entity Framework SQLite. Використано різноманітні методи дослідження, включаючи емпіричні, такі як спостереження, опис, вимірювання та експеримент, а також теоретичні, такі як аналіз, порівняння, формалізація та моделювання. Кожен з цих методів сприяв розробці підходу до створення застосунку.

Результати. Розроблено мобільний застосунок для планування задач, який буде зв'язуватись через API із вебзастосунком та базою даних

Галузь використання – планування справ та зберігання інформації в хмарному сховищі за допомогою мобільного застосунку.

ABSTRACT

Explanatory Note for Bachelor's Qualification Thesis: 91 pages, 11 tables, 43 figures, 2 appendices, 20 sources.

PLANNER, MOBILE ORGANIZER, WEB APPLICATION, MOBILE APPLICATION, API, ASP.NET, DATABASE, PATTERN MVVM

Subject of Research: Mobile and web applications designed to help users manage and organize their tasks, plans, and events.

Object of Research: Development of such applications that will provide users with a convenient tool for planning.

Objective of the Work: To create a reliable, convenient, and functional software product that will help users effectively manage their time and tasks, as well as to create an intuitive interface that will ensure easy navigation and usability for different categories of users, such as ordinary users or students.

Various research methods were used, including empirical methods such as observation, description, measurement, and experiment, as well as theoretical methods such as analysis, comparison, formalization, and modeling. Each of these methods contributed to the development of an approach to creating the application.

The following materials, methods, and tools were used for the implementation: programming language C#, Visual Studio development environment, ASP.NET framework, Entity Framework SQLite DBMS.

The mobile application for task planning has been developed, which will interact via API with the web application and database.

Field of application - scheduling tasks and storing information in a cloud storage via the mobile application.

Results: A mobile task planning application was developed that will be connected to a web application and database via an API.

Scope of Application: Task planning and information storage in the cloud storage using a mobile application.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	11
1.1 Огляд існуючих програмних засобів, що вирішують аналогічні завдання	11
1.2 Висновки за розділом 1	14
2 ВИБІР МОВИ ПРОГРАМУВАННЯ ТА ІНШИХ ТЕХНОЛОГІЙ РОЗРОБКИ	15
2.1 Вибір інструментарію розробки програмного забезпечення	15
2.2 Вибір інструментарію розробки мобільного застосунку	19
2.3 Вибір інструментарію розробки вебзастосунку	20
2.4 Вибір інструментарію для роботи з базою даних	21
2.5 Вибір шаблону проєктування	22
2.6 Висновки за розділом 2	26
3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ.....	28
3.1 Структурна схема програми	28
3.2 Опис модулів розроблюваної системи	29
3.3 Архітектура системи Web API.....	30
3.4 Функціональна схема програми	31
3.5 Висновки до розділу 3	31
4 РОЗРОБКА ПРОТОТИПІВ ІНТЕРФЕЙСІВ СИСТЕМИ	32
4.1 Мета та вибір інструменту для прототипування	32
4.2 Результати прототипування	33
4.3 Висновки за розділом 4	36
5 РОЗРОБКА ПРОГРАМИ	37
5.1 Призначення та умови застосування програми	37
5.2 Характеристика програми.....	38
5.3 Початкові та вихідні дані	39

5.4	Проектування бази даних.....	39
5.5	Основні алгоритми програми	44
5.5.6	Алгоритм роботи користувача з системою	48
5.6	Висновки до розділу 5	49
6	ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	50
6.1	Опис методики проведення тестування	50
6.2	Результати використання розробленого програмного забезпечення	51
6.3	Висновки до розділу 6	60
	ВИСНОВКИ	61
	ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ.....	63
	ДОДАТОК А. Текст програми	65
	ДОДАТОК Б. Слайди презентації	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface (Інтерфейс програмування застосунків)
IDE	– Integrated Development Environment (Інтегроване середовище розробки)
JVM	– Java Virtual Machine
MVC	– Model-View-Controller (Модель-Вид-Контролер)
MVP	– Model-View-Presenter (Модель-Вид-Пред'явник)
MVVM	– Model-View-ViewModel (Модель-Вид-Модель представлення)
VS	– Visual Studio
БД	– База даних
ІК	– Інтерфейс користувача
ПЗ	– Програмне забезпечення
СКБД	– Система керування базами даних

ВСТУП

В сучасному світі мобільні застосунки стають невід'ємною частиною життя людей, а темп їх життя постійно зростає, планування стає все більш важливим для них. Це стосується і студентів, яким потрібно не лише встигати за навчальним планом, але й знаходити час для інших важливих справ, таких як робота, спорт, хобі та спілкування з друзями.

Традиційні методи планування, такі як щоденники та календарі, не завжди є зручними та ефективними в динамічному житті студентів. Мобільні застосунки для планування стають все більш популярними, оскільки вони пропонують ряд переваг, таких як:

- зручність використання – мобільні застосунки доступні на смартфонах та планшетах, що робить їх доступними для користувачів в будь-який час і в будь-якому місці;
- функціональність – мобільні застосунки для планування пропонують широкий спектр функцій, таких як створення подій, списків справ, нотаток та розкладу занять.

Актуальність роботи полягає в створенні мобільного застосунку, що буде забезпечувати функціями планування та збереження нотаток та вебсервісу для зв'язку з базою даних, де будуть зберігатись всі данні.

Розробка мобільного застосунку для планування, який буде відповідати потребам студентів, є актуальною та перспективною задачею.

Об'єктом дослідження даної дипломної роботи є веб та мобільні застосунки для планування, які дозволяють студентам створювати та редагувати події, розклад занять, нотатки та профіль, зберігаючи інформацію на сервері, що дозволить користувачу мати доступ до його даних з декількох мобільних пристроїв.

Предметом дослідження даної дипломної роботи є процес розробки мобільного застосунку для планування, який включає в себе:

- аналіз предметної області;

- вибір інструментарію для розробки;
- розробку архітектури системи;
- розробку програмного продукту;
- тестування застосунку.

Метою даної дипломної роботи є створення надійного, зручного та функціонального програмного продукту, який допоможе користувачам ефективно керувати своїм часом та завданнями, а також є створення інтуїтивно зрозумілого інтерфейсу, який забезпечить легку навігацію та зручність використання.

В даній дипломній роботі буде розроблено мобільний застосунок, який буде зручним у використанні, функціональним та ефективним, буде відповідати потребам студентів, може допомогти їм краще організувати свій час, підвищити продуктивність та зменшити стрес.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Метою дипломної роботи є створення мобільного застосунку для планування, для функціонування якого необхідно створити вебсервіс для зв'язку мобільного застосунку із базою даних.

1.1 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

1.1.1 Програмний продукт «To-do List»

Як перший приклад програмного забезпечення для планування та організації справ розглянемо застосунок «To-do List» [1], це сервіс для планування щоденного розкладу. Приклад застосунку наведено на рисунку 1.1.

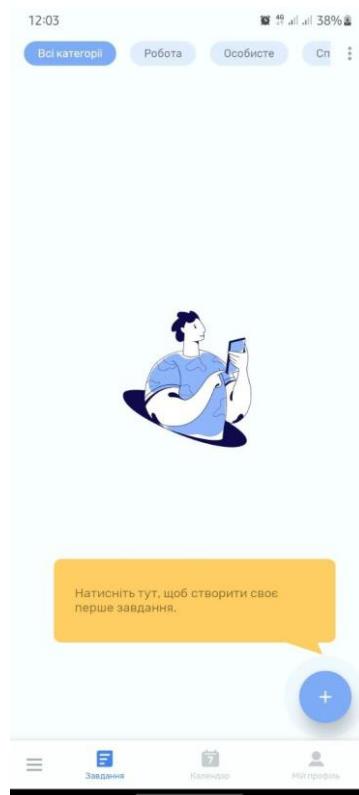


Рисунок 1.1 – Приклад застосунку «To-do List» [1]

Цей застосунок надає непоганий функціонал, такий як відображення запланованих подій у календарі, а також у вигляді списку (з обмеженням по кількості). Також є можливість додавати категорії подій (з обмеженням по кількості).

Із недоліків це те що можна створювати повторювані події, але змінити інформацію для однієї конкретної події не можна, лише змінювати інформацію до всіх наступних подій. Також це платний застосунок і більшість функціоналу доступна після оплати.

1.1.2 Програмний продукт «Microsoft To-do»

Другим аналогом розглянемо застосунок «Microsoft To-do» [2], це сервіс для планування щоденного розкладу. Приклад застосунку наведено на рисунку 1.2.

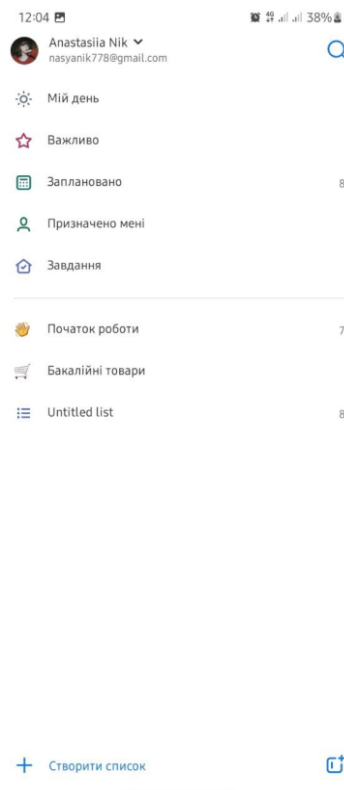


Рисунок 1.2 – Приклад застосунку «Microsoft To-do» [2]

Цей застосунок надає приблизно схожий функціонал, там є розділ із планами на день, важливі плани. Тут можна сортувати завдання за алфавітом або за датою.

Із недоліків це не дуже зручний інтерфейс, не зовсім зручне створення і відстеження всіх подій, які були створені.

1.1.3 Програмний продукт «Todoist»

Третім аналогом розглянемо сервіс «Todoist» [3], це застосунок для планування щоденного розкладу. Приклад застосунку наведено на рисунку 1.3.

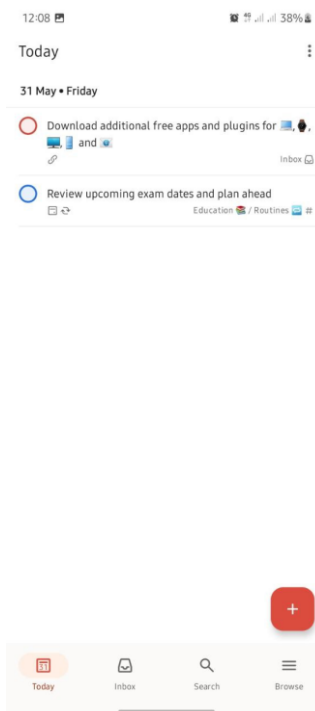


Рисунок 1.3 – Приклад застосунку «Todoist» [3]

Цей застосунок має функціонал створення подій у вигляді списку та нагадування. Є можливість фільтрації подій, та коментарі.

Із недоліків є те, що у безкоштовній версії, можна створювати обмежену кількість активних подій, немає функції нагадувань.

1.1.4 Порівняльна таблиця існуючих та розроблюваного продуктів

Вище було розглянуті популярні сервіси для планування та організації свого часу. Кожен з них має свої переваги та недоліки, такі як, наприклад, функціонал обмежений і доступний у повному обсязі після оплати.

У таблиці 1.1 наведено порівняння вже існуючих програмних продуктів та застосунку «Planify», що буде створено.

Таблиця 1.1 – Порівняння програмних продуктів

	To-do List	Microsoft To-Do	Todoist	Planify
Створення подій	+	+	+	+
Створення нотаток	–	–	–	+
Наявність категорій подій	+	–	+	+
Розклад для студентів	–	–	–	+
Статистика виконаних подій	+	–	–	+

1.2 Висновки за розділом 1

В першому розділі було проаналізовано існуючі методи вирішення завдання, а саме було проведено огляд фреймворків для розробки мобільних та вебзастосунків, що мають вирішувати поставлено задачу.

Для більш розгорнутого аналізу було проведено порівняння програмних продуктів, які виконують аналогічний функціонал. Для цього у кожного було визначено певні переваги та недоліки, на основі яких потім було побудовано порівняльну таблицю.

2 ВИБІР МОВИ ПРОГРАМУВАННЯ ТА ІНШИХ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Вибір інструментарію розробки програмного забезпечення

Головним етапом розробки програмного забезпечення (ПЗ) – є вибір інструментарію, за допомогою яких будуть втілюватись в життя визначені вимоги до програмного продукту.

2.1.1 Огляд особливостей обраної мови програмування

По-перше, необхідно обрати мову програмування для реалізації функціоналу, серед найбільш відповідних для розробки мобільних застосунків C# та Java. Порівняння мов наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

	C#	Java
	1	2
Парадигма програмування	Об'єктно-орієнтована, з підтримкою імперативного та функціонального програмування.	Чисто об'єктно-орієнтована мова програмування
Кросплатформність	Застосунки на C# можуть працювати на найрізноманітніших пристроях, таких як мобільний телефон, комп'ютер, ноутбук і сервер [4].	Java запускає байт-код програми Java у віртуальній машині Java Virtual Machine (JVM) на будь-якій платформі, на якій встановлена програма Java [5].

Продовження таблиці 2.1

	1	2
Сумісність з середовищами розробки	Сумісна з такими середовищами, як Visual Studio, MonoDevelop, JetBrains Rider.	Сумісна з IntelliJ IDEA, Eclipse, NetBeans.
Бібліотеки	Має велику екосистему .NET з безліччю бібліотек та фреймворків. Наприклад .NET Framework - це платформа від Microsoft, що надає широкий спектр бібліотек та інструментів для розробки на C#.	Має екосистему бібліотек, що робить її однією з найрізноманітніших та найбагатших у світі програмування. Вона має бібліотеки для стандартних задач. Також існує безліч бібліотек сторонніх розробників.
Спільнота	Є спільнота розробників з широкою підтримкою. А також отримує постійні оновлення і підтримку від Microsoft.	Є одною з найбільших та найактивніших спільнот розробників у світ

Для розробки було обрано мову програмування C#, оскільки ця мова має багатий інструментарій та багато можливостей.

2.1.2 Огляд особливостей обраного середовища розробки

Після вибору мови програмування, необхідно обрати середовище для розробки майбутнього програмного, а саме вибір був між Microsoft Visual Studio та JetBrains Rider. Для кращої візуалізації порівняння наведено у таблиці 2.2.

Таблиця 2.2 – Середовища розробки

	Microsoft Visual Studio	JetBrains Rider
	1	2
Функціональність	потужне і повнофункціональне середовище розробки, яке підтримує багато мов програмування, засобів для створення вебзастосунків, налаштування інтерфейсу, налагодження та інше. Воно має широкий набір інструментів і доповнень для поліпшення продуктивності розробників.	повнофункціональне середовище розробки, спеціально призначене для розробки на платформі .NET і ряді інших мов програмування. Надає інтегровану підтримку для інструментів від JetBrains, таких як ReSharper, і забезпечує розширену підтримку для функціонального програмування.
Підтримка мов програмування	підтримує широкий спектр мов програмування, включаючи C#, C++, Visual Basic, F#, JavaScript, Python та інші.	підтримує багато мов програмування, зокрема C#, VB.NET, F#, JavaScript, TypeScript, HTML, CSS, SQL, JSON, XML, Markdown і багато інших.
Кросплатформність	підтримує операційні системи Windows і macOS, але має обмежену підтримку для розробки на платформі Linux.	підтримує операційні системи Windows, macOS і Linux, що робить його хорошим вибором для розробки на будь-якій з цих платформ.

Продовження таблиці 2.2

	1	2
Інтегровані інструменти	має широкий набір інтегрованих інструментів, таких як вбудований редактор коду, дизайнер інтерфейсу, система контролю версій, інструменти налагодження, профілінгу і багато інших.	також надає інтегровані інструменти для редагування коду, системи контролю версій, налагодження, профілінгу і тестування, а також додаткові інструменти від JetBrains, такі як ReSharper для покращення продуктивності.
Спільнота	має велику спільноту розробників та активний вебсайт з документацією, уроками та форумами підтримки. Він також отримує постійні оновлення і підтримку від Microsoft.	має активну спільноту розробників і форуми підтримки, а також надає оновлення та підтримку для своїх продуктів.
Вартість	має декілька версій, включаючи безкоштовну Community Edition, яка надає базові можливості для індивідуальних розробників та невеликих команд. Однак, платні версії мають розширені функції і підтримку.	є комерційним продуктом, і для його використання потрібна придбання ліцензії. JetBrains також надає можливість використовувати Rider в рамках їх підписки All Products Pack, яка включає в себе інші інструменти компанії.

Проаналізувавши вище наведену інформацію, було обрано середовище Microsoft Visual Studio, через його багату функціональність, підтримку необхідних мов програмування, велику кількість документів та безкоштовну версію.

2.2 Вибір інструментарію розробки мобільного застосунку

Нижче наведено приклади фреймворків для розробки мобільних застосунків.

Xamarin – це платформа з відкритим кодом для створення сучасних та продуктивних застосунків для iOS, Android та Windows за допомогою .NET. Xamarin працює в керованому середовищі, яке надає такі функції, як розподіл пам'яті та збір сміття. Він дозволяє розробникам ділитися в середньому 90% своїх застосунків на різних платформах [6].

Xamarin.Forms – це фреймворк користувацького інтерфейсу з відкритим вихідним кодом, що дає змогу розробникам створювати застосунки Xamarin.Android, Xamarin.iOS і Windows на основі єдиної загальної кодової бази. Applications.Xamarin.Forms дає змогу розробникам створювати користувацькі інтерфейси в стилі XAML за допомогою коду на C#. Ці інтерфейси відображаються у вигляді ефективних нативних елементів керування на кожній платформі. Xamarin.Forms має велику екосистему бібліотек, які додають різноманітні функції в створюваний застосунок. Він надає спільний API для візуалізації нативних елементів керування на різних платформах, але деякі функції можуть бути недоступні на одних платформах [7].

Для розробки мобільного застосунку було обрано фреймворк Xamarin.Forms.

2.3 Вибір інструментарію розробки вебзастосунку

Існує декілька популярних фреймворків для створення вебзастосунків.

ASP.NET Web API – це потужний інструмент для розробки HTTP-сервісів, які легко інтегруються з браузерами, мобільними пристроями та іншими платформами. Цей фреймворк на базі .NET Framework ідеально підходить для створення RESTful API, які забезпечують гнучкість та масштабованість [8].

Фреймворк ASP.NET Core MVC – це простий фреймворк із відкритим вихідним кодом та є зручним інструментом для тестування. ASP.NET Core MVC надає можливість розробки динамічних вебсайтів на основі шаблонів, а також забезпечує чітке розділення завдань. Він дає вам повний контроль над розміткою та використовує останні вебстандарті. ASP.NET Core MVC побудовано на основі маршрутизації ASP.NET Core, що є потужним компонентом зіставлення URL-адрес, що забезпечує створення програм, які мають зрозумілі та доступні для пошуку URL-адреси [9].

Blazor – це фреймворк для створення інтерактивних вебінтерфейсів користувача з використанням .NET, що дозволяє створювати багатофункціональні та зручні інтерфейси, використовуючи мову програмування C# замість JavaScript. За допомогою Blazor логіку застосунку можна розділити на серверну та клієнтську частини, обидві написані на .NET. Blazor може рендерити користувацький інтерфейс на HTML та CSS, що робить його сумісним з різними браузерами, в тому числі мобільними пристроями. Blazor сумісний з широким спектром браузерів, включаючи мобільні пристрої. Він також інтегрований з сучасними хостинговими платформами, такими як Docker, що дозволяє легко розгортати і розширювати застосунок [10].

Для розробки вебзастосунку було обрано ASP.NET, адже це безкоштовна вебплатформа для створення красивих вебсайтів і вебзастосунків.

2.4 Вибір інструментарію для роботи з базою даних

2.4.1 Огляд систем керування базами даних

Є багато систем керування базами даних, які можна використовувати із мовою C#, розглянемо деякі з популярних:

SQLite – це компактна, легка вбудована база даних для зберігання та управління інформацією в застосунках. Це одна з найпопулярніших і широко використовуваних баз даних у світі. Головні переваги SQLite - простота й універсальність. Його не потрібно встановлювати або налаштовувати окремо, він працює всередині програми. Тому SQLite можна використовувати на різних платформах і з різними мовами програмування. SQLite підтримує стандартні SQL-запити та операції, які корисні для маніпулювання даними; SQLite забезпечує швидкий доступ до інформації та підтримує безліч типів даних, включно з цілими числами, рядками, датою та часом і двійковими даними [11].

Microsoft SQL Server – це система управління базами даних, яка була розроблена компанією Microsoft. В якості серверу даних він виконує основну функцію – зберігає і надає дані у відповідь на запити інших програм, які можуть працювати на тому самому сервері або в мережі. Для запитів використовується мова Transact-SQL, створена спільно Microsoft і Sybase, яка є розширенням стандарту ANSI/ISO для структурованих мов запитів SQL. Її використовують у широкому спектрі малих і середніх баз даних, а також у базах даних великих підприємств. Transact-SQL вже багато років конкурує з іншими системами управління базами даних [12].

MySQL - одна з найпопулярніших систем управління базами даних, що пропонує швидку і надійну роботу та багато можливостей MySQL базується на структурованій моделі даних, яка використовує табличну структуру. Всі таблиці мають рядки і стовпці, таким чином інформація в таблиці зберігається в рядках, а стовпці використовуються в якості опису даних. Для роботи з MySQL використовується мова Structured Query Language (SQL). За її

допомогою можна створювати, змінювати та отримувати дані з бази даних [13].

Для роботи над програмним продуктом було обрано СКБД SQLite, через зручність, компактність та легкою роботою із зберіганням та управлінням даних.

2.4.2 Огляд фреймворків для роботи з базою даних

Для роботи із базою даних необхідно обрати фреймворк, який буде надавати необхідний інструментарій. Далі наведено деякі популярні фреймворки:

- Entity Framework (EF) – офіційний ORM (об'єктно-реляційне відображення) фреймворк для платформи .NET. Він дозволяє розробникам працювати з даними як з об'єктами C# замість написання SQL-запитів. EF підтримує різні провайдери БД, такі як SQL Server, MySQL, PostgreSQL, SQLite та багато інших [14];

- NHibernate – це порт інструмента Hibernate Object Relational Mapping (ORM) для платформи Microsoft.NET. Він забезпечує платформу для відображення об'єктно-орієнтованих моделей домену до традиційних реляційних баз даних. Його мета – звільнити розробників від більшості завдань програмування, пов'язаних із зберіганням реляційних даних [15].

Для розробки заданого програмного продукту було обрано фреймворк Entity Framework, через зручність, широкий вибір інструментів та гнучку роботу.

2.5 Вибір шаблону проєктування

Важливим етапом розробки є вибір шаблону проєктування, нижче буде наведено деякі з них.

Model-View-ViewModel (MVVM) – це шаблон дизайну, який використовується при розробці архітектури застосунків. Він допомагає чітко відокремити логіку програми від інтерфейсу користувача (UI). Таке розділення логіки програми та інтерфейсу користувача може вирішити багато проблем розробки, спростити тестування, підтримку та вдосконалення застосунків, покращити можливості повторного використання коду та полегшити співпрацю розробників та дизайнерів інтерфейсу над розробкою різних частин програми.

Шаблон MVVM складається з 3 основних компонентів: Model (модель), view (перегляд) та viewmodel (модель перегляду). Кожен з них служить певній меті. Окрім розуміння обов'язків кожного компонента, важливо також зрозуміти, як вони взаємодіють між собою. На високому рівні подання розпізнає модель подання, а модель подання розпізнає модель, але модель не розпізнає модель подання, а модель подання не розпізнає подання. Таким чином, модель подання може відокремлювати подання від моделі та розвивати модель незалежно від подання [16].

На рисунку 2.2 наведено приклад взаємозв'язку між трьома компонентами.

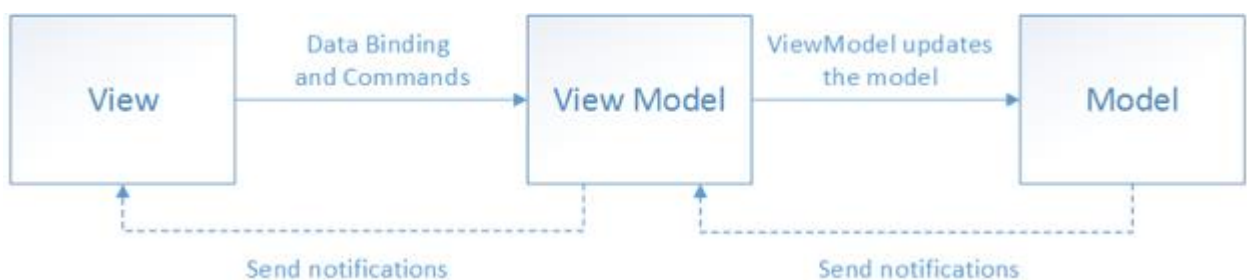


Рисунок 2.2 – Шаблон MVVM [16]

Деякі переваги цього шаблону полягають у наступному:

- можливість створювати модульні тести для моделей подань і моделей, не пов'язаних з поданнями. Модульне тестування моделей подань може виконувати ті ж функції, що і для подань;

- можливість змінити інтерфейс програми, не впливаючи на код, якщо подання повністю реалізовано в XAML. Отже, нова версія подання повинна працювати з існуючою моделлю подання;

- дизайнери та розробники можуть працювати над компонентами самостійно та одночасно в процесі розробки. Дизайнери можуть зосередитися на поданнях, а розробники – на моделях подань і компонентах моделей.

MVC (Model-View-Controller) – це архітектурний шаблон для розробки програмного забезпечення, який допомагає розділити програму на 3 основні частини.

Концепція MVC складається з наступних компонентів:

- модель (Model) – відповідає за обробку даних і бізнес-логіку застосунку. Він зберігає інформацію та визначає, як ця інформація обробляється та модифікується;

- вид (View) – відповідає за відображення даних користувачеві. Він представляє інформацію з моделі в зручному для користувача вигляді;

- контролер (Controller) – взаємодіє з користувачем і відправляє команди моделі і подання. Він керує потоком інформації між моделлю та поданням та реагує на дії користувача.

На рисунку 2.3 наведено приклад взаємозв'язку між компонентами шаблону.

MVC Architecture Pattern

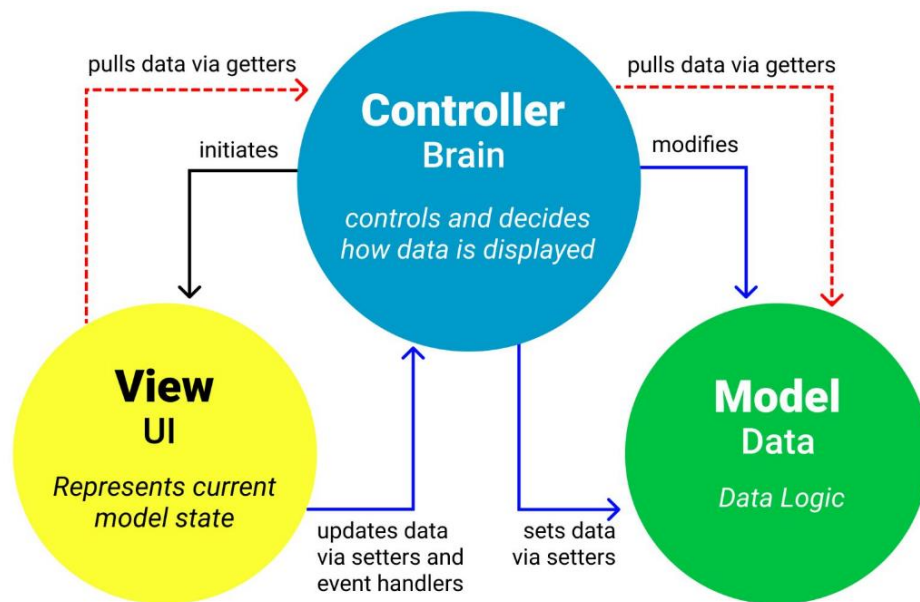


Рисунок 2.3 – Шаблон MVC [17]

MVC перетворює розробку складних застосунків на більш організований процес [17].

MVP (Model-View-Presenter) – це шаблон для розробки інтерфейсу користувача. Шаблон MVP – це класична версія MVC з дещо зміненим підходом, яка найчастіше використовується при розробці застосунків. Презентатор, який замінює контролер, має менше підключень до моделі і діє як функціональний рівень.

На рисунку 2.3 наведено приклад взаємозв'язку між компонентами шаблону.

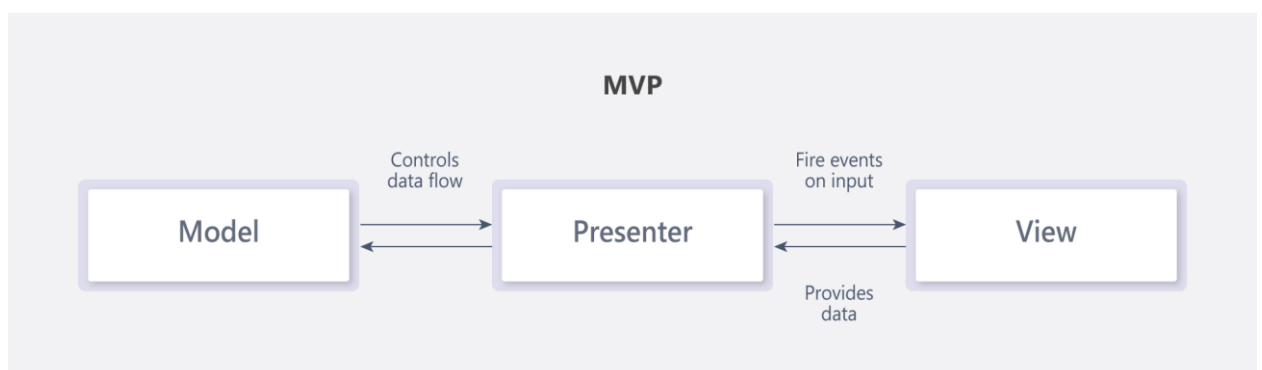


Рисунок 2.3 – Шаблон MVP [18]

Шаблон MVP, як і всі інші шаблони, був створений для того, щоб прискорити, спростити розробку і тестування програми [18].

Проаналізував можливості кожного із запропонованих шаблонів було обрано MVVM, адже він краще за все відокремлює компоненти, тобто модель, подання та модель подання, через що це дуже оптимізує розробку та спрощує роботу над програмним продуктом.

2.6 Висновки за розділом 2

На основі аналізу предметної області було обрано основні інструменти та методи для подальшої розробки програмного продукту.

Для розробки було обрано мову програмування C#, оскільки застосунки на C# можуть працювати на найрізноманітніших пристроях, вони можуть бути написані для мобільних телефонів, настільних комп'ютерів, ноутбуків і серверів.

В якості середовища розробки було обрано Microsoft Visual Studio, через його багату функціональність, підтримку обраної мови програмування, велику кількість документацій та безкоштовну версію.

Для розробки мобільного застосунку було обрано фреймворк Xamarin.Forms, адже він дає змогу розробникам створювати користувацькі інтерфейси в стилі XAML за допомогою коду на C#.

Для створення вебсервісу було обрано ASP.NET, адже це безкоштовна вебплатформа для створення красивих вебсайтів і вебзастосунків за допомогою HTML, CSS і JavaScript.

В якості системи керування базами даних було обрано SQLite, через компактність та легкість в роботі зі зберіганням та управлінням даних. Де головною унікальністю SQLite є те що його не потрібно встановлювати або налаштовувати окремо, він працює всередині програми. Для роботи із базою даних було обрано фреймворк, а саме Entity Framework, що дозволяє

розробникам працювати з даними у форматі об'єктів C# замість написання прямих SQL-запитів.

Одним з важливих етапів розробки є вибір шаблону проєктування, адже саме це дозволить зробити якісний та оптимальний продукт. В результаті аналізу різноманітних шаблонів проєктування було обрано шаблон MVVM (Model-View-ViewModel), адже він допомагає чітко відокремити логіку програми від її інтерфейсу користувача.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Структурна схема програми

Перед початком розробки необхідно визначити структуру програмного продукту, тому далі визначимо основні компоненти майбутнього застосунку.

Програмний продукт має складатись із мобільного застосунку та вебсервісу, що з'єднується із базою даних.

Мобільний застосунок виступає головною програмою, де повинен бути відображений наступний функціонал:

- реєстрація;
- авторизація;
- створення подій;
- відображення подій;
- створення розкладу;
- відображення розкладу;
- створення нотаток;
- відображення нотаток;
- відображення профілю.

Для підтримки мобільного застосунку повинен бути вебсервіс, з яким буде вестись зв'язок через API. Відповідно вебсервіс повинен спілкуватись із базою даних та повертати відповіді на запити.

На рисунку 3.1 наведено структурну схему програмного продукту.



Рисунок 3.1 – Структурна схема

3.2 Опис модулів розроблюваної системи

У таблиці 3.1 наведено опис модулів розроблюваної системи з визначенням основних функцій.

Таблиця 3.1 – Опис модулів системи

Модуль	Опис
Введення даних для реєстрації / авторизації	Тут користувач повинен мати можливість вводу необхідної інформації для реєстрації (ім'я, пошту, пароль та тип користувача) / авторизації (логін, пароль).
Відображення подій	Користувач повинен бачити 2 списки подій, сьогоднішні та майбутні.
Відображення нотаток	Користувач повинен бачити список із нотатками, відсортовані за датою додавання.
Відображення розкладу	Якщо користувач студент, то він повинен бачити 6 полів із відображенням розкладу та кнопку перемикач, для зміни типу тижня.
Відображення профілю	Користувач повинен побачити інформацію про себе (ім'я, пошта) а також про кількість подій та нотаток.

3.3 Архітектура системи Web API

Для розробки вебзастосунку необхідно розробити архітектуру web API.

Архітектура web API включає набір принципів та компонентів, які використовуються для побудови та розгортання інтерфейсів програмування застосунків через протокол HTTP. Основна мета архітектури web API полягає в наданні зручних і стабільних способів комунікації між різними системами та програмами.

Ось основні компоненти архітектури web API:

- клієнт – це програма або система, яка взаємодіє з web API. Клієнт може бути вебзастосунком, мобільним застосунком, настільним застосунком або іншою системою, що робить запити до API;

- сервер є центральним компонентом архітектури web API. Він приймає запити від клієнтів через протокол HTTP і надсилає відповіді з необхідними даними;
- формати відповідей – web API може повертати відповіді в різних форматах, таких як JSON, XML або інші, в залежності від потреб застосунків, які взаємодіють з API.

3.4 Функціональна схема програми

Після визначення структури програмного продукту необхідно розробити функціональну схему проєкту.

Функціональну схему програмного продукту наведено на рисунку 3.1.

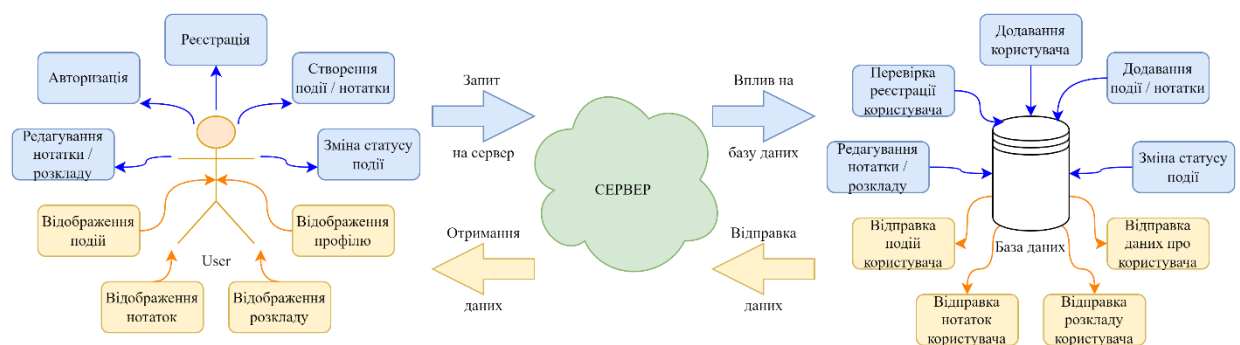


Рисунок 3.1 – Функціональна схема

3.5 Висновки до розділу 3

В даному розділі було визначено архітектуру веб та мобільного застосунків. Для відображення елементів мобільного застосунку було побудовано структурну схем. Далі було більш детально описано архітектуру веб застосунку, а саме принцип її роботи та основні компоненти.

На основі визначених даних, було побудовано функціональну схему програми, яка чітко показує взаємозв'язок веб та мобільного застосунків.

4 РОЗРОБКА ПРОТОТИПІВ ІНТЕРФЕЙСІВ СИСТЕМИ

4.1 Мета та вибір інструменту для прототипування

Прототипування інтерфейсу – це процес створення макетів майбутнього мобільного застосунку. Ці моделі можуть бути як простими ескізами, так і складними інтерактивними прототипами, що залежить від поставлених цілей.

Існує багато інструментів для прототипування, доступних як безкоштовно, так і платно. Деякі з популярних інструментів включають:

- Figma – це безкоштовний векторний онлайн-сервіс для прототипування та розробки інтерфейсів, де є можливість організації спільної роботи, коли над проєктом працює команда. Сервіс працює як у браузері так і в якості комп'ютерного застосунку. Головною перевагою є зберігання онлайн-версії файлів, над якими працював користувач. Даний редактор підходить для створення прототипів і дизайн-систем різної складності, яка буде залежати від проєктів [19];

- Adobe XD – це сервіс векторного дизайну для веб і мобільних програмних продуктів. Він доступний для різних платформ, таких як macOS і Windows, а також є версії для iOS і Android, що використовуються для попереднього перегляду результату роботи. Adobe XD дає змогу створювати фрейми для вебсайтів і створювати клікові прототипи [20].

Для розробки прототипів було обрано сервіс Figma, адже – це потужний та багатофункціональний інструмент для прототипування, який пропонує ряд переваг, що робить його чудовим вибором для дизайнерів та розробників. Нижче наведено ряд причин, чому було обрано саме цей сервіс:

- простота використання – Figma має інтуїтивно зрозумілий інтерфейс та просту криву навчання, що робить його доступним як для досвідчених дизайнерів, так і для початківців;

- функціональність – Figma пропонує широкий спектр функцій, які дозволяють створювати прототипи будь-якої складності. Тут є можливість

створювати макети інтерфейсу, додавати динамічні елементи, анімацію та навіть взаємодію;

- хмарне зберігання – Figma - це хмарний інструмент, що означає, що ваші прототипи завжди доступні вам з будь-якого пристрою;

- безкоштовна версія – Figma пропонує безкоштовну версію, яка включає в себе багато функцій, необхідних для створення базових прототипів;

- плагіни – існує безліч плагінів для Figma, які розширюють його функціональність. Ці плагіни дозволяють вам створювати прототипи більш складної взаємодії, додавати нові функції та інтегрувати Figma з іншими інструментами.

4.2 Результати прототипування

Розроблений прототип вікна авторизації та реєстрації наведено на рисунку 4.1.



Рисунок 4.1 – Прототипи вікон: а) авторизація; б) реєстрація

Розроблений прототип вікна подій наведено на рисунку 4.2.

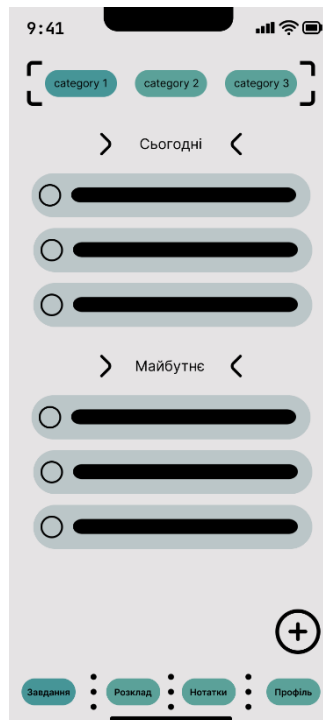


Рисунок 4.2 – Прототип вікна подій

Розроблений прототип вікна нотаток наведено на рисунку 4.3.

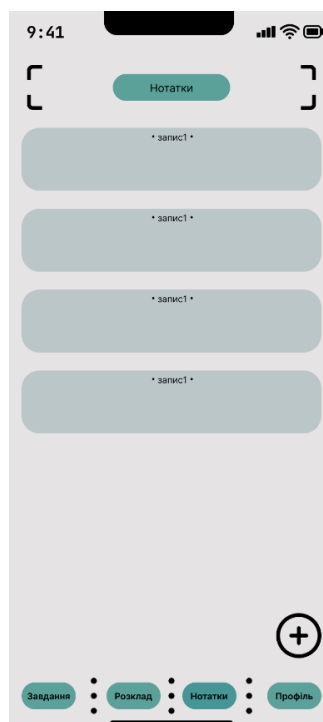


Рисунок 4.3 – Прототип вікна нотаток

Розроблений прототип вікна розкладу наведено на рисунку 4.4.

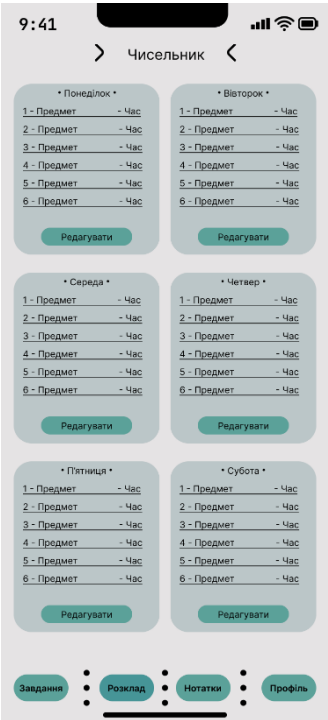


Рисунок 4.4 – Прототип вікна розкладу

Розроблений прототип вікна профілю наведено на рисунку 4.5.

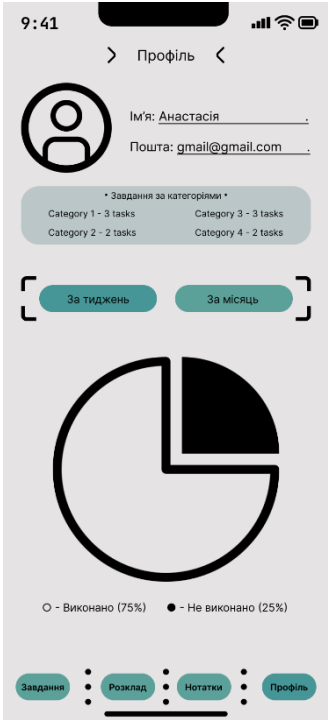


Рисунок 4.5 – Прототип вікна профілю

Завдяки функціоналу Figma було розроблено схему взаємодії вікон між собою (рис. 4.6), що показує зв'язки між вікнами та можливості програмного продукту

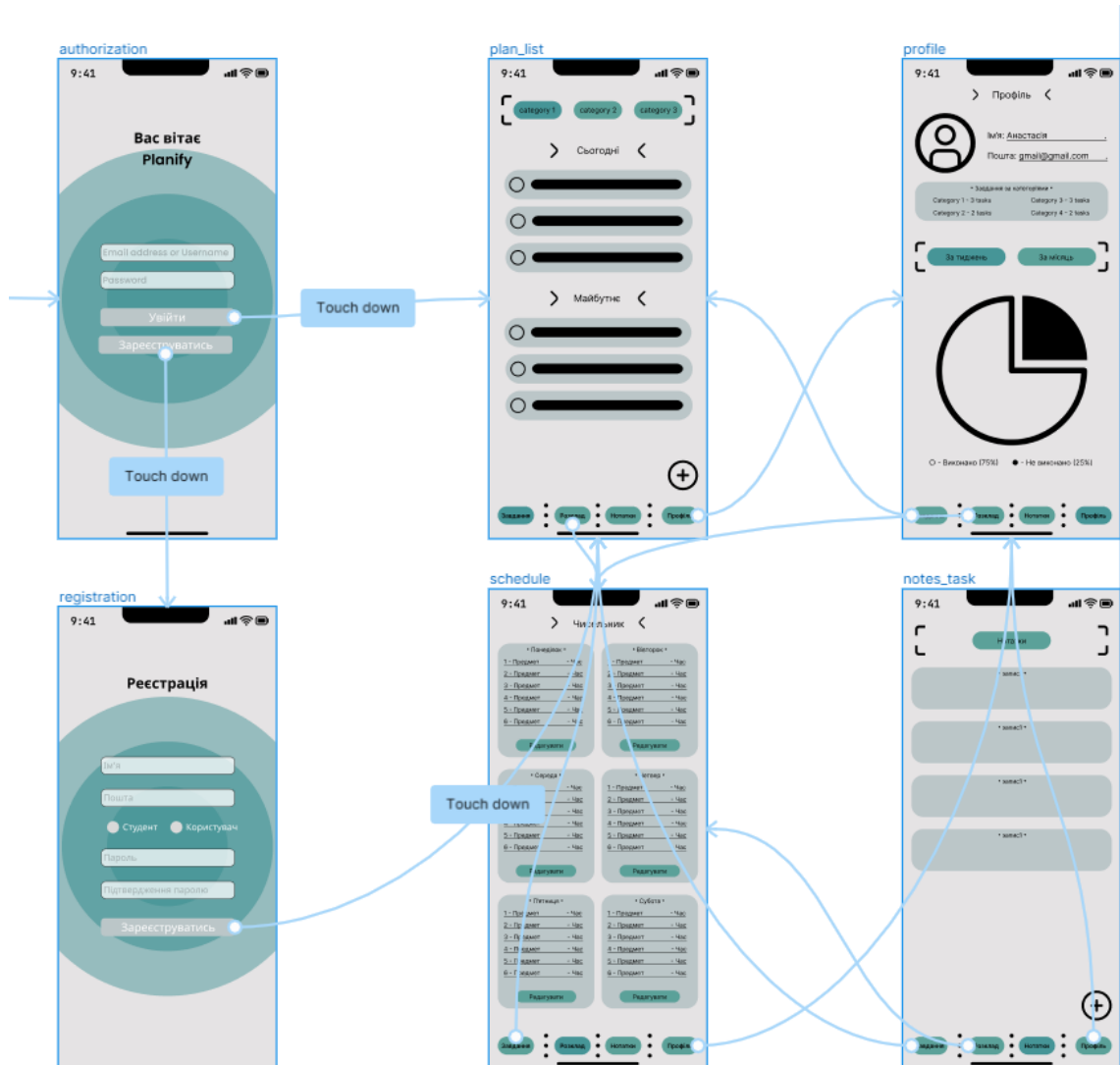


Рисунок 4.6 – Схема взаємодії вікон

4.3 Висновки за розділом 4

Для прототипування було обрано сервіс Figma, адже це багатофункціональний та дуже зручний інструмент для прототипування інтерфейсів будь якого типу.

За допомогою нього було розроблено прототипи застосунку, який буде розроблюватись.

5 РОЗРОБКА ПРОГРАМИ

5.1 Призначення та умови застосування програми

Програма призначена для вирішення проблеми планування та зберігання інформації в телефоні. Отже має наступний функціонал:

- зручний інтерфейс для вказання вхідних даних при реєстрації чи авторизації;
- інформативне і компактне відображення подій / нотаток / розкладу / профіля користувачу;
- інтуїтивно зрозуміле збереження даних про події / нотатки / розклад.

Так як програма має мобільну версію та сервер на якому знаходиться база даних, є певні вимоги до апаратної частини сервера та мобільного пристрою.

Відповідно сервер (ASP.NET) повинен мати:

- процесор – мінімальним вимогам до процесора на сервері Asp.NET є наявність 64-бітного процесора з тактовою частотою не менше 1.4 ГГц. Процесор повинен мати достатню кількість ядер (не менше 4) та пам'яті кешу (не менше 8 МБ), щоб забезпечити ефективну обробку запитів;
- пам'ять – мінімальним обсягом оперативної пам'яті на сервері ASP.NET є 2 ГБ. Однак, для високонавантажених вебзастосунків може знадобитися більш великий обсяг пам'яті;
- дисковий простір – на сервері повинен бути достатній обсяг вільного місця на жорсткому диску, щоб забезпечити зберігання даних у БД;
- операційна система – ASP.NET підтримується різними операційними системами, включаючи Windows Server 2019, Windows Server 2016 і Windows 10;
- браузер – для взаємодії з вебзастосунком, що працює на сервері ASP.NET, необхідний браузер, який підтримує HTML5 та CSS3;

- інтернет-з'єднання – для доступу до вебзастосунку, що працює на сервері ASP.NET, необхідне стабільне інтернет-з'єднання з достатньою швидкістю передачі даних.

Мобільний телефон повинен відповідати наступним вимогам:

- процесор – рекомендовано мати чотирьох-ядерний процесор з тактовою частотою не менше 2 ГГц;
- пам'ять – рекомендовано мати не менше 2 ГБ RAM для зручної роботи з застосунком планування;
- операційна система – телефон повинен працювати на Android з підтримкою 11 версії;
- рівень API – рекомендовано мати API 22 (Android 5.1);
- інтернет-з'єднання – для забезпечення синхронізації даних, оновлення та доступу до вебсервісу, важливо мати підтримку бездротових технологій, таких як Wi-Fi або мобільний Інтернет (4G або 5G).

Для повноцінного функціонування застосунку, необхідно аби вебсервіс був запущений на сервері, для того, щоб мобільний застосунок міг надсилати запити та відображати отримані з бази даних.

5.2 Характеристика програми

Програма має 2 частини, а саме веб та мобільний застосунки, що являються клієнтом та сервером відповідно.

Вебзастосунок є сервером, на якому зберігається БД та реалізована логіка роботи з нею, а саме алгоритми збереження та отримання даних із БД. Саме до нього надсилаються запити різних типів із мобільного застосунку, та він надсилає відповіді на них.

Мобільний застосунок є клієнтом, який надає інтерфейс для взаємодії із сервером, тут користувач отримує відображення даних із БД.

5.3 Початкові та вихідні дані

Вхідними даними є введені користувачем дані про нього, які вказуються при реєстрації. Також це дані логіну і пароллю – при авторизації, при умові що користувач вже зареєстрований.

Вихідними даними є отримана із бази даних інформація, яка належить користувачу, а саме події, нотатки, профіль та розклад, якщо користувач є студентом.

5.4 Проєктування бази даних

База даних в даному застосунку виконує важливу роль і має кілька основних функцій, таких як:

- зберігання даних – дозволяє зберігати великі обсяги структурованих даних, таких як інформація про користувачів, події, нотатки та розклад, таким чином вони будуть доступні для обробки та використання в різних частинах застосунку;
- забезпечення доступу до даних – дозволяє забезпечити доступ до даних для різних клієнтів або компонентів застосунку;
- обробка даних – виконує обробку даних на рівні сервера, що дозволяє виконувати запити до БД, сортувати та виконувати різні операції з даними;
- забезпечення цілісності та безпеки даних – дозволяє забезпечити цілісність даних, контролюючи правила введення та зберігання даних, адже виконується перевірка на введення даних та організовуються зв'язки між даними.

Концептуальна модель є важливим етапом при проєктуванні бази даних. Відповідно концептуальна модель допомагає визначити сутності та їх зв'язки у контексті бізнес-процесу. Вона відображає основні концепції і взаємозв'язки між об'єктами системи, що дозволяє краще розуміти потреби та

вимоги до бази даних. Також концептуальна модель допомагає визначити структуру даних, яка буде використовуватись у базі даних. Вона включає в себе визначення сутностей, їх атрибутів та зв'язків між ними. Це допомагає забезпечити цілісність даних, а також сприяє ефективній організації та збереженню даних.

Концептуальна модель служить засобом комунікації між розробниками. Вона допомагає уточнити вимоги до бази даних та забезпечити спільне розуміння структури та функціональності системи. Крім того, вона може бути використана для обговорення та отримання згоди щодо дизайну бази даних.

Концептуальна модель допомагає визначити ключові аспекти дизайну бази даних, такі як первинні ключі, зовнішні ключі та інші обмеження цілісності даних. Вона допомагає забезпечити правильне моделювання відношень між сутностями та визначення ролі кожного елемента в системі.

Проаналізувавши вимоги до застосунку, що проєктується, було створено наступні сутності:

- Users;
- Categories;
- Events;
- Notes;
- TimeTable;
- WeekTypes;
- TableItems.

На основі цих сутностей було побудовано концептуальну модель БД, що наведено на рисунку 5.1.

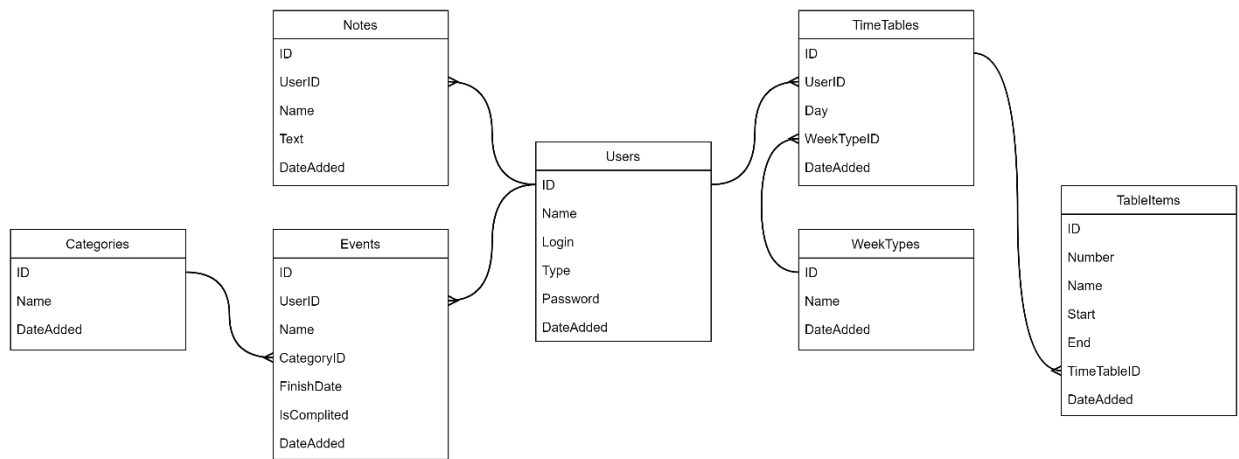


Рисунок 5.1 – Концептуальна модель

Сутність Users – сутність БД, що зберігає інформацію про користувачів. Опис полів наведено у таблиці 5.1.

Таблиця 5.1 – Сутність Users

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	Name	TEXT	DEFAULT
3	Login	TEXT	DEFAULT
4	Type	TEXT	DEFAULT
5	Password	TEXT	DEFAULT
6	DateAdded	DATETIME	DEFAULT

Сутність Categories – сутність БД, що зберігає інформацію про категорії подій. Опис сутності наведено у таблиці 5.2.

Таблиця 5.2 – Сутність Categories

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	Name	TEXT	DEFAULT
3	DateAdded	DATETIME	DEFAULT

Сутність Events – сутність БД, що зберігає інформацію про події. Опис сутності наведено у таблиці 5.3.

Таблиця 5.3 – Сутність Events

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	UserId	INT	DEFAULT
3	Name	TEXT	DEFAULT
4	CategoryId	INT	DEFAULT
5	FinishDate	DATETIME	DEFAULT
6	IsComplited	BIT	DEFAULT
7	DateAdded	DATETIME	DEFAULT

Сутність Notes – сутність БД, що зберігає інформацію про нотатки. Опис сутності наведено у таблиці 5.4.

Таблиця 5.4 – Сутність Notes

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	UserId	INT	DEFAULT
3	Name	TEXT	DEFAULT
4	Text	TEXT	DEFAULT
5	DateAdded	DATETIME	DEFAULT

Сутність TimeTables – сутність БД, що зберігає інформацію про розклад. Опис сутності наведено у таблиці 5.5.

Таблиця 5.5 – Сутність TimeTables

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	Day	TEXT	DEFAULT
3	UserId	INT	DEFAULT
4	WeekTypeId	INT	DEFAULT
5	DateAdded	DATETIME	DEFAULT

Сутність WeekTypes – сутність БД, що зберігає інформацію про тип тижня. Опис сутності наведено у таблиці 5.6.

Таблиця 5.6 – Сутність WeekTypes

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	Name	TEXT	DEFAULT
3	DateAdded	DATETIME	DEFAULT

Сутність TableItems – сутність БД, що зберігає інформацію про заняття. Опис сутності наведено у таблиці 5.7.

Таблиця 5.7 – Сутність TableItems

№	Назва поля	Тип даних	Розмір
1	Id	INT	DEFAULT
2	Number	INT	DEFAULT
3	Name	TEXT	DEFAULT
4	Start	DATETIME	DEFAULT
5	End	DATETIME	DEFAULT
6	TimeTableId	INT	DEFAULT
7	DateAdded	DATETIME	DEFAULT

5.5 Основні алгоритми програми

5.5.1 Алгоритм реєстрації / авторизації

У відповідні поля вводяться логін (пошта) та пароль, після чого ці дані надсилаються через API на сервер, де проходить перевірка із даними в базі даних. Даний алгоритм наведено у файлах «», «» та «», де спочатку дані надсилаються на сервер, а потім сервер, після обробки даних, надсилає відповідь.

Для реєстрації необхідно аби користувач ввів інформацію про себе, а саме, ім'я, пошту (далі логін), пароль та обрати тип користувача (студент чи звичайний користувач). Після чого дані надсилаються на сервер, для перевірки даних, чи наявні вони в БД, якщо ж ні – дані зберігаються, відповідно створюється новий обліковий запис користувача.

Фрагмент коду файлу «RegistrationPageViewModel.cs», де виконується надсилення даних реєстрації:

```
private async void RegistrationCommand()
{
    if(Pass != Pass2)
    {
        UserDialogs.Instance.Alert("Passwords are different!",
"Passwords ERROR", "Ok");
        Pass2 = "";
        return;
    }
    string uType = "";
    if (Student)
    {
        uType = "Student";
    }
    else
    {
        uType = "User";
    }
    using (HttpClient client = new HttpClient())
    {
        User user = new User()
        {
            Id = 0,
            Name = Name,
            Login = Email,
            Password = Pass,
            Type = uType,
            Events = new List<Event>(),
            Notes = new List<Note>()
        };
        string url = $"{Constants.APIConnectionHTTP}/registration";
```

```

        JsonConvert content = JsonConvert.Create(user);
        var response = await client.PostAsync(url, content);
        Constants.User = await response.Content.ReadFromJsonAsync<User>();
        string id = await client.GetStringAsync(url);
    }

```

Фрагмент коду файлу «Repository.cs», де прописано логіку збереження даних про нового користувача:

```

public void AddUser(User user)
{
    if (!dbContext.Users.Contains(user))
    {
        dbContext.Users.Add(user);
    }
    dbContext.SaveChanges();
}
public User CheckUser(User user)
{
    var chUser = dbContext.Users.FirstOrDefault(x => x.Login ==
user.Login && x.Password == user.Password);
    dbContext.Entry(chUser).Collection(x => x.Events).Load();
    dbContext.Entry(chUser).Collection(x => x.Notes).Load();
    chUser.Events = new(chUser.Events);
    chUser.Notes = new(chUser.Notes);
    chUser.Events.ForEach(y => y.User = null);
    chUser.Notes.ForEach(y => y.User = null);
    return chUser;
}

```

При авторизації користувач повинен ввести логін та пароль, після чого дані, так само надсилаються на сервер для перевірки наявності у базі даних. Якщо у БД є ці дані – до мобільного застосунку надсилається інформація про користувача.

Фрагмент коду файлу «AuthorizationPageViewModel.cs», як дані надсилаються на сервер:

```

private async void AuthorizationCommand()
{
    if (string.IsNullOrEmpty(Login) || string.IsNullOrEmpty>Password)) {
        UserDialogs.Instance.Alert("Введені не коректні дані!",
"Помилка авторизації", "Ок");
        Login = "";
        Password = "";
    }
    else
    {
        using (HttpClient client = new HttpClient())
        {
            string url = $"{Constants.APIConnectionHTTP}/login";
            User user = new User();
            user.Id = 0;
            user.Name = "";
            user.Login = Login;
            user.Password = Password;
            user.Type = "";
            user.Events = new List<Event>();
            user.Notes = new List<Note>();
            try

```

```

        {
            JsonConvert content = JsonConvert.Create(user);
            var response = await client.PostAsync(url, content);
            Constants.User = await
response.Content.ReadFromJsonAsync<User>();
            url = $"{Constants.APIConnectionHTTP}/getCategories";
            Constants.Categories =
JsonConvert.DeserializeObject<List<Category>>(await
client.GetStringAsync(url));
            url = $"{Constants.APIConnectionHTTP}/getWeekType";
            Constants.WeekTypes =
JsonConvert.DeserializeObject<List<WeekType>>(await
client.GetStringAsync(url));
            Constants.User.Events.ForEach(x => x.Category =
Constants.Categories.FirstOrDefault(y => y.Id == x.CategoryId));
        }
        catch (Exception ex) { }
        if (Constants.User != null)
        {
            if (Constants.User.Type == "Student")
            {
                await
NavigationService.NavigateAsync(nameof(EventsPageView));
            }
            else if (Constants.User.Type == "User")
            {
                await
NavigationService.NavigateAsync(nameof(UserEventsPageView));
            }
        }
        else
        {
            UserDialogs.Instance.Alert($"Користувача не
знайдено!\nСпробуйте інші дані!", "Помилка входу", "Ок");
            Login = "";
            Password = "";
        }
    }
}

```

Фрагмент коду файлу «Program.cs», де дані обробляються, та надсилається відповідь:

```

app.MapPost("/login", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var user = await context.Request.ReadFromJsonAsync<User>();
    await
context.Response.WriteAsync(JsonConvert.SerializeObject(repository.CheckUser(
user)));
});

```

Якщо користувач є студентом – йому буде доступно 4 вкладки застосунку, якщо ж звичайний користувач – лише три, адже розділ «Розклад» доступний лише для студентів.

5.5.2 Алгоритм створення подій / нотаток

Для створення події чи нотатки користувач повинен перейти у відповідне вікно. Для події необхідно ввести назву, категорію, дату і час планування, для нотаток вводиться лише назва і текст, далі дані надсилаються на сервер для збереження у БД.

Фрагмент коду файлу «Repository.cs», де прописано логіку збереження події в базі даних

```
public void AddEvent(Event newEvent)
{
    dbContext.Events.Add(newEvent);
    dbContext.SaveChanges();
}
```

5.5.3 Алгоритм створення розкладу

На початку роботи генерується псевдо-розклад, для подальшого збереження та редагування. Тут користувач повинен обрати день (понеділок-субота) та тип тижня (чисельник / знаменник) для редагування, натиснувши у відповідному полі кнопку «Редагувати», користувач повинен ввести необхідний перелік занять, заповнивши наступну інформацію: номер, назва, час початку і кінця. Після цього дані надсилаються у БД для збереження.

5.5.4 Алгоритм відображення подій / нотаток / розкладу

При авторизації користувача, мобільний застосунок отримує відповідь від сервера у вигляді об'єкта «Користувач», який містить в собі ім'я, логін, пароль, список подій та нотаток та розкладу. Також на сервер надсилаються запити на отримання системної інформації про категорії подій та типи тижнів.

Вся інформація зберігається і коли користувач відкриває певну вкладку, необхідна інформація відображається.

5.5.5 Алгоритм редагування нотаток / розкладу

Користувач повинен обрати подію чи нотатку, натиснувши у відповідному полі кнопку «Редагувати», починає змінювати необхідні йому дані. Після чого інформація надсилається на сервер і перезаписується в БД.

Для редагування розкладу, користувач обирає тип тижня та день, де натискає відповідну кнопку «Редагувати», після чого починає змінювати необхідні йому дані. Після редагування інформація надсилається на сервер і виконується перезапис в БД.

Фрагмент коду файлу «Repository.cs», де наведено логіку редагування нотатки:

```
public void EditNote(Note newNote)
{
    Note fNote = dbContext.Notes.FirstOrDefault(x => x.Id == newNote.Id);
    Note nNote = new Note()
    {
        UserId = newNote.UserId,
        Name = newNote.Name,
        Text = newNote.Text
    };
    if (fNote != null)
    {
        dbContext.Notes.Remove(fNote);
        dbContext.Notes.Add(nNote);
        dbContext.SaveChanges();
    }
}
```

5.5.6 Алгоритм роботи користувача з системою

Головний алгоритм системи наведено на рисунку 5.2.

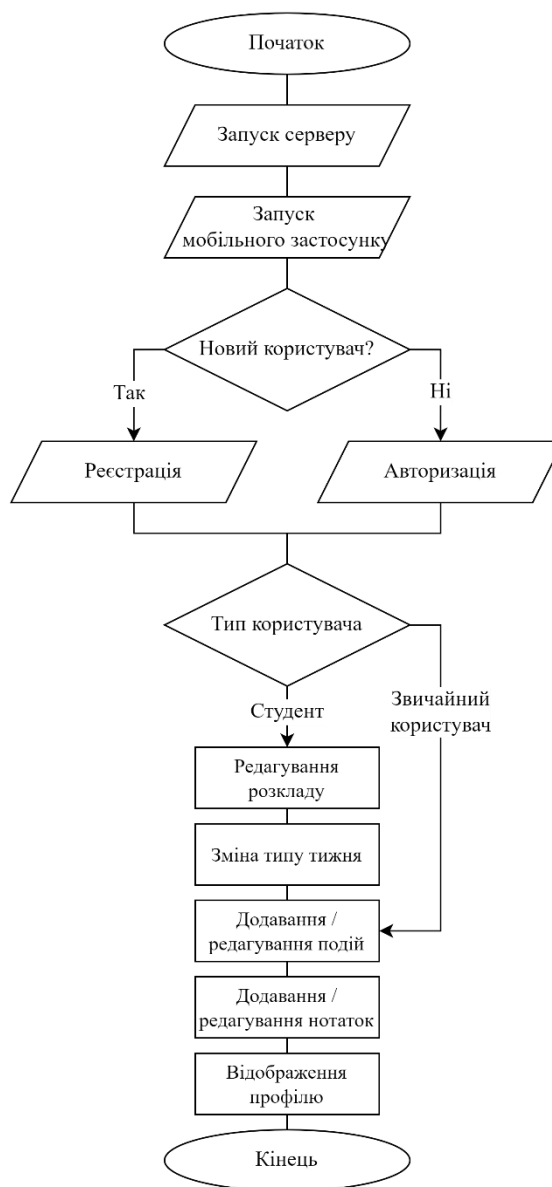


Рисунок 5.2 – Алгоритм системи

5.6 Висновки до розділу 5

В даному розділі було описано основні етапи та елементи розробки програмного забезпечення, а саме виділені мінімальні вимоги до апаратної частини, браузеру та мобільного пристрою, на яких буде використовуватись програмне забезпечення.

Також було приділено увагу до алгоритмів, та їх детального опису, адже на їх основі виконувалась розробка програмного забезпечення.

6 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Опис методики проведення тестування

Тестування буде проводитись методом «Ad Hoc» - його називають неформальним підходом до тестування програмного забезпечення, де випробувачі проводять тести без заздалегідь підготовлених тестових сценаріїв.

Особливості методу «Ad Hoc»:

- експериментальний підхід – тестувальники експериментують з програмним продуктом, щоб знайти помилки і протестувати різні функції та взаємодії. Вони можуть виконувати неочікувані дії, пробувати незвичні комбінації вхідних даних або переходити між функціями в застосунку без чітко визначеного сценарію;

- відсутність формальних тестових кейсів – спеціальні методи не використовують заздалегідь підготовлені тестові кейси або тестові сценарії. Тестувальники використовують власні знання, досвід та інтуїцію для визначення тестів;

- гнучкість і креативність – тестувальники можуть вільно змінювати напрямок тесту в залежності від проблем, які вони знаходять, або нових ідей, з якими вони стикаються. Це дозволяє їм гнучко реагувати на мінливі вимоги та ситуації;

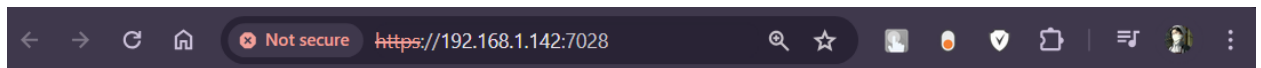
- документування результатів – важливо, щоб тестувальники записували знайдені проблеми, вжиті дії та отримані результати. Це дозволяє їм відстежувати і розуміти виконану роботу і спрощує процес подальшої перевірки і виправлення проблем;

- ефективність – у конкретних ситуаціях спеціальні методи часто використовуються на ранніх стадіях розробки, коли формальні тест-кейси недоступні або неефективні. Вони також корисні для швидкого тестування

окремих функцій, усунення критичних помилок або виявлення неочікуваної поведінки програмного продукту.

6.2 Результати використання розробленого програмного забезпечення

Для початку тестування необхідно аби було запущено сервер (рис. 6.1)



Commands menu:

```
/registration  
/login  
/getUsers  
/getCategories  
/getWeekTypes  
/addEvent  
/getEvents  
/editEvent  
/addNote  
/getNotes  
/editNote
```

Рисунок 6.1 – Запущений сервер

Тестування починається зі сторінки авторизації. Спробувавши ввести дані, було отримано повідомлення, що такого користувача не існує (рис. 6.2).

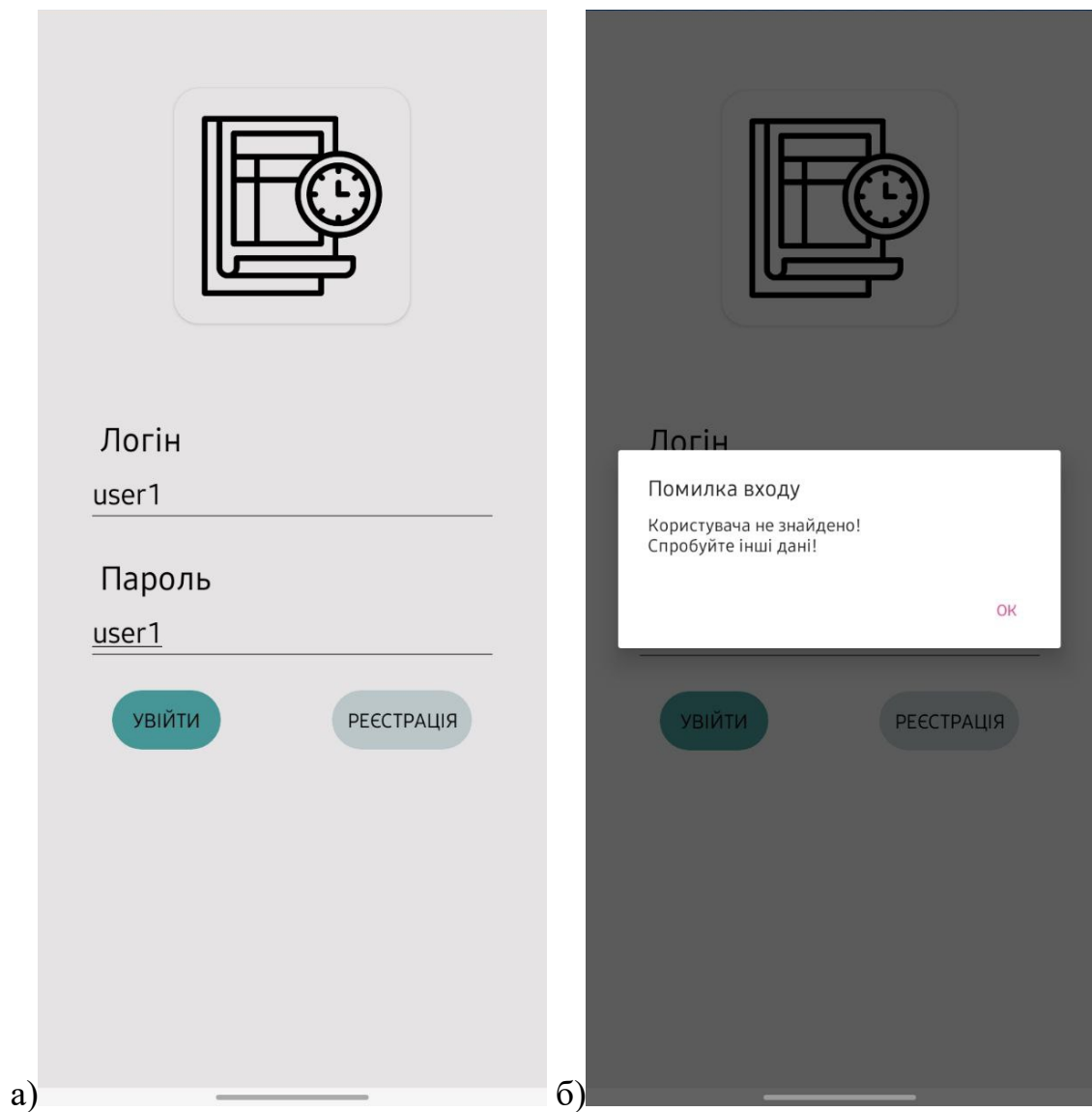


Рисунок 6.2 – Приклад авторизації неіснуючого користувача, де а) введення даних, б) результат помилки

Далі була спроба створити новий акаунт студента за допомогою реєстрації та авторизуватись (рис. 6.3).

The image displays two mobile application screens, labeled 'а)' and 'б)', representing the registration and login processes. Both screens feature a header icon of a book and a clock. Screen 'а)' is the registration form, with fields for 'Ім'я' (Name) containing 'test', 'Пошта' (Email) containing 'test@gmail.com', a role selection with 'Студент' (Student) selected and 'Користувач' (User) unselected, 'Пароль' (Password) containing 'test', and 'Підтвердження паролю' (Confirm password) containing 'test'. A 'ЗАРЕЄСТРУВАТИСЬ' (Register) button is at the bottom. Screen 'б)' is the login form, with fields for 'Логін' (Login) containing 'test@gmail.com' and 'Пароль' (Password) containing 'test'. It includes 'УВІЙТИ' (Login) and 'РЕЄСТРАЦІЯ' (Registration) buttons.

а) **Ім'я**
test

Пошта
test@gmail.com

☒ Студент ☐ Користувач

Пароль
test

Підтвердження паролю
test

ЗАРЕЄСТРУВАТИСЬ

б) **Логін**
test@gmail.com

Пароль
test

УВІЙТИ РЕЄСТРАЦІЯ

Рисунок 6.3 – Приклад успішної реєстрації та авторизації, де а) реєстрація нового користувача, б) авторизація

Тепер треба перевірити створення подій, так було створено декілька подій (рис. 6.4).

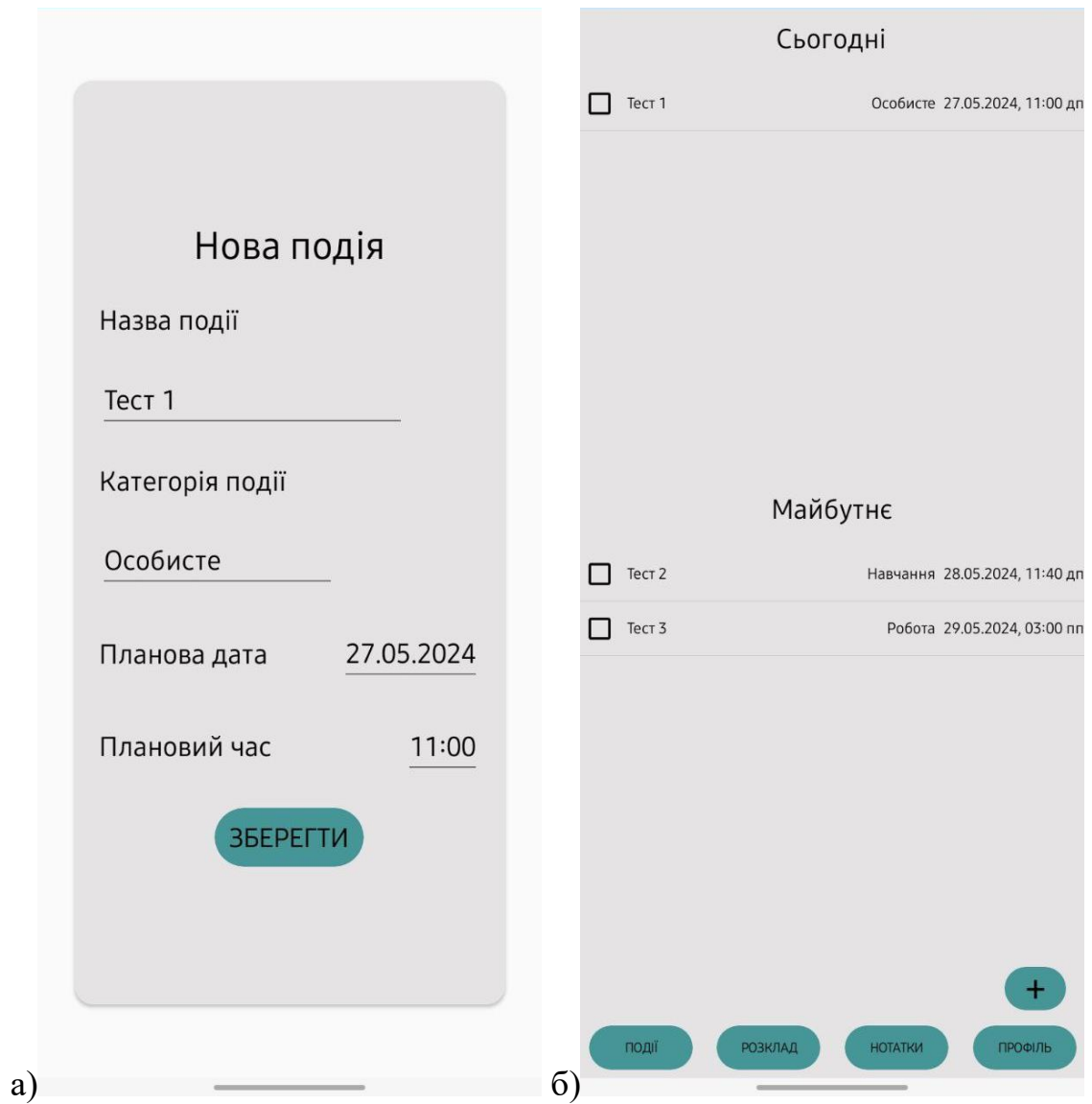


Рисунок 6.4 – Приклад успішного додавання подій, де а) приклад створення події, б) перелік доданих подій

У розділі з розкладом, відображено 6 полів з кнопкою (рис. 6.5).

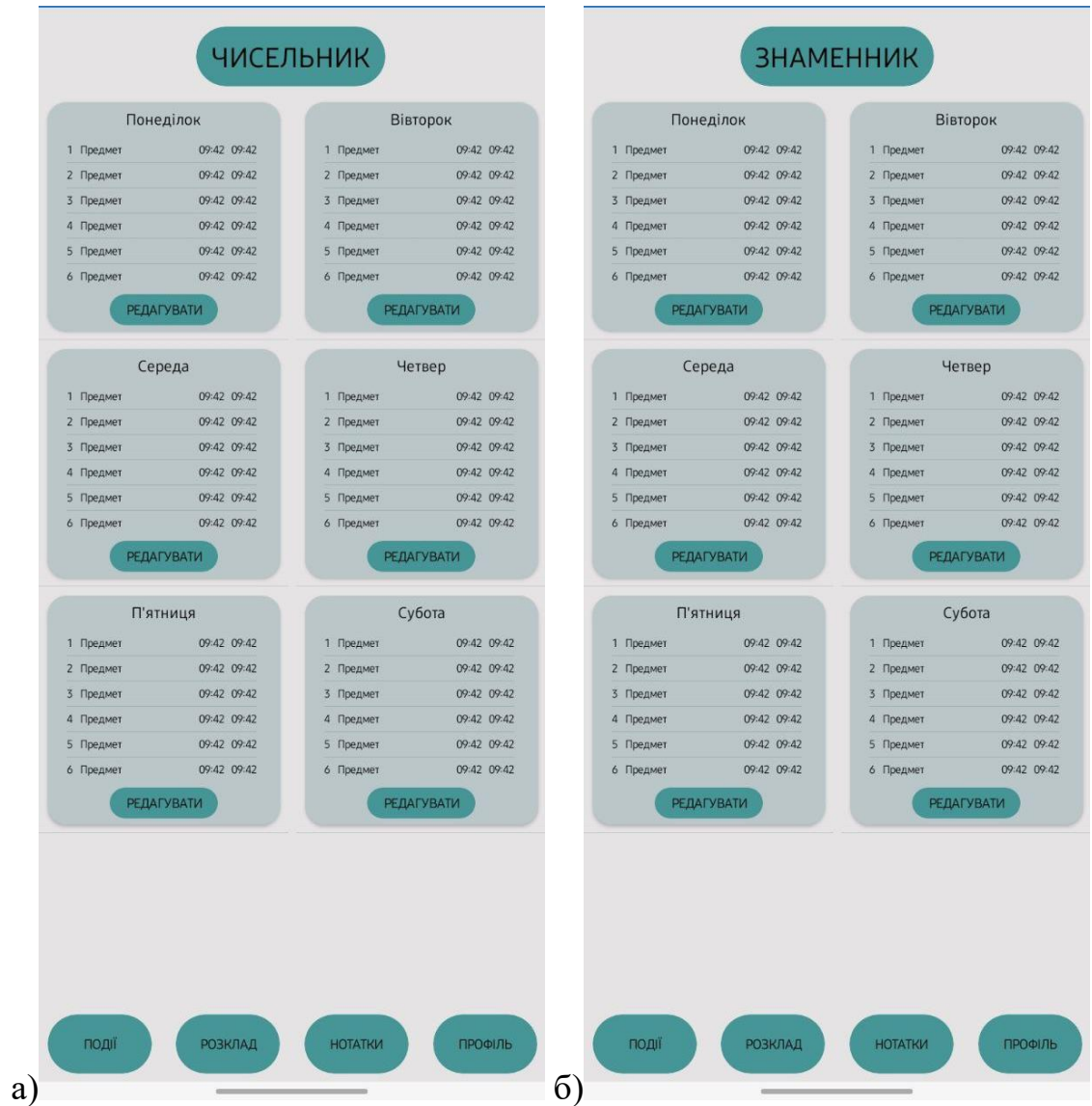


Рисунок 6.5 – Сторінка з розкладом, де а) розклад за чисельником, б) розклад за знаменником

Далі була спроба змінити розклад в понеділок за чисельником, результат наведено на рисунку 6.6.



Рисунок 6.6. – Змінений розклад в понеділок

Далі перейшовши у нотатки, була спроба створити нову (рис. 6.7).

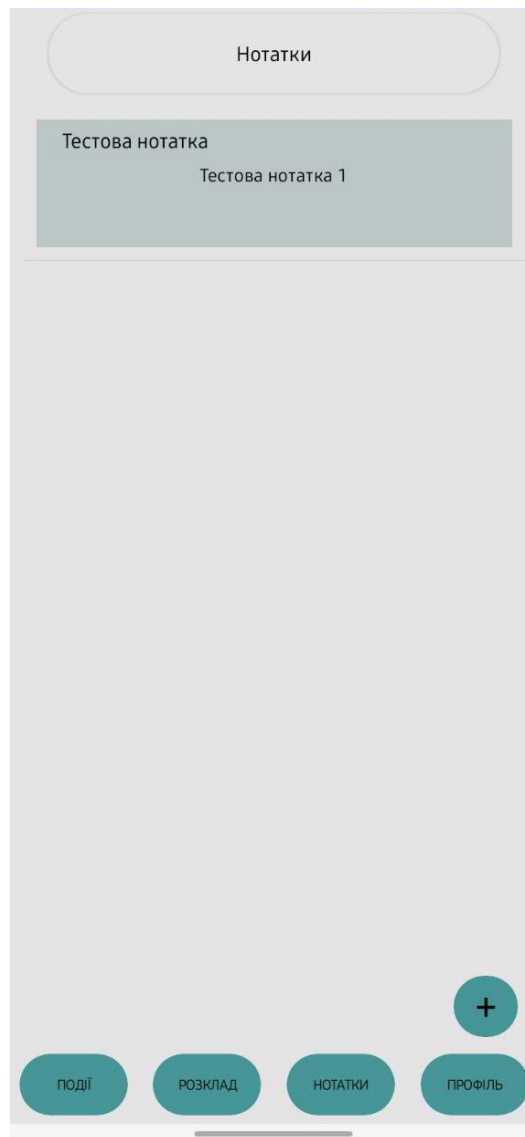


Рисунок 6.7 – Приклад створення нотатки

Далі була перевірка редагування створеної нотатки (рис. 6.8).

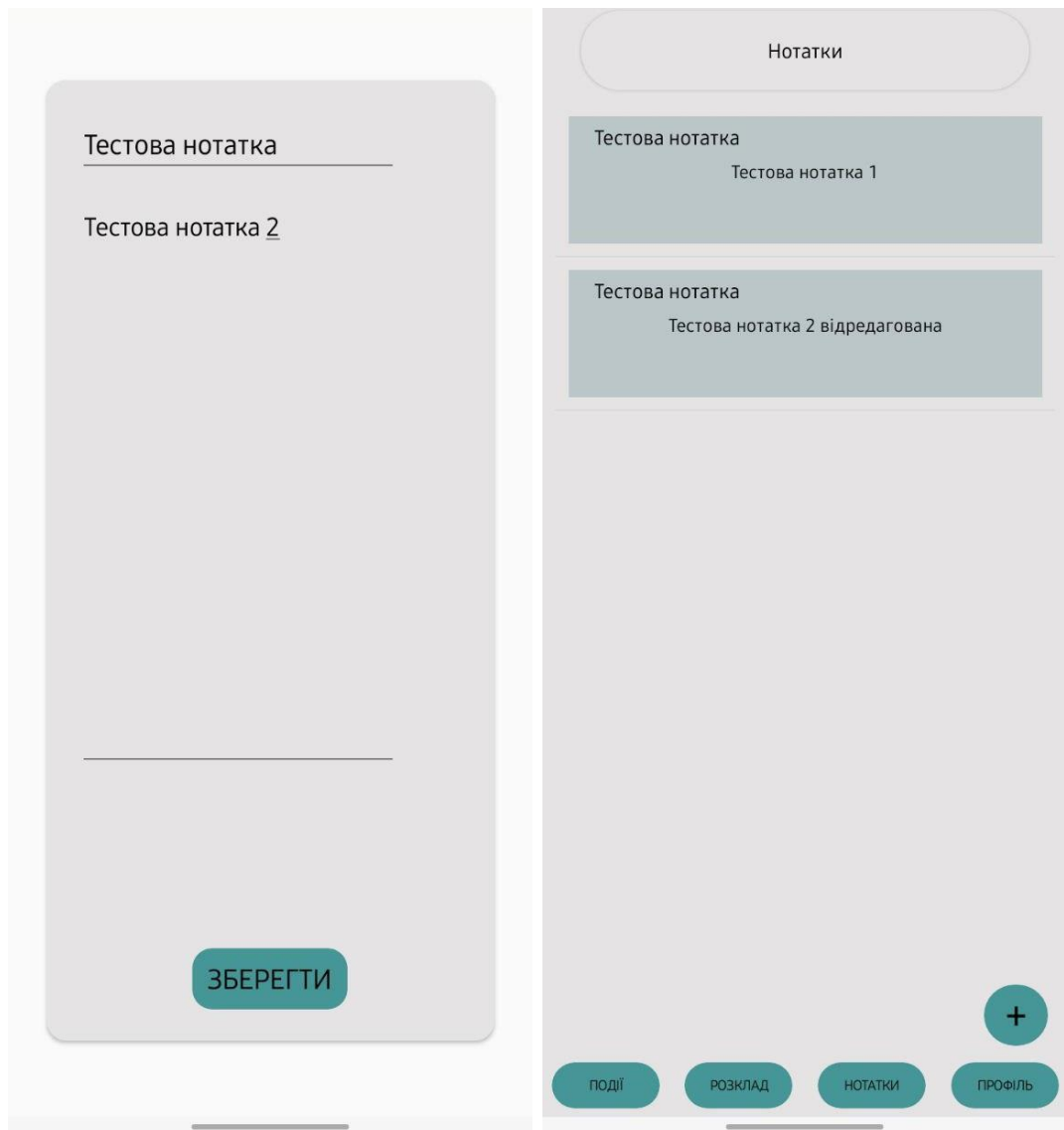


Рисунок 6.8 – Приклад редагування нотатки

Далі була перевірка статистики виконаних подій в профілі (рис. 6.9).

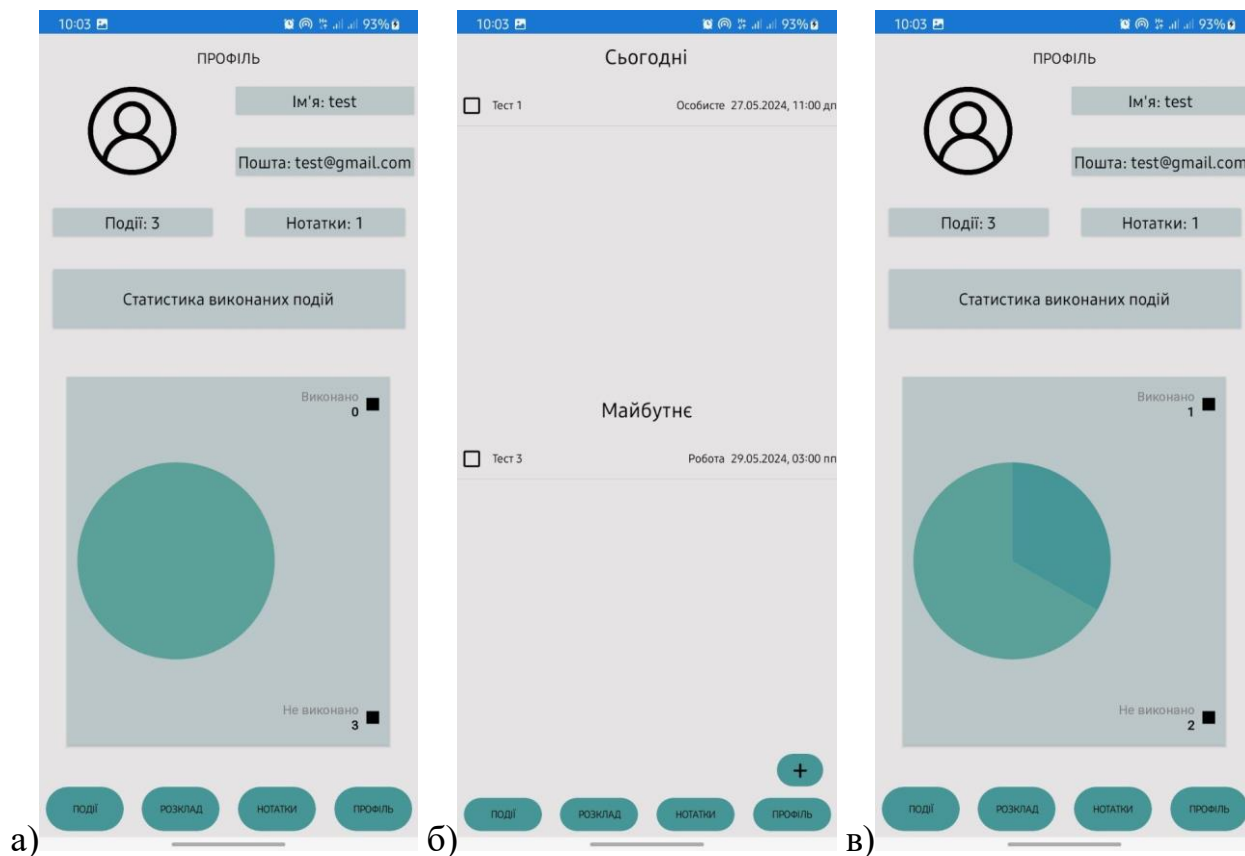


Рисунок 6.9 – Перевірка статистики, де а) перше відкриття статистики, б) відмітка, що завдання виконане, в) відображення змін в статистиці

Далі була перевірка реєстрації звичайного користувача, приклад наведено на рисунку 6.10.

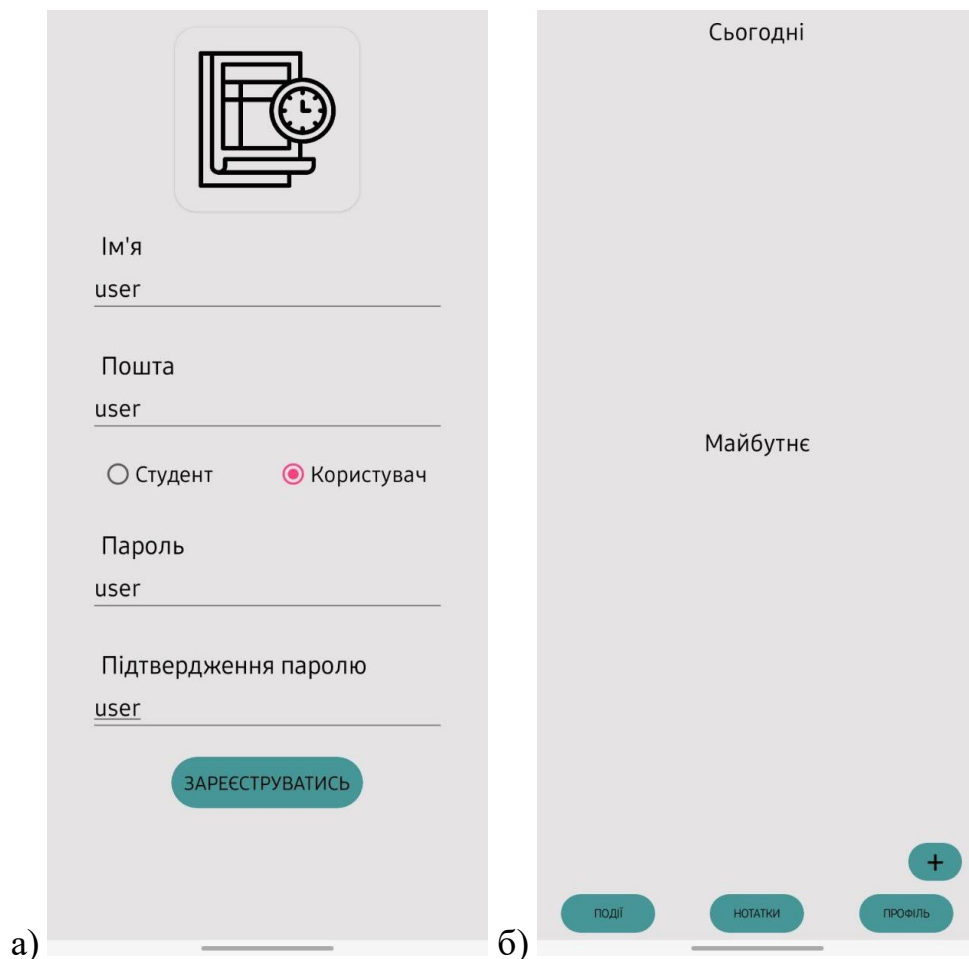


Рисунок 6.10 – Реєстрація звичайного користувача, де а) вікно реєстрації, б) початкове вікно користувача

На зображенні видно різницю режимів, адже у звичайних користувачів немає розділу з розкладом, а отже їм доступні розділи подій та нотаток. Функціонал створення подій та нотаток аналогічний.

6.3 Висновки до розділу 6

В даному розділі проводилось тестування розробленого програмного забезпечення методом «Ad Hoc», тобто неформальним підходом до тестування, де не було прописано конкретного сценарію дій тестувальника. Програмний продукт було ретельно протестовано та виявлено, що програмне забезпечення відповідає всім поставленим вимогам та готове до використання та експлуатації.

ВИСНОВКИ

У ході виконання дипломної роботи, було проведено детальне ознайомлення із предметною областю – планування та організації часу.

На основі поставленого завдання було проведено аналіз предметної області, а саме розглянуто вже існуючі програмні засоби, які вирішують поставлене завдання. Також їх було порівняно, визначивши певні переваги та недоліки.

Враховуючи зібрані під час аналізу дані, було обрано інструментарій для розробки ПЗ, а саме обиралась мова програмування і середовище для розробки, СКБД та інструментарію роботи з нею. Отже вибір припав на мову програмування C# та середовище розробки – Visual Studio 2022, а SQLite обрано в якості СКДБ, із використанням фреймворку Entity Framework, для взаємодії бази даних із програмою. Для розробки вебсервісу було обрано фреймворк ASP.NET, а для мобільного застосунку було обрано фреймворк Xamarin.Forms. Окрім цього було обрано шаблон проєктування, а саме MVVM (Model-View-ViewModel pattern).

Далі було розроблено архітектуру програми, а саме описано модулі системи, побудовано структурну та функціональну схеми. На основі цих даних були побудовані прототипи інтерейсів.

Наступним етапом розробки була розробка алгоритмів для реалізації різноманітних компонентів системи, таких як авторизація, реєстрація, відображення, отримання та надсилення даних.

В результаті було розроблено програмний продукт, який надає рішення поставленим задачам. Зокрема було реалізовано наступні функції:

- авторизація та реєстрація в системі;
- створення, редагування та відображення подій;
- створення, редагування та відображення нотаток;
- відображення профілю та статистики виконаних подій.

Тестування показало, що ПП є працездатним та ефективним для вирішення даних задач організації даних, а також має важливу перевагу – дані зберігаються на сервері, а отже доступні з будь-якого мобільного пристрою.

В подальшому можливо модернізувати ПП, розширивши його можливості та функціонал, аби зробити його більш доступним та зручним у використанні.

Результати роботи апробовані на науково-практичній конференції «Тиждень науки-2024», яка дистанційно проводилася у Національному університеті «Запорізька політехніка» у 2024 році.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Програмний продукт «To-do List – Schedule Planner» [Електронний ресурс] – Режим доступу :
<https://play.google.com/store/apps/details?id=todolist.scheduleplanner.dailyplanner.todo.reminders>
2. Програмний продукт «Microsoft To Do: Lists & Tasks» [Електронний ресурс] – Режим доступу :
<https://play.google.com/store/apps/details?id=com.microsoft.todos>
3. Програмний продукт «Todoist: to do list & planner» [Електронний ресурс] – Режим доступу :
<https://play.google.com/store/apps/details?id=com.todoist>
4. Огляд C# [Електронний ресурс] – Режим доступу :
<https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview>
5. Кросплатформність Java [Електронний ресурс] – Режим доступу :
<https://ravesli.com/cross-platform-of-java/>
6. Xamarin [Електронний ресурс] – Режим доступу :
<https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin>
7. Xamarin Forms [Електронний ресурс] – Режим доступу :
<https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms>
8. ASP.NET Web API [Електронний ресурс] – Режим доступу :
<https://learn.microsoft.com/en-us/dotnet/framework/wcf/wcf-and-aspnet-web-api>
9. ASP.NET [Електронний ресурс] – Режим доступу :
<https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0>
10. ASP.NET Core Blazor [Електронний ресурс] – Режим доступу :
<https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0>
11. Що таке SQLite? [Електронний ресурс] – Режим доступу :
<https://viter.zapisi.cx.ua/ukraincyam/shho-take-sqlite-prostimi-slovami-zrozumile-poyasnennya.html>
12. Microsoft SQL Server [Електронний ресурс] – Режим доступу :

https://uk.wikipedia.org/wiki/Microsoft_SQL_Server

13. Все що потрібно знати про бази даних: MySQL [Електронний ресурс] – Режим доступу : <https://dan-it.com.ua/uk/blog/vse-shho-potribno-znati-pro-bazi-danih-dlja-pochatkivciv-mysql-postgresql-mongodb/>

14. Entity Framework [Електронний ресурс] – Режим доступу : https://uk.wikipedia.org/wiki/Entity_Framework

15. <https://en.wikipedia.org/wiki/NHibernate>

16. Model-View-ViewModel Pattern [Електронний ресурс] – Режим доступу : <https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>

17. Що таке MVC патерн для програмування [Електронний ресурс] – Режим доступу : <https://blog.ithillel.ua/ru/articles/building-dynamic-web-applications-using-mvc>

18. Розробка вебсайтів MVP [Електронний ресурс] – Режим доступу : <https://brander.ua/technologies/mvp>

19. Figma [Електронний ресурс] – Режим доступу : <https://uk.wikipedia.org/wiki/Figma>

20. Adobe XD [Електронний ресурс] – Режим доступу : https://en.wikipedia.org/wiki/Adobe_XD

ДОДАТОК А
Текст програми

Фрагменты коду вебзастосунку.

Код файлу «Program.cs»:

```
using Newtonsoft.Json;
using PlanifyWeb.Domain;
using PlanifyWeb.Domain.Entities;
using PlanifyWeb.Helpers;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddTransient<PlanifyDbContext>();
builder.Services.AddTransient<Repository>();

var app = builder.Build();

app.MapGet("/", async (context) =>
{
    await context.Response.WriteAsync($"Commands menu:\n" +
        $"\"/registration\" +           // done
        $"\"/login\" +                 // done
        $"\"/getUsers\" +               // done
        $"\"/getCategories\" +         // done
        $"\"/getWeekTypes\" +         // done
        $"\"/addEvent\" +               // done
        $"\"/getEvents\" +             // done
        $"\"/editEvent\" +             // done
        $"\"/addNote\" +               // done
        $"\"/getNotes\" +             // done
        $"\"/editNote\"");
});

app.MapPost("/registration", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var user = await context.Request.ReadFromJsonAsync<User>();
    repository.AddUser(user);

    await context.Response.WriteAsync(user.Id.ToString());
});
app.MapPost("/login", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var user = await context.Request.ReadFromJsonAsync<User>();

    await
context.Response.WriteAsync(JsonConvert.SerializeObject(repository.CheckUser(
user)));
});

app.MapGet("/getUsers", () =>
{
    var repository = app.Services.GetService<Repository>();
    var users = repository.GetUsers();
    return JsonConvert.SerializeObject(users);
});
app.MapGet("/getCategories", () =>
{
```

```

        var repository = app.Services.GetService<Repository>();
        var categ = repository.GetCategories();
        return JsonConvert.SerializeObject(categ);
    });
app.MapGet("/getWeekType", () =>
{
    var repository = app.Services.GetService<Repository>();
    var types = repository.GetWeekTypes();
    return JsonConvert.SerializeObject(types);
});

app.MapPost("/addEvent", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var ev = await context.Request.ReadFromJsonAsync<Event>();

    repository.AddEvent(ev);
});
app.MapGet("/getEvents", async (context) =>
{
    var repository = app.Services.GetService<Repository>();
    User user = new User();
    user.Id = int.Parse(context.Request.Query[Constants.ID]);

    var ev = repository.GetEvents(user);
    await context.Response.WriteAsync(JsonConvert.SerializeObject(ev));
});
app.MapPost("/editEvent", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var newEvent = await context.Request.ReadFromJsonAsync<Event>();

    repository.EditEvent(newEvent);
});

app.MapPost("/addNote", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var note = await context.Request.ReadFromJsonAsync<Note>();

    repository.AddNote(note);
});
app.MapGet("/getNotes", async (context) =>
{
    var repository = app.Services.GetService<Repository>();
    User user = new User();
    user.Id = int.Parse(context.Request.Query[Constants.ID]);

    var notes = repository.GetNotes(user);
    await
context.Response.WriteAsync(JsonConvert.SerializeObject(notes));
});
app.MapPost("/editNote", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var newNote = await context.Request.ReadFromJsonAsync<Note>();

    repository.EditNote(newNote);
});

```

```

});

app.MapPost("/addTimeTable", async (context) =>
{
    var repository = app.Services.GetService<Repository>();

    var res = await context.Request.ReadFromJsonAsync<TimeTable>();

    repository.AddTimeTable(res);
});
app.MapGet("/getTimeTable", async (context) =>
{
    var repository = app.Services.GetService<Repository>();
    int id = int.Parse(context.Request.Query[Constants.ID]);
    int typeId = int.Parse(context.Request.Query[Constants.WEEKTYPE]);

    var timeTable = repository.GetTimeTable(id, typeId);
    await
context.Response.WriteAsync(JsonConvert.SerializeObject(timeTable));
});

app.Run();

```

Код файла «Repository.cs»:

```

using Microsoft.EntityFrameworkCore;
using PlanifyWeb.Domain.Entities;

namespace PlanifyWeb.Domain
{
    public class Repository
    {
        private readonly PlanifyDBContext dbContext;
        public Repository(PlanifyDBContext dbContext)
        {
            dbContext = dbContext;
        }
        public void AddUser(User user)
        {
            if (!dbContext.Users.Contains(user))
            {
                dbContext.Users.Add(user);
            }
            dbContext.SaveChanges();
        }
        public User CheckUser(User user)
        {
            var chUser = dbContext.Users.FirstOrDefault(x => x.Login ==
user.Login && x.Password == user.Password);
            dbContext.Entry(chUser).Collection(x => x.Events).Load();
            dbContext.Entry(chUser).Collection(x => x.Notes).Load();

            chUser.Events = new(chUser.Events);
            chUser.Notes = new(chUser.Notes);

            chUser.Events.ForEach(y => y.User = null);
            chUser.Notes.ForEach(y => y.User = null);
            return chUser;
        }

        public List<User> GetUsers()
        {
            var users = dbContext.Users;

```

```

        users.ForEachAsync(u => dbContext.Entry(u).Collection(x =>
x.Events).Load());
        users.ForEachAsync(u => dbContext.Entry(u).Collection(x =>
x.Notes).Load());
        var tmp = users.ToList();
        tmp.ForEach(x => x.Events.ForEach(y => y.User = null));
        tmp.ForEach(x => x.Notes.ForEach(y => y.User = null));
        return tmp;
    }
    public List<Category> GetCategories()
    {
        return dbContext.Categories.ToList();
    }
    public List<WeekType> GetWeekTypes()
    {
        return dbContext.WeekTypes.ToList();
    }

    public void AddEvent(Event newEvent)
    {
        dbContext.Events.Add(newEvent);
        dbContext.SaveChanges();
    }
    public List<Event> GetEvents(User user)
    {
        return dbContext.Events.Where(x => x.UserId ==
user.Id).ToList();
    }
    public void EditEvent(Event newEvent)
    {
        Event fEv = dbContext.Events.FirstOrDefault(x=> x.Id ==
newEvent.Id);

        Event nEv = new Event()
        {
            UserId = newEvent.UserId,
            Name = newEvent.Name,
            CategoryId = newEvent.CategoryId,
            FinishDate = newEvent.FinishDate,
            IsComplited = newEvent.IsComplited
        };
        if (fEv != null)
        {
            dbContext.Events.Remove(fEv);
            dbContext.Events.Add(nEv);
            dbContext.SaveChanges();
        }
    }

    public void AddTimeTable(TimeTable table)
    {
        dbContext.TimeTables.Add(table);
        dbContext.SaveChanges();

        int id = dbContext.TimeTables.FirstOrDefault(x=> x.UserId ==
table.UserId).Id;

        for (int i = 0; i < table.TableItems.Count; i++)
        {
            TableItem t = new TableItem();
            TableItem tab = table.TableItems[i];

            t.TimeTableId = id;

            t.Day = tab.Day;

```

```

        t.Number = tab.Number;
        t.Name = tab.Name;
        t.Start = tab.Start;
        t.End = tab.End;

        dbContext.Add(t);
        dbContext.SaveChanges();
    }
    dbContext.SaveChanges();
}

public List<TimeTable> GetTimeTable(int id, int typeId)
{
    var tt = dbContext.TimeTables.Where(x => x.UserId == id &&
x.WeekTypeId == typeId).ToList();
    if(tt == null)
    {
        tt = new List<TimeTable>();
        for(int i = 0; i < 6; i++)
        {
            tt.Add(new TimeTable()
            {
                UserId = id,
                WeekTypeId = typeId,
                Day = i.ToString()
            });
            dbContext.Add(tt);
        }
    }
    tt = dbContext.TimeTables.Where(x => x.UserId == id && x.WeekTypeId ==
typeId).ToList();

    for (int i = 0; i < 6; i++)
    {
        List<TableItem> items = dbContext.TableItems.Where(x =>
x.TimeTableId == tt[i].Id && x.Day == tt[i].Day).ToList();
        if (items == null)
        {
            for(int j = 1; j <= 8; j++)
            {
                items.Add(new TableItem
                {
                    Number = j,
                    Name = "Предмет",
                    Start = DateTime.Now,
                    End = DateTime.Now,
                    Day = j.ToString(),
                    TimeTableId = tt[i].Id,
                    WeekTypeId = tt[i].WeekTypeId
                });
            }
        }
        tt.ForEach(x=> x.TableItems = items);
    }
    return tt;
}

public void EditTimeTable(TimeTable oldTT, TimeTable newTT)
{
    TimeTable fTT = dbContext.TimeTables.FirstOrDefault(x =>
x.Id == oldTT.Id);
    if (fTT != null)
    {
        dbContext.TimeTables.Remove(fTT);
        dbContext.TimeTables.Add(newTT);
        dbContext.SaveChanges();
    }
}

```

```

    }
}

public void AddNote(Note note)
{
    dbContext.Notes.Add(note);
    dbContext.SaveChanges();
}

public List<Note> GetNotes(User user)
{
    return dbContext.Notes.Where(x=> x.UserId ==
user.Id).ToList();
}

public void EditNote(Note newNote)
{
    Note fNote = dbContext.Notes.FirstOrDefault(x=> x.Id ==
newNote.Id);

    Note nNote = new Note()
    {
        UserId = newNote.UserId,
        Name = newNote.Name,
        Text = newNote.Text
    };
    if (fNote != null)
    {
        dbContext.Notes.Remove(fNote);
        dbContext.Notes.Add(nNote);
        dbContext.SaveChanges();
    }
}
}
}

```

Фрагменти коду мобільного застосунку:

Код файлу «RegistrationPageViewModel.cs»:

```

using Acr.UserDialogs;
using Acr.UserDialogs.Infrastructure;
using PlanifyMobile.Services;
using PlanifyMobile.Services.Entities;
using PlanifyMobile.Views;
using Prism.Commands;
using Prism.Navigation;
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Json;
using System.Text;
using System.Windows.Input;

namespace PlanifyMobile.ViewModels
{
    class RegistrationPageViewModel : ViewModelBase
    {
        public RegistrationPageViewModel(INavigationService
navigationService) : base(navigationService)
        {
        }

        private string name;
        public string Name
        {
            get => name;

```

```

        set => SetProperty(ref name, value);
    }

    private string email;
    public string Email
    {
        get => email;
        set => SetProperty(ref email, value);
    }

    private bool student;
    public bool Student
    {
        get => student;
        set => SetProperty(ref student, value);
    }

    private bool user;
    public bool User
    {
        get => user;
        set => SetProperty(ref user, value);
    }

    private string pass;
    public string Pass
    {
        get => pass;
        set => SetProperty(ref pass, value);
    }

    private string pass2;
    public string Pass2
    {
        get => pass2;
        set => SetProperty(ref pass2, value);
    }

    private ICommand registration;
    public ICommand Registration
    {
        get => registration ??= new
DelegateCommand(RegistrationCommand);
    }

    private async void RegistrationCommand()
    {
        if(Pass != Pass2)
        {
            UserDialogs.Instance.Alert("Passwords are different!",
"Passwords ERROR", "Ok");
            Pass2 = "";
            return;
        }
        string uType = "";
        if (Student)
        {
            uType = "Student";
        }
        else
        {
            uType = "User";
        }
        using (HttpClient client = new HttpClient())
        {

```

```

        User user = new User()
        {
            Id = 0,
            Name = Name,
            Login = Email,
            Password = Pass,
            Type = uType,
            Events = new List<Event>(),
            Notes = new List<Note>()
        };

        string url =
        $"{Constants.APIConnectionHTTP}/registration";

        JsonContent content = JsonContent.Create(user);
        var response = await client.PostAsync(url, content);
        Constants.User = await
        response.Content.ReadFromJsonAsync<User>();

        string id = await client.GetStringAsync(url);
    }
    await
    NavigationService.NavigateAsync(nameof(AuthorizationPageView));
}
}
}

```

Код файлу «RegistrationPage.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"
    xmlns:viewmodels="clr-namespace:PlanifyMobile.ViewModels"
    x:DataType="viewmodels:RegistrationPageViewModel"
    x:Class="PlanifyMobile.Views.RegistrationPageView">
    <ContentPage.Content>

        <Grid RowDefinitions="*, 40, 60, 40, 60, 50, 40, 60, 40, 60, *"
            ColumnDefinitions="*, *"
            BackgroundColor="#E5E3E4"
            Padding="0, 20, 0, 0">

            <Frame Grid.Row="0"
                Grid.ColumnSpan="2"
                HorizontalOptions="Center"
                VerticalOptions="Center"
                BackgroundColor="#E5E3E4"
                CornerRadius="20">
                <Image Source="logo.png"
                    BackgroundColor="#E5E3E4"/>
            </Frame>

            <Label Grid.Row="1"
                Grid.ColumnSpan="2"
                Text="Имя"
                FontSize="26"
                FontAttributes="Bold"
                TextColor="#000000"
                Margin="50, 10, 10, 0"/>
            <Entry Grid.Row="2"
                Grid.ColumnSpan="2"
                Text="{Binding Name}"
                FontSize="25"
                TextColor="#000000"

```

```

        Margin="40, 0, 40, 15"/>

<Label Grid.Row="3"
        Grid.ColumnSpan="2"
        Text="Повнота"
        FontSize="26"
        FontAttributes="Bold"
        TextColor="#000000"
        Margin="50, 10, 10, 0"/>
<Entry Grid.Row="4"
        Grid.ColumnSpan="2"
        Margin="40, 0, 40, 15"
        FontSize="25"
        TextColor="#000000"
        Text="{Binding Email}"/>

<RadioButton Grid.Row="5"
        Grid.Column="0"
        Content="Студент"
        FontSize="23"
        FontAttributes="Bold"
        TextColor="#000000"
        Margin="50, 0, 0, 20"
        IsChecked="{Binding Student}"/>

<RadioButton Grid.Row="5"
        Grid.Column="1"
        Content="Користувач"
        FontSize="23"
        FontAttributes="Bold"
        TextColor="#000000"
        Margin="5, 0, 0, 20"
        IsChecked="{Binding User}"/>

<Label Grid.Row="6"
        Grid.ColumnSpan="2"
        Text="Пароль"
        FontSize="26"
        FontAttributes="Bold"
        TextColor="#000000"
        Margin="50, 10, 10, 0"/>
<Entry Grid.Row="7"
        Grid.ColumnSpan="2"
        Margin="40, 0, 40, 15"
        FontSize="25"
        TextColor="#000000"
        Text="{Binding Pass}"/>

<Label Grid.Row="8"
        Grid.ColumnSpan="2"
        Text="Підтвердження паролю"
        FontSize="26"
        FontAttributes="Bold"
        TextColor="#000000"
        Margin="50, 10, 10, 0"/>
<Entry Grid.Row="9"
        Grid.ColumnSpan="2"
        Text="{Binding Pass2}"
        FontSize="25"
        TextColor="#000000"
        Margin="40, 0, 40, 15"/>
<Button Grid.Row="10"
        Grid.ColumnSpan="2"

```

```

        Text="Зареєструватись"
        FontSize="20"
        FontAttributes="Bold"
        TextColor="#000000"
        HorizontalOptions="Center"
        VerticalOptions="Start"
        BackgroundColor="#469597"
        CornerRadius="40"
        Command="{Binding Registration}"/>

    </Grid>

</ContentPage.Content>
</ContentPage>

```

Код файлу «EventsPageViewModel.xaml»:

```

using Acr.UserDialogs;
using PlanifyMobile.Services;
using PlanifyMobile.Services.Entities;
using PlanifyMobile.Views;
using Prism.Commands;
using Prism.Navigation;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Json;
using System.Windows.Input;

namespace PlanifyMobile.ViewModels
{
    class EventsPageViewModel : ViewModelBase
    {
        public EventsPageViewModel(INavigationService
navigationService) : base(navigationService)
        {
        }
        public override void Initialize(INavigationParameters
parameters)
        {
            CategoriesBtn = Constants.Categories.Select(x=> new
CategoryBtn { Category = x, AddCategoriesComm = AddCategoriesComm}).ToList();

            Constants.User.Events.ForEach(x => x.Category =
Constants.Categories.FirstOrDefault(y => y.Id == x.CategoryId));

            TEvents = Constants.User.Events.Where(x=> x.FinishDate.Date
== DateTime.Now.Date && x.IsComplited == false).ToList();
            FEvents = Constants.User.Events.Where(x=> x.FinishDate.Date
!= DateTime.Now.Date && x.IsComplited == false).ToList();

            TEvents.ForEach(x => x.EventComm = new
DelegateCommand<Event>(CompliteClickAsync));
            FEvents.ForEach(x => x.EventComm = new
DelegateCommand<Event>(CompliteClickAsync));
        }

        public override void OnNavigatedTo(INavigationParameters
parameters)
        {
            Constants.User.Events.ForEach(x => x.Category =
Constants.Categories.FirstOrDefault(y => y.Id == x.CategoryId));

```

```

        TEvents = Constants.User.Events.Where(x =>
x.FinishDate.Date == DateTime.Now.Date && x.IsComplited == false).ToList();
        FEvents = Constants.User.Events.Where(x =>
x.FinishDate.Date != DateTime.Now.Date && x.IsComplited == false).ToList();

        TEvents.ForEach(x => x.EventComm = new
DelegateCommand<Event>(CompliteClickAsync));
        FEvents.ForEach(x => x.EventComm = new
DelegateCommand<Event>(CompliteClickAsync));
    }

    // --- Categories ---

    private List<CategoryBtn> categoriesBtn = new
List<CategoryBtn>();
    public List<CategoryBtn> CategoriesBtn
    {
        get => categoriesBtn;
        set => SetProperty(ref categoriesBtn, value);
    }
    private ICommand addCategoriesComm;
    public ICommand AddCategoriesComm
    {
        get => addCategoriesComm ??= new
DelegateCommand(CategoriesClick);
    }

    private void CategoriesClick()
    {
        UserDialogs.Instance.Alert("Category", "Cat", "OK");
    }

    // --- Events ---

    private List<Event> tEvents;
    public List<Event> TEvents
    {
        get => tEvents ??= new List<Event>();
        set => SetProperty(ref tEvents, value);
    }

    private List<Event> fEvents = new List<Event>();
    public List<Event> FEvents
    {
        get => fEvents ??= new List<Event>();
        set => SetProperty(ref fEvents, value);
    }

    private ICommand eventComm;
    public ICommand EventComm
    {
        get => eventComm ??= new
DelegateCommand<Event>(CompliteClickAsync);
    }

    private async void CompliteClickAsync(Event ev)
    {
        ev.IsComplited = true;

        using (HttpClient client = new HttpClient())
        {
            string url = $"{Constants.APIIconectionHTTP}/editEvent";

```

```

        JsonConvert content = JsonConvert.Create(ev);
        var response = await client.PostAsync(url, content);
    }
    TEvents = Constants.User.Events.Where(x =>
x.FinishDate.Date == DateTime.Now.Date && x.IsComplited == false).ToList();
    FEvents = Constants.User.Events.Where(x =>
x.FinishDate.Date != DateTime.Now.Date && x.IsComplited == false).ToList();

    TEEvents.ForEach(x => x.EventComm = new
DelegateCommand<Event>(CompliteClickAsync));
    FEEvents.ForEach(x => x.EventComm = new
DelegateCommand<Event>(CompliteClickAsync));
    }

    // --- Add event ---
    private ICommand addEvent;
    public ICommand AddEvent
    {
        get => addEvent ??= new DelegateCommand(AddEventClick);
    }
    private async void AddEventClick()
    {
        await
NavigationService.NavigateAsync(nameof(AddEventPageView));
    }

    // --- Navigation buttons ---

    private ICommand timeTable;
    public ICommand TimeTable
    {
        get => timeTable ??= new DelegateCommand(TimeTableClick);
    }
    private ICommand notes;
    public ICommand Notes
    {
        get => notes ??= new DelegateCommand(NotesClick);
    }
    private ICommand profile;
    public ICommand Profile
    {
        get => profile ??= new DelegateCommand(ProfileClick);
    }

    private async void TimeTableClick()
    {
        await
NavigationService.NavigateAsync(nameof(TimeTablePageView));
    }
    private async void NotesClick()
    {
        await
NavigationService.NavigateAsync(nameof(NotesPageView));
    }
    private async void ProfileClick()
    {
        await
NavigationService.NavigateAsync(nameof(ProfilePageView));
    }
    }
}

```

Код файлу «EventsPage.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"
    xmlns:viewmodels="clr-namespace:PlanifyMobile.ViewModels"

    xmlns:extensions="http://xamarin.com/schemas/2020/toolkit"
    xmlns:services="clr-namespace:PlanifyMobile.Services"
    xmlns:entities="clr-
namespace:PlanifyMobile.Services.Entities"
    x:DataType="viewmodels:EventsPageViewModel"
    x:Class="PlanifyMobile.Views.EventsPageView">
    <ContentPage.Content>

        <Grid RowDefinitions="50, Auto, 50, Auto, 50, 50"
            ColumnDefinitions="*, *, *, *"
            BackgroundColor="#E5E3E4">

            <Label Grid.Row="0"
                Grid.ColumnSpan="4"
                Text="Сьогодні"
                FontSize="25"
                FontAttributes="Bold"
                TextColor="#000000"
                HorizontalOptions="Center"
                VerticalOptions="Center"/>
            <ListView ItemsSource="{Binding TEvents}"
                Grid.Row="1"
                Grid.ColumnSpan="4">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <ViewCell.View>
                                <Grid ColumnDefinitions="Auto, *, *,
Auto">

                                    <CheckBox Grid.Column="0"

x:DataType="entities:Event"

                                        IsChecked="False"
                                        VerticalOptions="Center"
                                        HorizontalOptions="End"
                                        Color="#000000">
                                    <CheckBox.Behaviors>

<extensions:EventToCommandBehavior EventName="CheckedChanged"

Command="{Binding EventComm}"

CommandParameter="{Binding .}"/>

                                </CheckBox.Behaviors>
                                </CheckBox>
                                <Label Grid.Column="1"
                                    x:DataType="entities:Event"
                                    Text="{Binding Name}"
                                    FontSize="14"
                                    TextColor="#000000"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Start"/>
                                <Label Grid.Column="2"
                                    x:DataType="entities:Event"
                                    Text="{Binding
Category.Name}"

                                    FontSize="14"

```

```

        TextColor="#000000"
        VerticalOptions="Center"
        HorizontalOptions="End"/>
<Label Grid.Column="3"
        x:DataType="entities:Event"
        Text="{Binding FinishDate,
StringFormat='{{0:dd\\\\.MM\\\\.yyyy\\, hh\\\:mm tt}}'"
        FontSize="14"
        TextColor="#000000"
        VerticalOptions="Center"
        HorizontalOptions="End"/>
    </Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>

<Label Grid.Row="2"
        Grid.ColumnSpan="4"
        Text="Майбыте"
        FontSize="25"
        FontAttributes="Bold"
        TextColor="#000000"
        HorizontalOptions="Center"
        VerticalOptions="Center"/>
<ListView ItemsSource="{Binding FEvents}"
        Grid.Row="3"
        Grid.ColumnSpan="4">
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <ViewCell.View>
                    <Grid ColumnDefinitions="Auto, *, *,
Auto">
                        <CheckBox Grid.Column="0"
                                x:DataType="entities:Event"
                                IsChecked="False"
                                VerticalOptions="Center"
                                HorizontalOptions="End"
                                Color="#000000">
                            <CheckBox.Behaviors>
                                <extensions:EventToCommandBehavior EventName="CheckedChanged"
                                Command="{Binding EventComm}"
                                CommandParameter="{Binding .}"/>
                            </CheckBox.Behaviors>
                        </CheckBox>
                        <Label Grid.Column="1"
                                x:DataType="entities:Event"
                                Text="{Binding Name}"
                                FontSize="14"
                                TextColor="#000000"
                                VerticalOptions="Center"
                                HorizontalOptions="Start"/>
                        <Label Grid.Column="2"
                                x:DataType="entities:Event"
                                Text="{Binding
Category.Name}"
                                FontSize="14"
                                TextColor="#000000"

```

```

        VerticalOptions="Center"
        HorizontalOptions="End"/>
    <Label Grid.Column="3"
        x:DataType="entities:Event"
        Text="{Binding FinishDate,
StringFormat='{{0:dd\\.MM\\.yyyy\\, hh\\:mm tt}}'"
        FontSize="14"
        TextColor="#000000"
        VerticalOptions="Center"
        HorizontalOptions="End"/>
    </Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>

<Button Grid.Row="4"
    Grid.Column="3"
    Text="+"
    FontSize="35"
    FontAttributes="Bold"
    TextColor="#000000"
    Padding="-20"
    Margin="15, 15, 15, 0"
    CornerRadius="30"
    BackgroundColor="#469597"
    HorizontalOptions="End"
    VerticalOptions="End"
    WidthRequest="50"
    Command="{Binding AddEvent}"/>

<Button Grid.Row="5"
    Grid.Column="0"
    Text="Події"
    FontSize="12"
    FontAttributes="Bold"
    TextColor="#000000"
    HorizontalOptions="Center"
    VerticalOptions="End"
    BackgroundColor="#469597"
    CornerRadius="30"
    Margin="7"/>

<Button Grid.Row="5"
    Grid.Column="1"
    Text="Розклад"
    FontSize="12"
    FontAttributes="Bold"
    TextColor="#000000"
    HorizontalOptions="Center"
    VerticalOptions="End"
    BackgroundColor="#469597"
    CornerRadius="30"
    Margin="7"
    Command="{Binding TimeTable}"/>

<Button Grid.Row="5"
    Grid.Column="2"
    Text="Нотатки"
    FontSize="12"
    FontAttributes="Bold"
    TextColor="#000000"
    HorizontalOptions="Center"
    VerticalOptions="End"
    BackgroundColor="#469597"

```

```

        CornerRadius="30"
        Margin="7"
        Command="{Binding Notes}"/>
<Button Grid.Row="5"
        Grid.Column="3"
        Text="Профіль"
        FontSize="12"
        FontAttributes="Bold"
        TextColor="#000000"
        HorizontalOptions="Center"
        VerticalOptions="End"
        BackgroundColor="#469597"
        CornerRadius="30"
        Margin="7"
        Command="{Binding Profile}"/>
</Grid>

</ContentPage.Content>
</ContentPage>

```

ДОДАТОК Б
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
Кафедра програмних засобів

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЛАНУВАННЯ ТА ОРГАНІЗАЦІЇ СПРАВ PLANIFY

ВИКОНАЛА: СТУДЕНТКА 4 КУРСУ, ГРУПИ КНТ-130 НІКУЛІНА А. С.
КЕРІВНИК: К.Т.Н., ДОЦЕНТ КАПЛІЄНКО Т. І.

1

Рисунок Б.1 – Слайд №1

ЗМІСТ

01

ОБ'ЄКТ, ПРЕДМЕТ, МЕТА І
ЗАДАЧІ РОЗРОБКИ

02

АНАЛІЗ ПРЕДМЕТНОЇ
ОБЛАСТІ

03

ВИБІР ІНСТРУМЕНТІВ
РОЗРОБКИ

04

СТРУКТУРНА ТА
ФУНКЦІОНАЛЬНА СХЕМИ

05

ПРОЄКТУВАННЯ БАЗИ ДАНИХ

06

ОСНОВНІ МЕТОДИ ТА
АЛГОРИТМИ

07

ПРОТОТИПУВАННЯ ІНТЕРФЕЙСІВ
ТА ПРИКЛАДИ ЗАСТОСУВАННЯ

08

ВИСНОВКИ

2

Рисунок Б.2 – Слайд №2

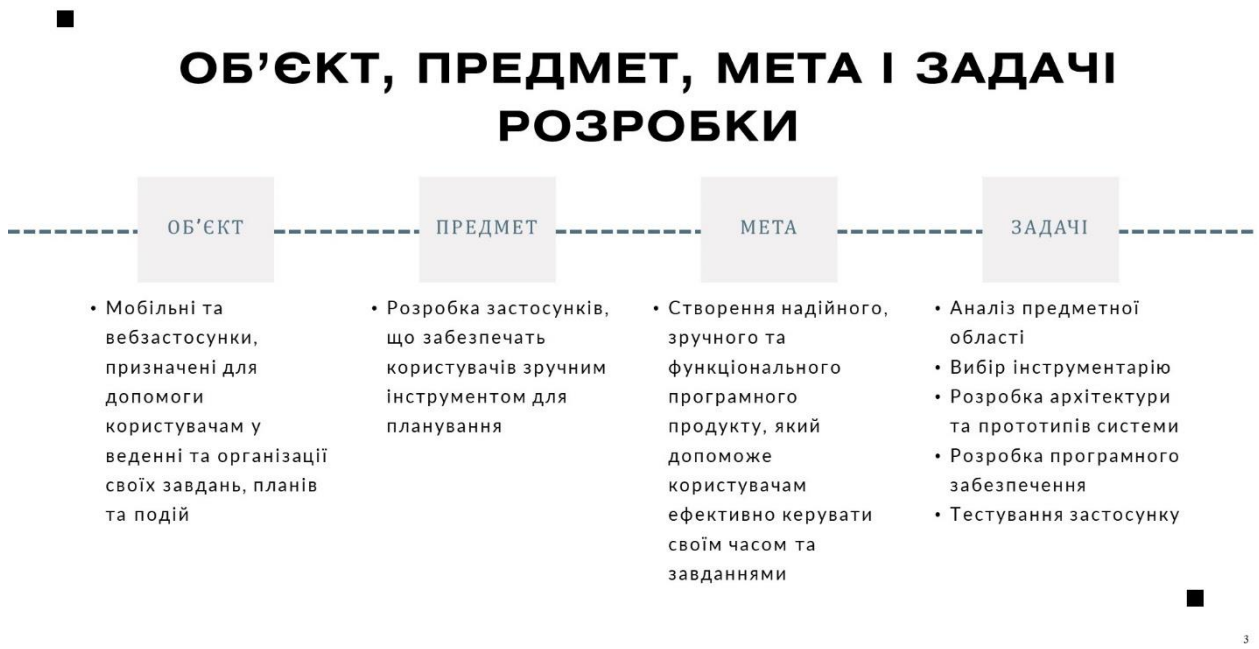
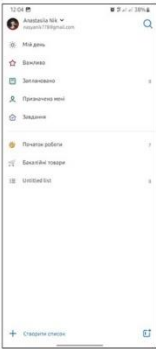


Рисунок Б.3 – Слайд №3

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ



To-do List



Microsoft To-do



Todoist

Порівняльна таблиця програмних продуктів

	To-do List	Microsoft To-Do	Todoist	Planify
Створення подій	+	+	+	+
Створення нотаток	-	-	-	+
Наявність категорій подій	+	-	+	+
Розклад для студентів	-	-	-	+
Статистика виконаних подій	+	-	-	+

Рисунок Б.4 – Слайд №4

ІНСТРУМЕНТАРІЙ РОЗРОБКИ

	C#	Java
Парадигма програмування	Об'єктно-орієнтована, з підтримкою імперативного та функціонального програмування.	Чисто об'єктно-орієнтована мова програмування
Кросплатформність	Застосунки на C# можуть працювати на найрізноманітніших пристроях, таких як мобільний телефон, комп'ютер, ноутбук і сервер.	Java запускає байт-код програми Java у віртуальній машині Java Virtual Machine (JVM) на будь-якій платформі, на якій встановлена програма Java.
Сумісність з середовищами розробки	Сумісна з такими середовищами, як Visual Studio, MonoDevelop, JetBrains Rider.	Сумісна з IntelliJ IDEA, Eclipse, NetBeans.
Бібліотеки	Має велику екосистему .NET з безліччю бібліотек та фреймворків. Наприклад .NET Framework - це платформа від Microsoft, що надає широкий спектр бібліотек та інструментів для розробки на C#.	Має екосистему бібліотек, що робить її однією з найрізноманітніших та найбагатших у світі програмування. Вона має бібліотеки для стандартних задач. Також існує безліч бібліотек сторонніх розробників.
Спільнота	Є спільнота розробників з широкого підтримкою. А також отримує постійні оновлення і підтримку від Microsoft.	Є одною з найбільших та найактивніших спільнот розробників у світі

5

Рисунок Б.5 – Слайд №5

ІНСТРУМЕНТАРІЙ РОЗРОБКИ

	Microsoft Visual Studio	JetBrains Rider
Функціональність	потужне і повнофункціональне середовище розробки, яке підтримує багато мов програмування, засобів для створення вебзастосунків, налаштування інтерфейсу, налагодження та інше. Воно має широкий набір інструментів і доповнень для поліпшення продуктивності розробників.	повнофункціональне середовище розробки, спеціально призначене для розробки на платформі .NET і ряді інших мов програмування. Надає інтегровану підтримку для інструментів від JetBrains, таких як ReSharper, і забезпечує розширену підтримку для функціонального програмування.
Підтримка мов програмування	підтримує широкий спектр мов програмування, включаючи C#, C++, Visual Basic, F#, JavaScript, Python та інші.	підтримує багато мов програмування, зокрема C#, VB.NET, F#, JavaScript, TypeScript, HTML, CSS, SQL, JSON, XML, Markdown і багато інших.
Інтегровані інструменти	має широкий набір інтегрованих інструментів, таких як вбудований редактор коду, дизайнер інтерфейсу, система контролю версій, інструменти налагодження, профілюнгу і багато інших.	також надає інтегровані інструменти для редагування коду, системи контролю версій, налагодження, профілюнгу і тестування, а також додаткові інструменти від JetBrains, такі як ReSharper для покращення продуктивності.
Спільнота	має велику спільноту розробників та активний вебсайт з документацією, уроками та форумами підтримки. Він також отримує постійні оновлення і підтримку від Microsoft.	має активну спільноту розробників і форуми підтримки, а також надає оновлення та підтримку для своїх продуктів.
Вартість	має декілька версій, включаючи безкоштовну Community Edition, яка надає базові можливості для індивідуальних розробників та невеликих команд. Однак, платні версії мають розширені функції і підтримку.	є комерційним продуктом, і для його використання потрібна придбання ліцензії. JetBrains також надає можливість використовувати Rider в рамках їх підписки All Products Pack, яка включає в себе інші інструменти компанії.

6

Рисунок Б.6 – Слайд №6

ІНСТРУМЕНТАРІЙ РОЗРОБКИ



Рисунок Б.7 – Слайд №7

СТРУКТУРНА СХЕМА ЗАСТОСУНКУ

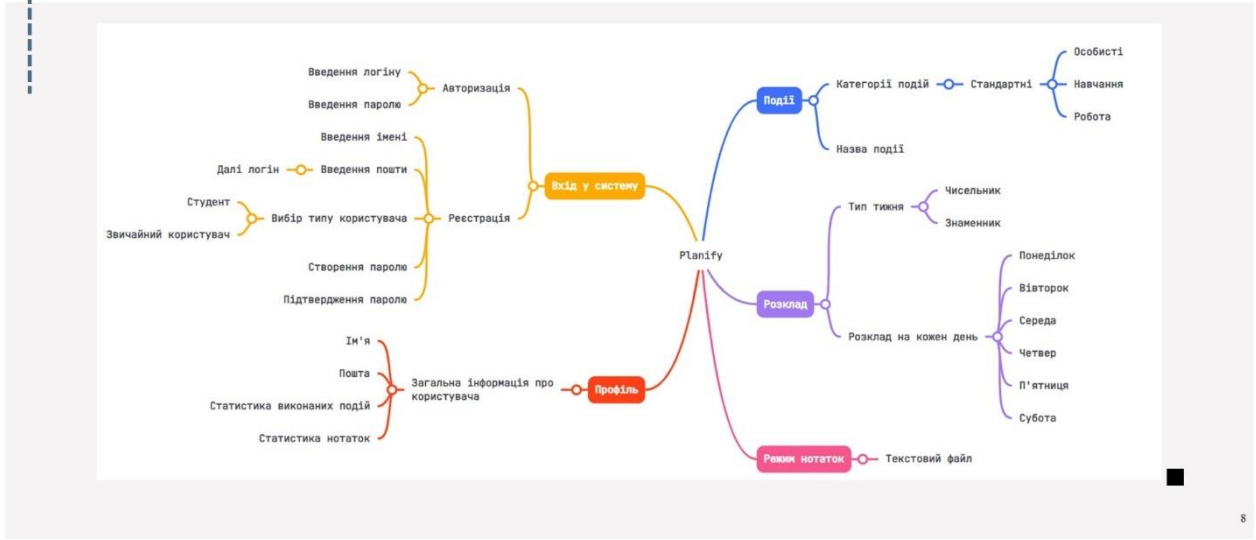


Рисунок Б.8 – Слайд №8

■ ФУНКЦІОНАЛЬНА СХЕМА ЗАСТОСУНКУ

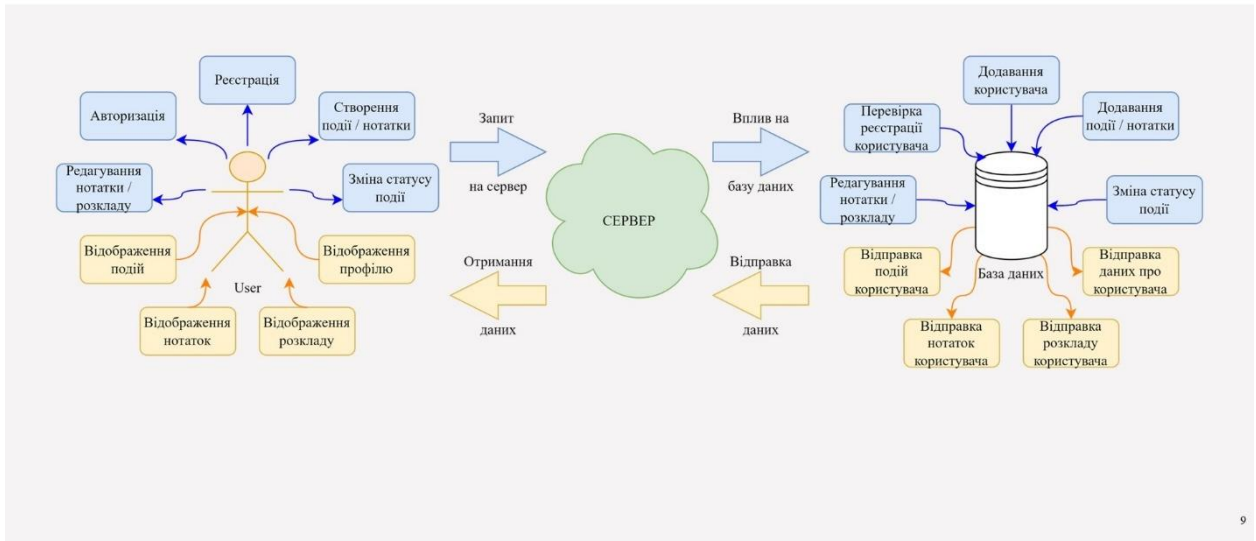


Рисунок Б.9 – Слайд №9

■ ПРОЄКТУВАННЯ БАЗИ ДАНИХ

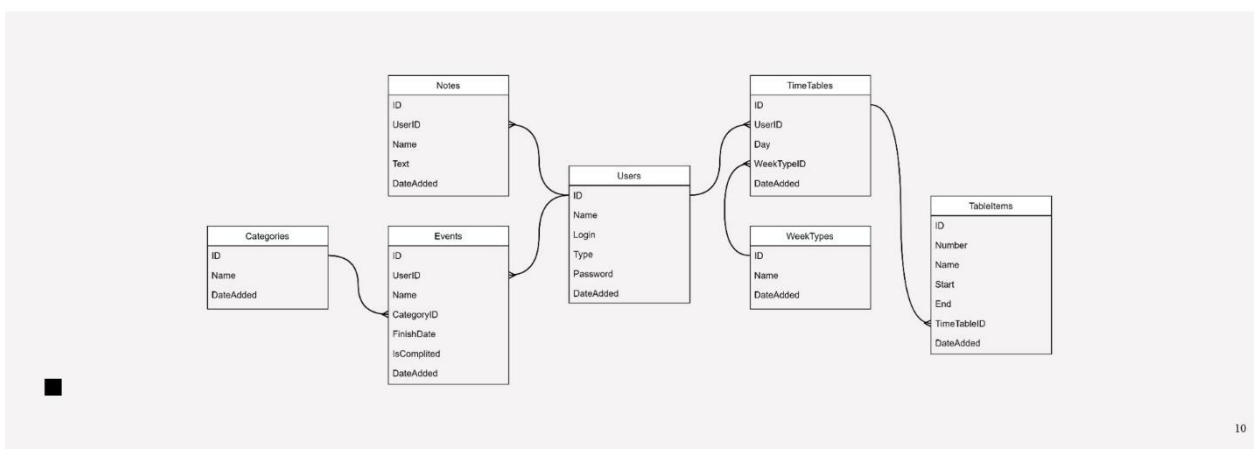
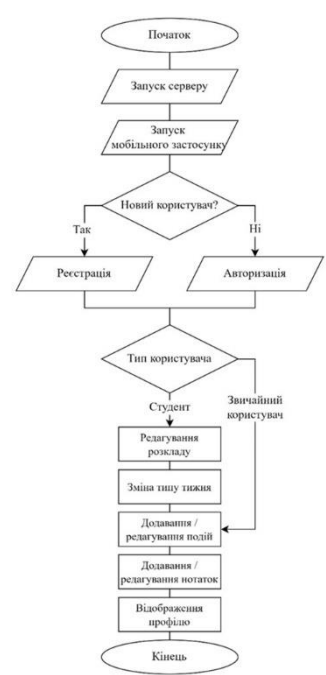


Рисунок Б.10 – Слайд №10



АЛГОРИТМ РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

Рисунок Б.11 – Слайд №11

ОСНОВНІ МЕТОДИ ТА АЛГОРИТМИ

- Події
- Розклад
- Нотатки
- Профіль

ДОДАВАННЯ
Користувачів
Подій
Нотаток
Розкладу

РЕДАГУВАННЯ
Нотаток
Розкладу

ОТРИМАННЯ
Даних про користувача
Подій
Нотаток
Розкладу

```
public void AddUser (User user)
{
    if (!dbContext.Users.Contains(user))
    {
        dbContext.Users.Add(user);
    }
    dbContext.SaveChanges();
}
```

```
public void AddEvent(Event newEvent)
{
    dbContext.Events.Add(newEvent);
    dbContext.SaveChanges();
}
```

Рисунок Б.12 – Слайд №12

ПРОТОТИПУВАННЯ ІНТЕРФЕЙСІВ

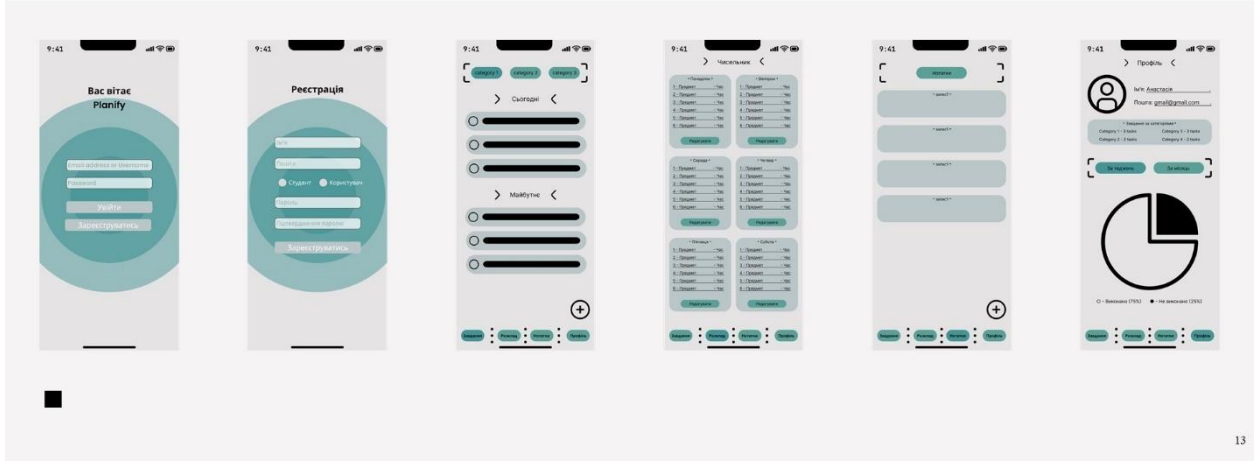


Рисунок Б.13 – Слайд №13

ТЕСТУВАННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

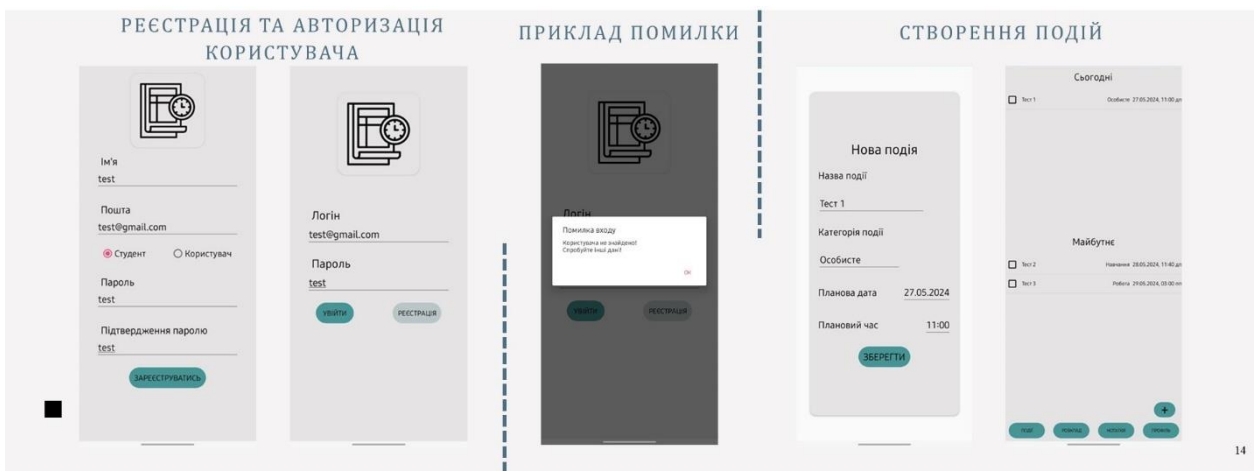
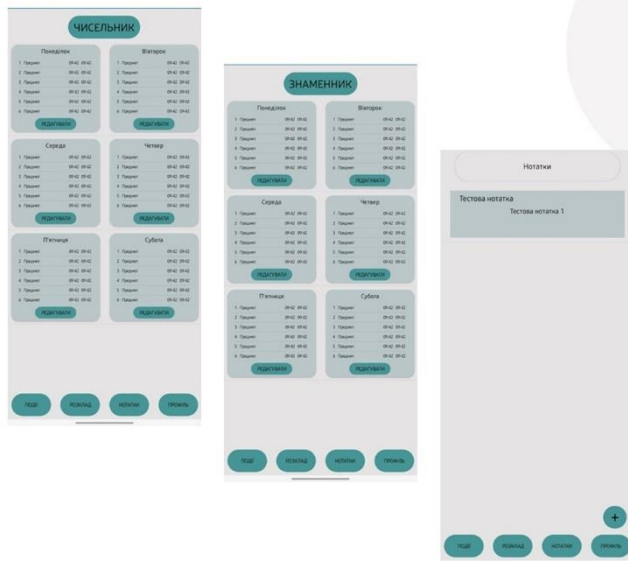


Рисунок Б.14 – Слайд №14

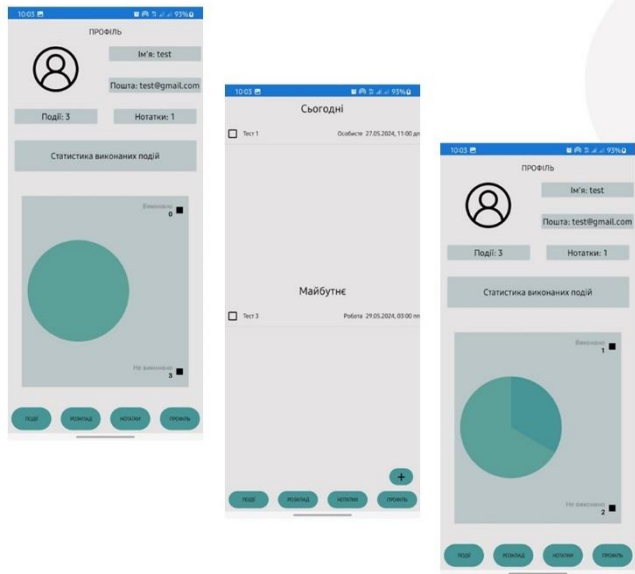


ІНТЕРФЕЙС ТА ПРИКЛАДИ ЗАСТОСУВАННЯ

РОЗКЛАД ТА НОТАТКИ

15

Рисунок Б.15 – Слайд №15



ІНТЕРФЕЙС ТА ПРИКЛАДИ ЗАСТОСУВАННЯ

ДІЇ З ПРОФІЛЕМ

16

Рисунок Б.16 – Слайд №16

ВИСНОВКИ

У ході виконання дипломної роботи, було проведено детальне ознайомлення із предметною областю – планування та організації часу.

На основі поставленого завдання було проведено аналіз предметної області, а саме розглянуто вже існуючі програмні засоби, які вирішують поставлене завдання.

Враховуючи зібрані під час аналізу дані, було обрано інструментарій для розробки ПЗ.

Далі було розроблено архітектуру програми, а саме описано модулі системи, побудовано структурну та функціональну схеми. На основі цих даних були побудовані прототипи інтерфейсів.

Наступним етапом розробки була розробка алгоритмів для реалізації різноманітних компонентів системи, таких як авторизація, реєстрація, відображення, отримання та надсилання даних.

В результаті було розроблено програмний продукт, який надає рішення поставленим задачам.

Тестування показало, що ПЗ є працездатним та ефективним для вирішення даних задач організації даних, а також має важливу перевагу – дані зберігаються на сервері, а отже доступні з будь-якого мобільного пристрою.

В подальшому можливо модернізувати ПЗ, розширивши його можливості та функціонал, аби зробити його більш доступним та зручним у використанні.

Результати роботи апробовані на науково-практичній конференції «Тиждень науки-2024», яка дистанційно проводилася у Національному університеті «Запорізька політехніка» у 2024 році.

Рисунок Б.17 – Слайд №17