НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення веб-технологій та мобільних пристроїв»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Засоби вирішення господарських проблем кампуса КПІ на базі ГІС систем»

Виконав:
студент IV курсу, групи ТІ-62
Гавриляк Олександр Володимирович _____

Керівник:
Доцент, кандидат технічних наук,
Гагарін Олександр Олександрович _____

Рецензент:
Доцент, кандидат технічних наук,
Степанець Олександр Васильович ___ _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Гавриляка Олександра Володимировича
(прізвище, ім’я, по батькові)

1. Тема роботи Засоби вирішення господарських проблем кампуса КПІ на базі ГІС систем

керівник роботи Гагарін Олександр Олександрович, кандидат технічних наук, доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020 р. № **1168-с**

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: мови програмування Dart та C#; фреймворки Flutter, ASP.NET Core та Entity Framework Core.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) сформувати задачі, шо повинна вирішувати система; проаналізувати аналогічні рішення такі як My LA 311, Pakistan Citizen Portal і 2GIS, виділити їх недоліки та переваги; розробити діаграму взаємодії користувача з системою; навести опис взаємодії складових частин системи (сервер, мобільний додаток, база даних, тощо); реалізувати мобільну частину та проілюструвати розроблений користувальницький інтерфейс; створити базу даних та показати її схему на діаграмі; розробити архітектуру всієї системи та її складових; створити архітектуру серверної частини; реалізувати сервер для системи.

5. Перелік ілюстративного матеріалу

багаторівнева архітектура системи «Awesome Map»; схема бази даних; діаграма прецедентів системи; діаграма послідовності; архітектура серверної частини;_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”___”_____2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.10.2019	
2.	Вивчення та аналіз задачі	13.04.2020	
3.	Розробка архітектури та загальної структури системи	20.04.2020	
4.	Розробка структур окремих підсистем	27.04.2020	
5.	Програмна реалізація системи	11.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020	
7.	Захист програмного продукту	26.05.2020	
8.	Передзахист	10.06.2020	
9.	Захист	16.06.2020	

Студент _____ ___Гавриляк О. В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _Гагарін О. О._____
(підпис) (прізвище та ініціали,)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Фреймворк (англіцизм, неологізм від framework - каркас, структура) – програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

Мок (англіцизм, неологізм від Mock) – це модельований об'єкт, що імітує поведінку реальних об'єктів, для контролювання, найчастіше для реалізації тестів в об'єктно-орієнтованому програмуванні.

Ahead-of-time (AOT) компілювання – це процес компілювання в високорівневих мовах програмування, таких як C, C++ тощо, в машинний код таким чином, щоб бінарний файл міг виконатися від самого початку.

Компонент – незалежний модуль вихідного коду програми, призначений для повторного використання і розгортання. На фреймворкі Flutter, компонетном виступає Widget (віджет) – базовий клас, який має свій стан, зовнішній вигляд (користувальницький інтерфейс) та поведінку.

CodePen – це середовище соціального розвитку для провідних дизайнерів та розробників, що дозволяє створити і розгорнути веб-застосунок, показати свою роботу, створити тестовий приклад.

Діджіталізація – процес цифрової трансформації суспільства.

Інтегроване середовище розробки (IDE) – це програмне забезпечення, яке надає комплексні засоби комп'ютерним програмістам для розробки програмного забезпечення. IDE зазвичай складається щонайменше з редактора вихідного коду, засобів автоматизації побудови та налагоджувача.

Автентифікації – процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора.

Авторизація - це функція визначення прав доступу / привілеїв до ресурсів, яка пов'язана з інформаційною безпекою та безпекою комп'ютерів загалом та зокрема з контролем доступу.

АНОТАЦІЯ

Мета роботи: розробити систему «Awesome Map», яка вирішить моніторинг господарських проблеми та публічних заходів промислового району для ефективного та вдалого створення управлінських рішень.

За допомогою аналізу аналогічних рішень та виділення їх недоліків і переваг, були створені задачі й вимоги, яким повинна відповідати система.

Обсяг дипломної роботи складає 65 сторінок, 27 рисунків, 1 таблиця, 14 використаних джерел літератури, та 3 додатка.

Реалізована система та описано весь користувальницький інтерфейс мобільного додатку, що був розбитий на головні функціональні компоненти. Виділено багато цікавих моментів дизайну та обґрунтовано їх доцільне використання. Також описана та реалізована багаторівнева архітектура для всієї системи, дозволяючи збудувати гнучкий продукт, де легко видалити та замінити одну з частин системи на іншу.

Ключові слова: GIS рішення, GIS технології, багаторівнева архітектура, мобільний додаток, сучасний дизайн мобільного додатку, REST – архітектура, компонента архітектура.

ABSTRACT

Purpose: to develop a system "Awesome Map", which will solve the monitoring of economic problems and public events of the industrial area for the effective and successful creation of management decisions.

By analyzing similar solutions and highlighting their disadvantages and advantages, the tasks and requirements that the system must meet were created.

The volume of the thesis is 65 pages, 27 figures, 1 table, 14 references, and 3 appendices.

The system is implemented and the whole user interface of the mobile application is described, which was divided into the main functional components. Many interesting design points are highlighted and their expedient use is substantiated. A multi-level architecture for the entire system is also described and implemented, allowing a flexible product to be built where it is easy to remove and replace one part of the system with another.

Key words: GIS solution, GIS technologies, multilevel architecture, mobile application, modern design of mobile application, REST - architecture, component architecture.

ВСТУП

В наш час діджеталізації створюється потреба в моніторингу господарських проблем та публічних заходів промислового району для ефективного та вдалого створення управлінських рішень. Системи, що вирішують зазначену проблему все частіше реалізуються у вигляді сервісних центрів з віддаленим доступом та спеціалізованими мобільними додатками.

Пропонована система «Awesome Map» вирішить як муніципальні проблеми університету, наприклад повний бак сміття біля гуртожитку, так і життєво-небезпечні проблеми забезпечення порядку та закону.

Система, що розробляється має клієнт-серверну архітектуру та вирішує задачі моніторингу наявних господарських проблем (прорив водо-,тепло-, електричних мереж, наявність незручностей, сміття та таке інше), а також планування та проведення публічних заходів на території університету. Серверна частина організує ведення бази даних проблем що виникають, їх фіксацію та доступ до інформації необхідної для вироблення управлінських завдань.

Збір проблемної інформації та зведення її до єдиного сервісного центру виконується користувачами системи за допомогою мобільного додатку.

Подібні мобільні додатки починають розроблятися не тільки в Україні, а й по всьому світу, наприклад додаток для Los Angeles «MyLA311»[1] або для всього Пакистану – «Pakistan Citizen Portal»[2], або звичайна зручна карта мобільного додатка «2GIS» [3]. Всі додатки намагаються вирішувати свої поставлені проблеми за допомогою GIS (Geographic Information System) технологій. На жаль, кожен з них має свої недоліки, але, на щастя, має і свої переваги. За допомогою аналізу існуючих рішень, впроваджена система розроблялась з урахування минулого досвіду, таким чином, вона позбулась найпоширеніших недоліків та підкреслила свої унікальні переваги.

Система розроблялась на базі світових технологій: Google Cloud Platform, сучасного інструмента користувальницького інтерфейсу для мобільних додатків від Google – Flutter, та останніх технологій від Microsoft, для створення серверної частини системи – ASP.NET Core.

Бета версія системи та апробація всіх її компонентів на даний момент завершені. З повною версією опису системи можна ознайомитися на сайті apeps.kpi.ua у розділі студентські випускні роботи.

1. ПОСТАНОВКА ЗАДАЧІ

1.1 Завдання роботи

Створено список завдань, що повинні бути вирішенні:

1. Розробити основні задачі для системи, впровадити завдання та вимоги для мобільної й серверної частин.
2. Проаналізувати існуючі аналогічні рішення.
3. Обрати та описати засоби розробки кожної з частин системи.
4. Описати взаємодію користувача з системою на діаграмі прецедентів.
5. Надати опис програмної реалізації продукту.
6. Показати взаємодію частин системи та потоку даних між ними в часі на прикладі діаграми послідовності.
7. Створити схему бази даних системи.

1.2 Основні задачі системи

Система повинна вирішувати задачі моніторингу наявних господарських проблем, (прорив водо-,тепло-, електричних мереж, наявність незручностей, сміття та таке інше), а також планування та проведення публічних заходів на території кампуса університету.

Адміністрація НТУУ «КПІ» ім. Ігоря Сікорського повинна мати змогу відредагувати проблему або захід відповідно до робочих процесів системи.

1.3 Задачі для мобільного додатку

1.3.1 Загальні вимоги

Мобільній додаток повинен відповідати рекомендаціям дизайну для Android та використовувати загальнодоступні значки та шаблони інтерфейсу користувача.

У додатку повинні бути не перевизначені звичні функції системних значків (наприклад кнопка «Назад»).

У додатку не відбувається зміна системних значків абсолютно новими значками, якщо використовуються стандартні функції інтерфейсу користувача.

Мобільний додаток повинен бути адаптованим для використання його на різних пристроях з різними розмірами екранів.

Якщо в додатку пропонується адаптована версія стандартного системного значка, цей значок повинен в значній мірі нагадувати вихідний системний значок і порядок роботи з ним повинен відповідати поведінки системного значка.

Додаток не має перевизначать або некоректно використовувати стандартні шаблони інтерфейсу користувача Android, наприклад значки або їх дії, щоб не плутати користувачів і не ускладнювати їх роботу.

1.3.2 Головні задачі

Користувач системи повинен мати змогу переглядати існуючі проблеми та заходи на окремих картах і на одній загальній.

В системі повинна бути реалізована фільтрація проблем за наступними показниками:

– назва;

– дата створення (початок проміжку до сьогодні);

– категорія;
– статус проблеми.

та заходів за іншими показниками:

– назва;
– дата проведення з (початок обраного проміжку);
– дата проведення до (кінець обраного проміжку);
– місце;
– категорії.

Доцільно зробити перехід від міток на карті до більш детального списку елементів, відображеного користувачу.

Користувач повинен мати змогу переглянути тільки свої проблеми та заходи, а також відредагувати чи видалити їх, відповідно до робочих процесів системи.

Користувач повинен мати змогу змінити свої особисті та вхідні дані у системі.

1.3.3 Додаткові задачі

Рекомендовано створити більш гнучке налаштування інтерфейсу, а саме: можливість змінити головну мову інтерфейсу з української на англійську, а також можливість змінити тему інтерфейсу зі стандартної (блакитної) на нічну (темну) тему.

1.4 Задачі для сервера

1.4.1 Загальні вимоги

Сервер має опрацьовувати http запити та повертати коректні відповіді.

Робота з даними повинна бути забезпечена за допомогою базових операцій REST - архітектурних додатків. [4] Для операцій з мережевим сховищем впроваджуються запити:

- вилучення даних – GET запит;
- збереження даних – POST запит;
- зміна даних – PUT або PATCH запит;
- видалення – DELETE запит.

1.4.2 Головні задачі

Сервер повинен надавати усі необхідні дані мобільному додатку.

Він відповідальний за авторизацію та реєстрацію користувача в системі. Сервер надає доступ до бази даних, де зберігаються:

- проблеми;
- типи проблем;
- заходи;
- типи заходів;
- коментарі;
- іконки;
- файли і таке інше.

Висновки до розділу

В цьому розділі були розглянуті загальні вимоги до сучасного мобільного додатку. В першій частині впроваджені головні та додаткові вимоги, які повинен

вирішувати мобільний додаток в рамках робочих процесів системи. В другій частині розділу, розглянуті вимоги до архітектури серверної частини та загальна роль сервера в системі «Awesome Map». Описані основні запити до сервера, як:

1. Вилучення даних.
2. Збереження даних.
3. Зміна даних.
4. Видалення даних.

Описані основні сутності, з якими повинен працювати сервер при взаємодії з базою даних.

Також важливою частиною було описати показники фільтрації для головних сутностей системи: проблем та заходів.

2. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ АНАЛОГІЧНИХ РІШЕНЬ

2.1 Система My LA 311

За допомогою системи MyLA311 у Лос-Áнджелесі, штат Каліфорнія, США, можна дізнатися інформацію про місто та послуги лише за декілька хвилин. [1] Використовування цього додатку дає змогу швидкого та легкого доступу до найпопулярніших послуг міста, включаючи видалення графіті, ремонт вибоїн та забір об’ємних предметів.

Роздивимось більш детально мобільний додаток. На першому екрані (Рис 2.1.1) бачимо три пункти:

– створення службового запиту;

– перегляд;

– інші служби.

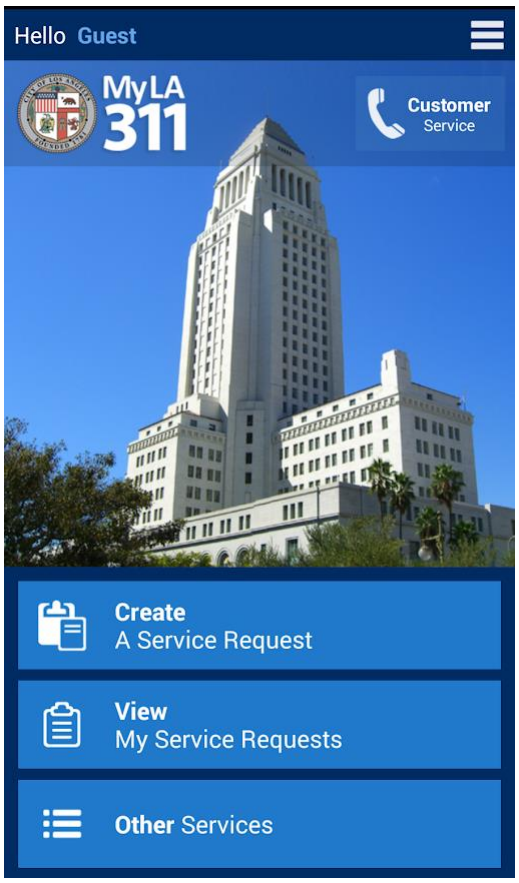

Рисунок 2.1.1 – Головний екран MyLA311

Головним функціоналом кожного з цих пунктів є:

– функція «Створення службового запита», що дозволяє швидко та легко запросити найпопулярніші послуги міста, включаючи видалення графіті, ремонт вибоїн та забір об'ємних предметів;

– функція «Перегляд», яка відображає найближчі парки, бібліотеки, пожежні та поліцейські пункти, громадські басейни, тенісні корти, поля для гольфу, паркінг тощо;

– функція «Інші службі» дозволяє перейти до інших служб.

Інших сервісів наведено дуже багато (Рис 2.1.2). Для цього навіть впроваджено пошук.

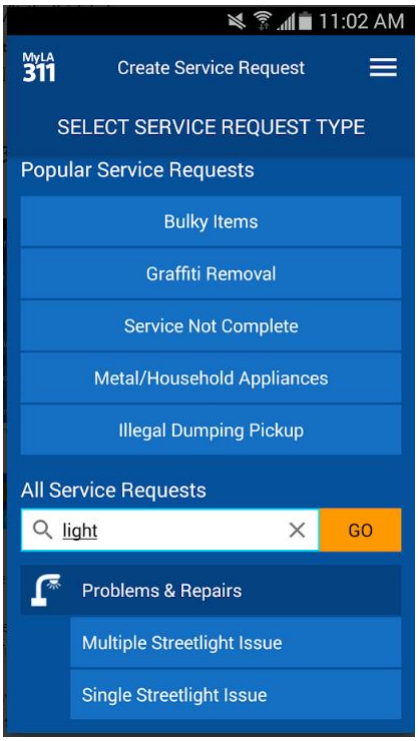

Рисунок 2.1.2 – Пошук за доступними сервісами в системі

Завдяки сервісам можливо:

– оплачувати рахунки в місті за воду та електроенергію;

– переглядати стрічку новин;

– переглядати поточні статуси своїх запитів проблем і таке інше.

Переваги:

– доступ до великої цифрової системи міста, що має багато сервісів;

– зрозумілий та простий варіант інтерфейсу;

– можливість залишити мітку анонімно.

Недоліки:

– застарілий інтерфейс, що не відповідає новітнім вимогам для Android та IOS користувачів;

– доступ до встановлення додатку можна отримати лише з території Сполучених Штатів Америки або потрібно використовувати VPN сервіси.

2.1 Система Pakistan Citizen Portal

Портал громадянина Пакистану – це інтегрована система розміщення скарги громадян, яка з’єднує всі урядові організації як на федеральному, так і на провінційному рівнях. (Рис. 2.2.1) Система служить носієм скарг у відповідні офіси по всьому Пакистану. Додаток слугує додатковим каналом між громадянином та владою. [2]

Рисунок 2.2.1 – Екран привітання Pakistan Citizen Portal

Не зважаючи на те, що реєстрація обов’язкова, система дозволяє встановити додаток з будь-якої точки світу. В ній існує лише один сервіс «Проблем», що дозволяє створити скаргу та дізнатися її результат на головному екрані (Рис. 2.2.2), або більш детально в пунктах меню «Мої проблеми».

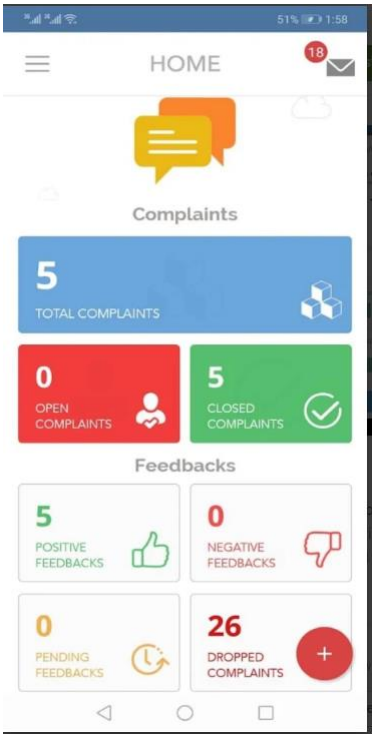

Рисунок 2.2.2. – Головний екран

Кожну скаргу можна віднести до різних категорій (Рис. 2.2.3).

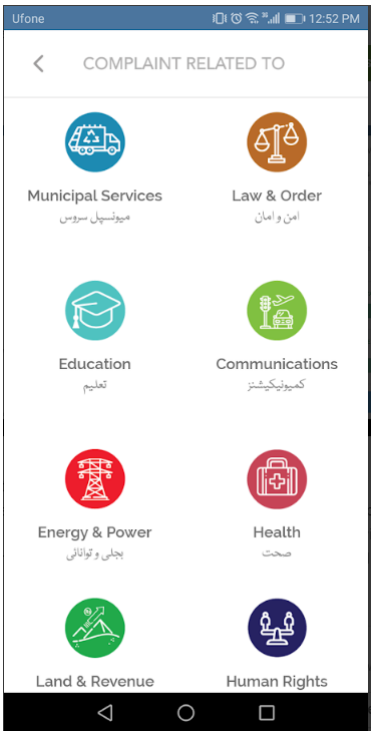

Рисунок 2.2.3 – Категорії скарг

В результаті обраної категорії, відповідні служби повинні вирішувати цю проблему та надавати зворотній зв’язок.

Переваги:

- доступність додатку по всьому світу;
- закріплення системи на рівні країни, що дозволяє вдало вирішувати проблеми та впроваджувати будь-які інші сервіси.

Недоліки:

- система покриває лише одну потребу – скарги;
- не повністю інтернаціоналізований мобільний додаток.

2.2 Система 2GIS

2GIS - це каталог з картою та GPS навігацією в Україні та за кордоном. Додаток працює без підключення до Інтернету - завантажте міську базу даних і використовуйте її в літаку, метро або в зоні роумінгу.[3] (Рис. 2.3.1)

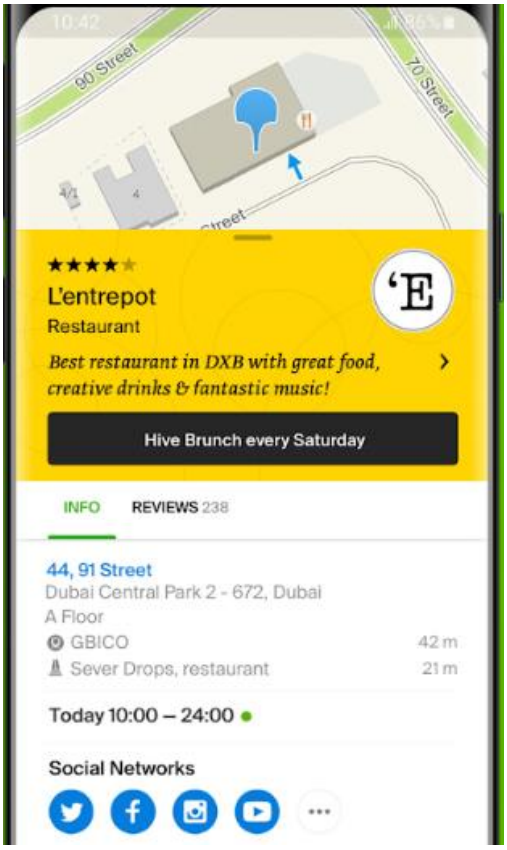

Рисунок 2.3.1 Приклад інтерфейсу

Система впроваджує велику кількість інформаційних сервісів: розклад роботи кафе, поштових відділень, лікарень або аптек; дозволяє знаходити та створювати маршрут аж до самих дверей вашого пункту призначення.

Переваги:

– працює на території України;

– неперевершена карта з великою кількістю інформації;

– додаток відповідає сучасному інтерфейсу Android та IOS системам.

Недоліком крім інформаційних сервісів та побудування маршруту є те, що немає нічого спільного із допомогою виправлення ситуації навколо.

Висновок до розділу

В цьому розділі було розглянуто три популярних рішення на базі GIS систем, та виділено їх недоліки на переваги. Їх порівняння відображено у таблиці 2.1.

Таблиця 2.1 – Порівняння альтернативних рішень

Назва / Критерії	My LA 311	Pakestan Citizen Portal	2GIS
Сучасний інтерфейс	-	-	+
Підтримка державними органами влади	+	+	-
Різноманітний функціонал використання	+	-	+
Використання на території України	-	-	+
Зручна робота з картою	-	-	+

Потрібна зауважити стосовно останнього пункту, дійсно зручна робота з картою зроблена лише в останньому рішенню на GIS технологіях – 2GIS, тому що тільки для цього рішення це є основним функціоналом системи, з чим стикається користувач кожну секунду проведену у додатку. На відміну від двох інших рішень, де карта є додатковим інструментом до основних робочих процесів систем.

3. ЗАСОБИ РОЗРОБКИ

3.1 Технології для реалізації мобільного додатку

Створення мобільного додатку відбувалось за допомогою Flutter Framework на мові Dart в середовищі Visual Studio Code.

3.1.1 Flutter Framework – крос-платформний фреймворк

Flutter - це портативний інструментарій для користувальницького інтерфейсу Google для створення красивих програм, створеня мобільних пристроїв, Інтернет застосунків та настільних ПК з однієї бази коду. Flutter працює з існуючим кодом, використовується розробниками та організаціями по всьому світу і є безкоштовним та відкритим кодом.

Для користувачів Flutter реалізує прекрасні інтерфейси додатків.

Для розробників Flutter знижує смугу вступу для створення програм. Це прискорює розробку додатків і зменшує витрати та складність виробництва додатків на різних платформах.

Для дизайнерів Flutter допомагає досягти оригінального дизайнерського бачення, не втрачаючи якості чи компромісів. Він також виступає як продуктивний інструмент прототипування.

Flutter призначений для розробників, які шукають швидкий спосіб створення красивих додатків або спосіб залучення більшої кількості користувачів за допомогою однієї інвестиції.

Flutter також призначений для інженерних менеджерів, які керують розробниками. Flutter дозволяє менеджерам створювати єдину команду розробників

додатків для мобільних пристроїв, Інтернету та настільних комп'ютерів, об'єднуючи свої інвестиції в розвиток, щоб швидше надсилати більше функцій, перевозити одну і ту ж функцію на декілька платформ одночасно і знижувати витрати на обслуговування.

Flutter також призначений для дизайнерів, які хочуть, щоб їх оригінальні дизайнерські бачення постачалися послідовно, з високою якістю, всім користувачам. Насправді CodePen тепер підтримує Flutter. [5]

По суті, Flutter призначений для користувачів, які хочуть красивих додатків із чудовим рухом та анімацією, а також користувацькі інтерфейси з персонажем.

Flutter доступний для програмістів, знайомих з об'єктно-орієнтованими поняттями (класи, методи, змінні тощо) та імперативними концепціями програмування (циклі, умовні умови тощо).

Для вивчення та використання Flutter не потрібен попередній досвід.

Ми бачили, як люди з дуже невеликим досвідом програмування навчаються та використовують Flutter для створення прототипів та розробки додатків.

3.1.2 Dart – мова програмування

Dart - оптимізована клієнтом мова програмування для додатків на кількох платформах. Вона розроблена Google і використовується для створення мобільних, настільних, серверних та веб-додатків. Dart - це об'єктно-орієнтована мова, заснована на класах із синтаксисом у стилі C. [6]

Використовують Dart для написання простих сценаріїв або повнофункціональних додатків. Незалежно від того, створюють мобільний додаток, веб-додаток, сценарій командного рядка або додаток на сервері, для цього є рішення Dart.

Гнучка технологія компілятора дозволяє запускати код Dart різними способами, залежно від цільової платформи та цілей:

– Dart Native: для програм, орієнтованих на пристрої (мобільні, настільні, серверні та інші), Dart Native включає в себе як Dart VM з компіляцією JIT (вчасно), так і компілятор AOT (заздалегідь) для створення машинного коду;

– Dart Web: для програм, орієнтованих на Інтернет, Dart Web включає як компілятор часу розробки (dartdevc), так і компілятор часу виробництва (dart2js).

Існує чотири способи запуску коду Dart:

– Компілювання як JavaScript. Для запуску в основних веб-браузерах Dart покладається на компілятор від джерела до джерела в JavaScript. Dart був "розроблений так, щоб легко писати інструменти для розробки, добре підходять до сучасних розробок додатків і здатні до високопродуктивних реалізацій". Під час запуску коду Dart у веб-браузері код є попередньо скомпільований у JavaScript за допомогою компілятора dart2js. Компільований як JavaScript, Dart-код сумісний з усіма основними браузерами, у яких браузери не потребують прийняття Dart. Завдяки оптимізації компільованого виводу JavaScript, щоб уникнути дорогих перевірок і операцій, код, записаний у Dart, може в деяких випадках працювати швидше, ніж еквівалентний код, написаний вручну, використовуючи ідіоми JavaScript;

– Автономний. Комплект розробок програмного забезпечення Dart (SDK) постачається з автономною програмою Dart VM, що дозволяє Dart-коду працювати в інтерфейсі командного рядка. Оскільки мовні засоби, включені до SDK Dart, написані здебільшого на Dart, автономна Dart VM є важливою частиною SDK. Ці інструменти включають компілятор dart2js та менеджер пакунків під назвою pub. Dart постачається з повною стандартною бібліотекою, що дозволяє користувачам писати повністю працюючі системні додатки, такі як користувацькі веб-сервери;

– Попередньо скомпільований. Dart-код можна компілювати за допомогою AOT у машинний код (нативний набір інструкцій). Програми, створені за допомогою Flutter, SDK для мобільних додатків, створених за допомогою Dart, розміщуються в магазинах додатків як компільовані AOT Dart-кодом;

– Рідні. Dart 2.6 з dart2native компілятором для компіляції в автономний, власний код виконавчих файлів. До Dart 2.6 ця функція відкривала цю можливість лише на мобільних пристроях iOS та Android через Flutter.

3.1.3 Visual Studio Code – редактор вихідного коду

Visual Studio Code (VS Code) - це спрощений редактор коду з підтримкою таких операцій розвитку, як налагодження, виконання завдань та контроль версій. Він спрямований на надання лише інструментів, необхідних розробнику для швидкого циклу збирання-налагодження коду, а також залишає більш складні робочі процеси для більш повних представлених IDE, таких як ID Visual Studio. Код VS працює на macOS, Linux та Windows. [7]

У опитуванні розробників Stack Overflow 2019 Visual Studio Code потрапив у найпопулярніший інструмент середовища для розробників, 50,7% із 87,317 респондентів заявили, що використовують його. [9]

В її основі лежить рамка Electron, яка використовується для розробки веб-додатків Node.js, які працюють на механізмі компонування Blink. Visual Studio Code використовує той самий компонент редактора (кодова назва "Монако"), який використовується в Azure DevOps (раніше називався Visual Studio Online і Visual Studio Team Services).

Замість системи проектів вона дозволяє користувачам відкривати один або кілька каталогів, які потім можуть бути збережені у робочих просторах для подальшого повторного використання. Завдяки цьому, програма може працювати як мовно-агностичний редактор коду для будь-якої мови, всупереч Microsoft Visual Studio, яка в свою чергу, використовує патентний файл рішення .sln та файли проекту, що стосуються конкретного проекту, що зовсім не зручно та не ефективно. VS Code підтримує ряд мов програмування та набір функцій, що відрізняються для кожної

мови. Небажані файли та папки можна виключити з дерева проектів за допомогою налаштувань. Багато функцій коду Visual Studio не відкриваються через меню або користувальницький інтерфейс, але до них можна отримати доступ трохи пошукав цю інформацію в Інтернеті.

Visual Studio Code стала настільки популярною завдяки своїм розширенням, доступних через центральне сховище. Сюди входять, як доповнення до редактора, так і мовна підтримка. Помітною особливість є можливість самому створювати розширення, що додають підтримку нових мов, тем та налагоджувачів, здійснюють аналіз статичного коду і таке інше.

Visual Studio Code дозволяє користувачам встановлювати кодову сторінку, на якій зберігається активний документ, символ нової лінії та мову програмування активного документа. Це дозволяє використовувати його на будь-якій платформі, у будь-якій місцевості та для будь-якої мови програмування.

3.2 Технології для реалізації серверної частини

Сервер створений за допомогою ASP.NET Core Framework з використанням бази даних MS SQL.

3.2.1 ASP.NET Core Framework – Web Framework від Microsoft

ASP.NET Core - це web framework з відкритим кодом, створений Microsoft, для створення сучасних веб-додатків та служб за допомогою .NET платформи.

ASP.NET є кросплатформеним і працює на Windows, Linux, macOS та Docker. Під час використання ASP.NET серверний код, такий як бізнес-логіка та доступ до даних, записується за допомогою C#, F# або Visual Basic.

Оскільки ASP.NET розширює .NET, можна використовувати велику екосистему пакетів і бібліотек, доступних усім розробникам .NET. Також можливо створити власні бібліотеки, якими є змога поділитися між будь-якими програмами, написаними на платформі .NET. [8]

3.2.2 MS SQL – база даних

Microsoft SQL Server - це система управління реляційними базами даних, розроблена Microsoft. Як сервер бази даних, це програмний продукт з основною функцією зберігання та отримання даних, як цього вимагають інші програмні програми, який може працювати або на тому ж комп'ютері, або на іншому комп'ютері в мережі. [10]

3.3.3 C# - мова програмування

C # (вимовляється «Сі Шарп») - сучасна, об'єктно-орієнтована і безпечна мова програмування. C# має своє коріння в сімействі мов C і буде одразу знайома

програмістам C, C ++, Java та JavaScript. C# - об'єктно-орієнтована мова, але C# додатково включає підтримку програмованого орієнтування на компоненти. Сучасний дизайн програмного забезпечення все більше покладається на програмні компоненти у вигляді автономних та самоописуючих пакетів функціональності. Ключовим для таких компонентів є те, що вони представляють модель програмування із властивостями, методами та подіями. Вони мають атрибути, які надають декларативну інформацію про компонент. Вони містять власну документацію. C# надає мовні конструкції для прямої підтримки цих концепцій, роблячи C# природною мовою, для створення та використання програмних компонентів.

Кілька функцій C# допомагають створювати надійні та довговічні програми. Збір сміття автоматично відновлює пам'ять, зайняту недоступними невикористаними об'єктами. Обробка винятків забезпечує структурований та розширюваний підхід до виявлення та відновлення помилок. Безпечно типова конструкція мови не дає змоги читати з неініціалізованих змінних, індексувати масиви за їх межами або виконувати некоректні приведення типів.[11]

3.3 Технічні вимоги до програмного забезпечення сервера

Операційна система: Windows 7 SP1 або новіше (64-bit)
Процесор: x86 або x64
Оперативна пам’ять (RAM) : 512 MB (мінімум), 1 GB (рекомендовано)
Дисковий простір: 1000 MB (не включаючи дисковий простір IDE/tools).

Висновок до розділу

Система «Awesome Map» була розроблена за допомогою новітніх технологій найуспішніших IT компаній світу, таких як Google та Microsoft.

Використовуючи Google Cloud Platform, система потенційно має доступ майже до всіх функцій Google, що дозволяє не тільки отримувати дані з карти, а також мати змогу авторизувати користувача за допомогою Google акаунта, через OAuth сервер.

Використання Flutter Framework з однієї сторони дійсно збільшує ризики впровадження та підтримку продукту для бізнеса в майбутньому, адже ця технологія нова і нема великого бізнес досвіду у розробників, але спираючись на досвід викладачів в НТУУ «КПІ ім. Ігоря Сікорського», ця технологія вже знайшла свою частину ринку, в мобільній розробці зокрема, та буде швидко збільшувати свою аудиторію розробників найближчим часом.

Як результат, система зможе мати підтримку та впровадження новітніх функцій від майбутніх розробників ще мінімум 8 років, що дозволяє зменшити ризики використання даного продукту в бізнес проєктах та опираючись на нього, будувати свої додаткові системи.

4.ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Взаємодія користувача з системою

Створення будь-якої системи починається з опису можливих варіантів її взаємодії з користувачем. Для цього була спроектована Use-Case діаграма, або діаграма прецедентів, що дозволяє відобразити використання системи в найпростішому вигляді, та показати взаємозв'язок між користувачем і різними варіантами використання системи, в яких вона втягує користувача.

За правилам діаграми прецедентів, проектована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою за допомогою так званих прецедентів.

Актор - це набір ролей, які виконує користувач в ході взаємодії з деякою сутністю (системою, підсистемою, класом). Актор може бути людиною, іншою системою, підсистемою або класом, які представляють щось за межами розглянутої сутності.

Прецедент - це опис набору послідовних подій (включаючи можливі варіанти), що виконуються системою, які призводять до результату спостережуваного Актором. Прецеденти описують сервіси, що надаються системою Актор, з якими вона взаємодіє. Причому прецедент ніколи не пояснює, «як» працює сервіс, а тільки описує, «що» робиться.

Між елементами діаграми прецедентів можуть існувати різні відносини, які описують взаємодію екземплярів акторів і варіантів використання.

Стандартні види відносин між акторами і прецедентами:

- асоціації (association relationship);
- розширення (extend relationship);
- узагальнення (generalization relationship);
- включення (include relationship).

Діаграма прецедентів для системи зображено на рис. 4.1.

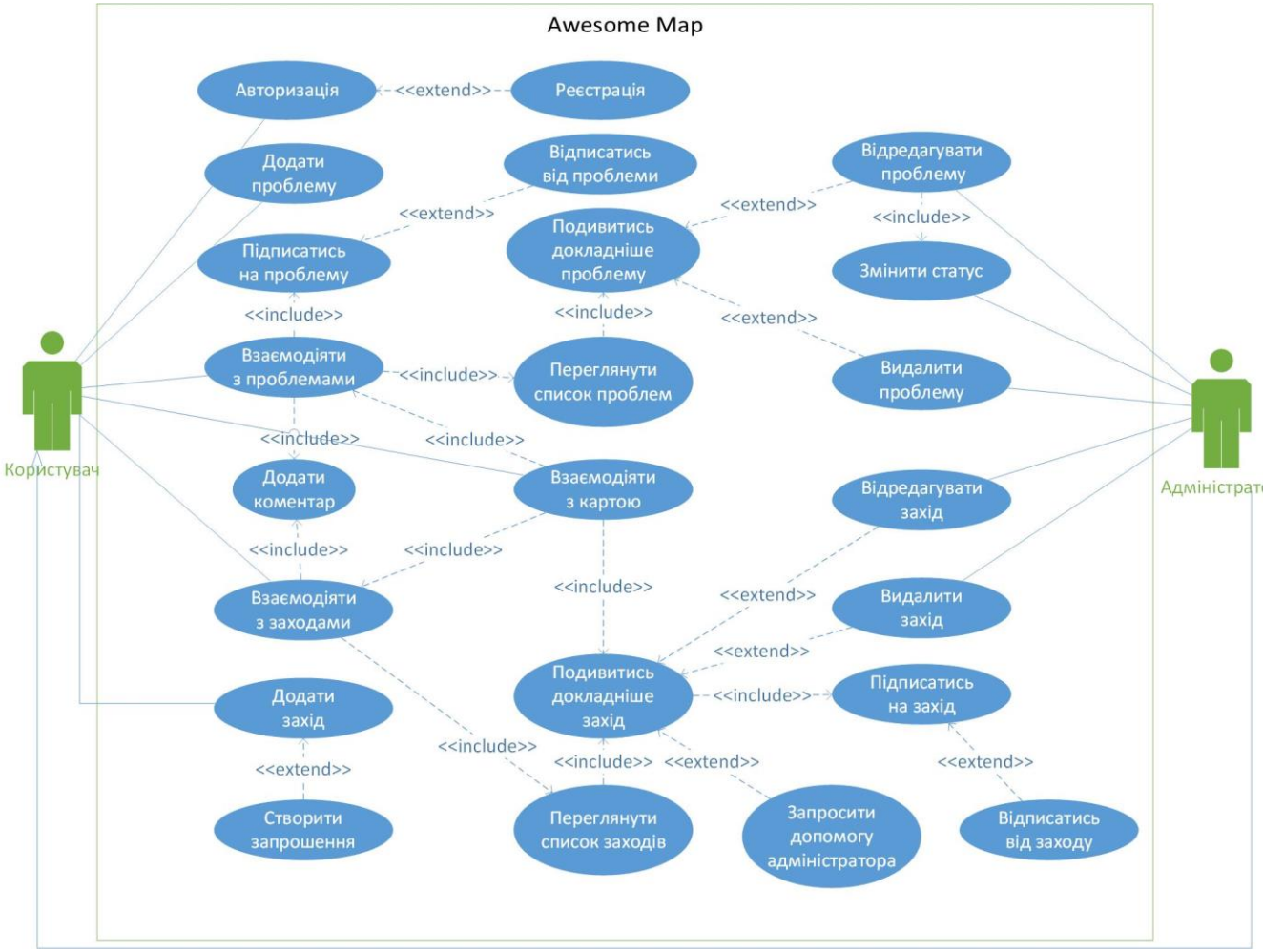

Рисунок 4.1 – Діаграма прецедентів

Аналізуючи діаграму, в системі є дві ролі: Користувач і Адміністратор.

Користувач має змогу зареєструватись, якщо немає ще облікового запису або ж авторизуватись. Реєстрація - це розширення авторизації, а не включення, тому що користувач, який має вже обліковий запис, не повинен мати змогу знову зареєструватись.

Основні дії користувача зводяться до взаємодії з існуючими проблемами та заходами або їх створення. Якщо більш детально, то користувач має змогу додати свою проблему або підписатись на чужу проблему, що дозволить слідкувати за нею та дізнатись результат. Також реалізована можливість відписатись від проблеми, тільки, якщо користувач вже на неї підписався.

Той самий функціонал створений для заходів. Взаємодія з цими двома сутностями реалізовано за допомогою карти або списку.

У кожному з цих варіантів користувач має змогу подивитись докладніше.

Користувач може відредагувати свою проблему, якщо її адміністрація не почала читати, що автоматично переводить проблему в статус «В опрацюванні» і анулює дозвіл на редагування.

На відміну від проблем, у користувача є змога відредагувати свій захід завжди.

Адміністрація має всі привілеї, як і користувач, але розширює свій функціонал можливостями відредагувати проблему або змінити її статус, а також видалити проблему чи захід з системи.

4.2 Діаграма послідовності

Для кращого розуміння, як саме повинні взаємодіяти користувач, мобільний додаток, сервер та база даних, було створено діаграму послідовності (Рис. 4.2).
Діаграма послідовностей показує взаємодії об'єктів, розташованих у часовій послідовності. Вона зображує об'єкти та класи, що беруть участь у сценарії, та послідовність повідомлень, що обмінюються між об'єктами, необхідні для виконання функціонального сценарію. Діаграми послідовності зазвичай пов'язані з реалізацією випадку використання в логічному представленні системи, що розробляється. Діаграми послідовності іноді називають діаграмами подій або сценаріями подій.

Діаграма послідовностей показує, як паралельні вертикальні лінії (рятувальні лінії) різні процеси або об'єкти, що живуть одночасно, і, як горизонтальні стрілки, повідомлення, що обмінюються між ними, у тому порядку, в якому вони відбуваються. Це дозволяє конкретизувати прості сценарії виконання графічно.

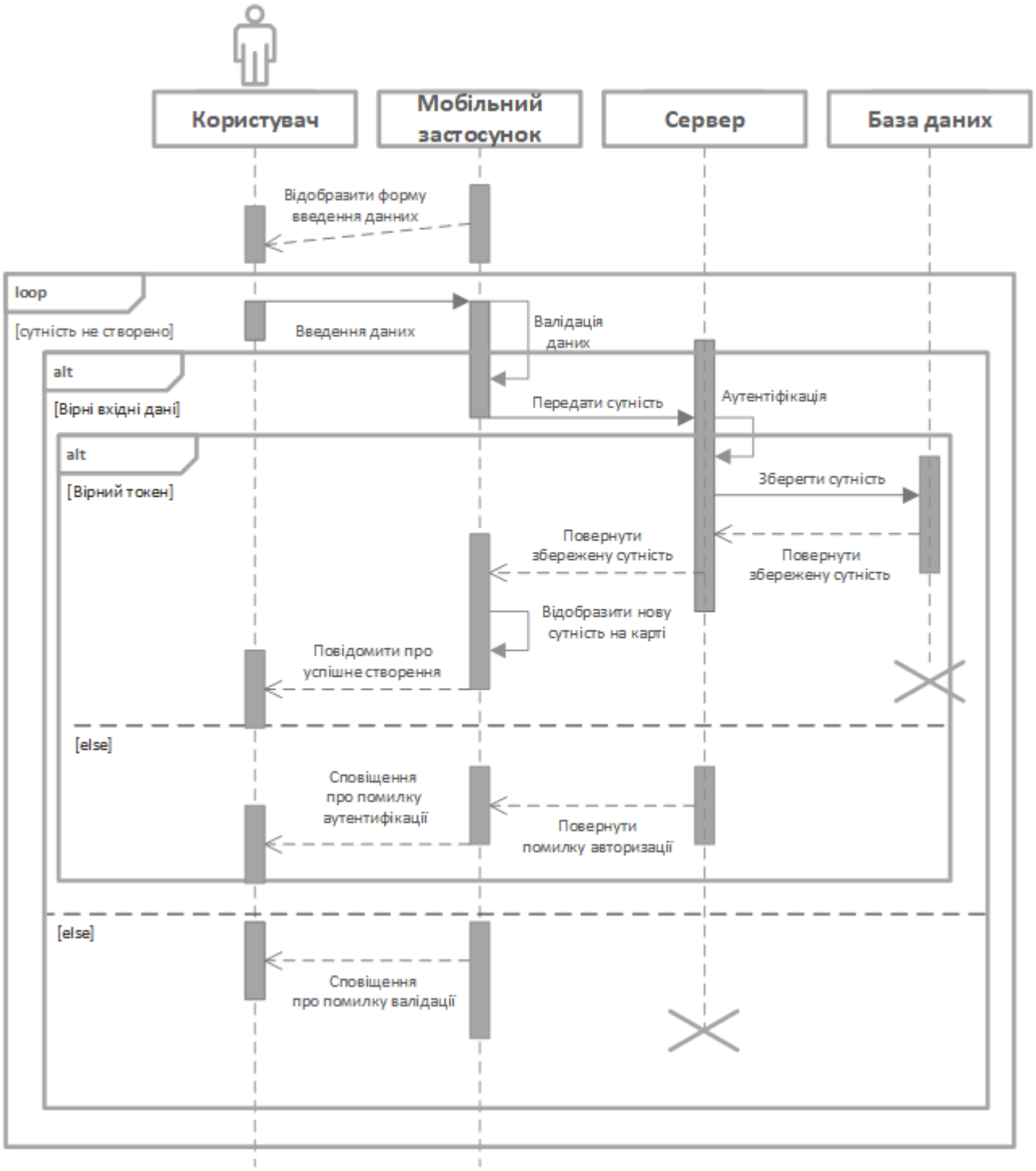

Рисунок 4.2 – Діаграма послідовності створення сутності в системі

На діаграмі відображено створення сутностей в системі, таких як «проблеми» та «заходи».

Користувачу з бажанням створити власну мітку мобільний додаток надає відповідну форму для проблеми або заходу. Користувач взаємодіє з мобільним

додатком надаючи вхідні дані для сутності. Після мобільний додаток перевіряє на правильність введених даних, де в гіршому випадку сповіщає користувача про помилку валідації його даних та потребує знову введення вхідних даних, що зображено на діаграмі елементом «loop» з умовою «[сутність не створено]» та перевіркою валідації в елементі «alt» з умовами «[Вірні вхідні дані]» та «[else]».

Після коректного валідування, мобільний додаток передає сутність серверу за допомогою одного з http запитів. В даній системі це має бути POST запит, що інформує сервер з REST архітектурою про потрібне створення сутності. Сервер на своїй стороні повинен спочатку автентифікувати користувача за переданим JWT-токеном у заголовку запита, та продовжити створення сутності або повернути помилку автентифікації мобільному додатку, що в свою чергу повідомить користувача про невдалу спробу.

Після вдалої автентифікації, сервер передає сутність базі даних для збереження, яка поверне оновлену сутність з заповненим ідентифікаційним полем (Id).

Після вдалого збереження проблеми або заходу, сервер повертає сутність, як відповідь на http запит мобільному додатку, що в свою чергу повідомляє користувача про успішне створення сутності.

Більш детально описано в додатках, де наведена реалізація алгоритму для сутності «Проблем» (Додаток 2) та її опис (Додаток 3).

4.3 Інтерфейс мобільного додатку

4.3.1 Екран входу

При першому запуску, користувачу відображається екран входу та реєстрації (Рис 4.3.1) з додатковою можливістю подивитись що може мобільний додаток.

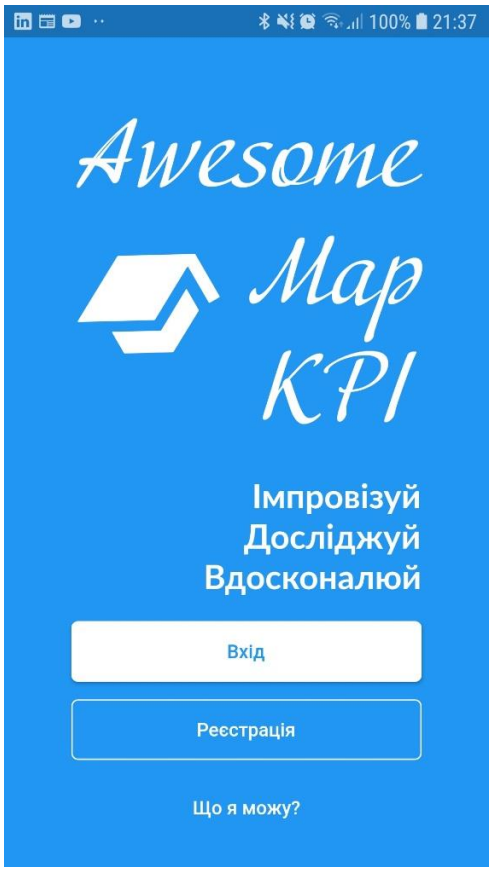

Рисунок 4.3.1 – Екран входу та реєстрації

В розділі «Що я можу» гість системи може продивитись зображення в GIF-форматі, де проілюстровано, що може робити система, наприклад:

– додати проблему;
– перегляд інших проблем;
– коментування проблем;
– додати захід;
– перегляд заходів;
– коментування заходів;
– фільтрування заходів або проблем.

Більш детально в репозиторії «Awesome Map» [12].

4.3.2 Головний екран

Після авторизації користувач потрапляє на головний екран додатку. (Рис. 4.3.2), де має доступ до головного функціоналу.

Навігаційна панель знизу, що виділена червоним на рисунку 4.3.2 відображає з якими сутностями зараз користувач працює.

Рисунок 4.3.2 – Головний екран додатку

Якщо виділений пункт «Проблеми», як на рисунку 4.3.2, то на карті відображаються лише проблеми, а компонент фільтрації працює лише з ними.

На вкладці «Карта» відображаються і проблеми, і заходи одночасно, що дає змогу користувачу зосередитись на карті та виокремити територію, яка йому цікава.

На вкладці «Заходи» відповідно можна побачити лише заходи.

4.3.3 Перемикач «Карта / Список»

На рисунку 4.3.3 компонент перемикач «Карта / Список» виділений червоним.

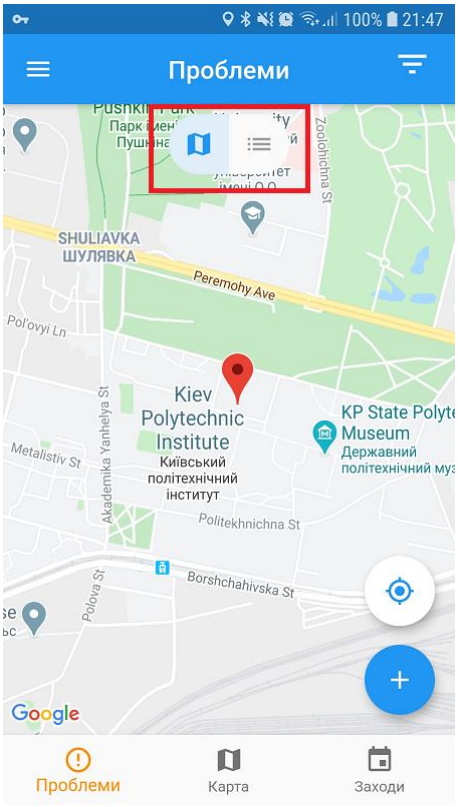

Рисунок 4.3.3 - Перемикач «Карта / Список»

Компонент дозволяє переключитись до списку елементів, що відображаються на карті, або повернутись знову зі списку до карти.

4.3.4 Компонент карта

Екран карти з проблемами (Рис 4.3.3) або з заходами (Рис 4.3.4.1) має ідентичні елементи управління користувальницького інтерфейсу.

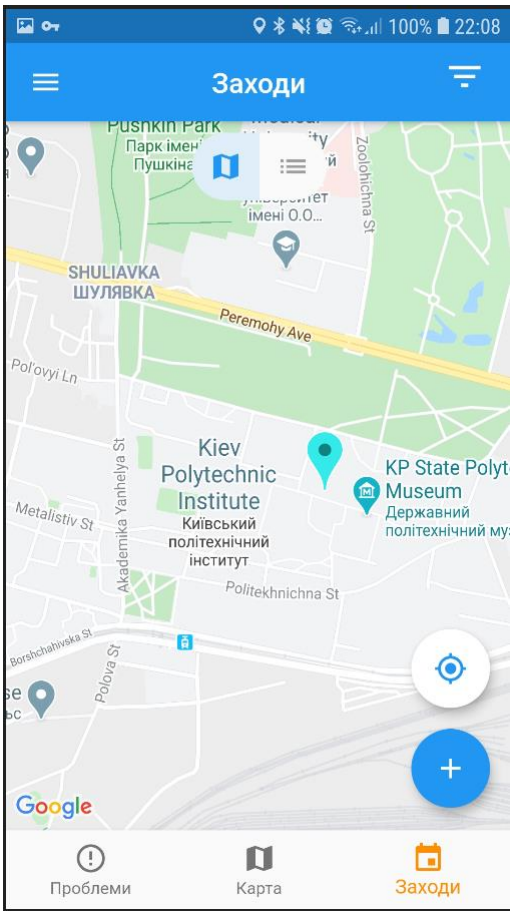

Рисунок 4.3.4.1 – Компонент карта для заходів

На цьому компоненті присутні елементи додавання («+»), пошук своєї локації та вже відомий перемикач «Карта / Список». Компонент дозволяє натиснути на мітку та отримати більш детальну інформацію про мітку в компоненті деталізації (Рис. 4.3.5.1 – Рис. 4.3.5.2)

4.3.5 Компонент деталізації

Компонент деталізації дозволяє отримати більше інформації про обрану мітку. На нього можна потрапити обравши мітку з карти або натиснувши «Детальніше» зі списку «Проблем» чи натиснувши на захід з відповідного списку «Заходів». (Рис. 4.3.5.1 – Рис. 4.3.5.2).

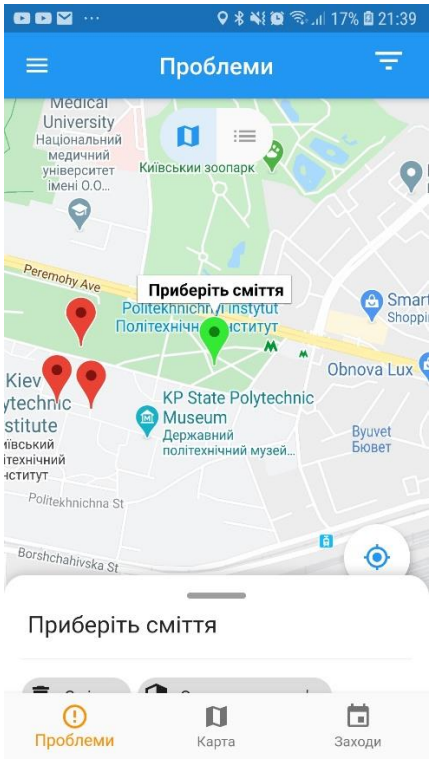

Рисунок 4.3.5.1 – Обрана мітка проблеми з детальним компонентом

Компонент можна розгорнути натиснувши на темну риску зверху компонента та потягнувши наверх або вниз – згорнути. Компонент для проблем та заходів дещо відрізняються але для користувача вони мають однакову логіку розгортання та згортання, та виконані в одному стилі, що створює інтерфейс додатку цілісним та повним.

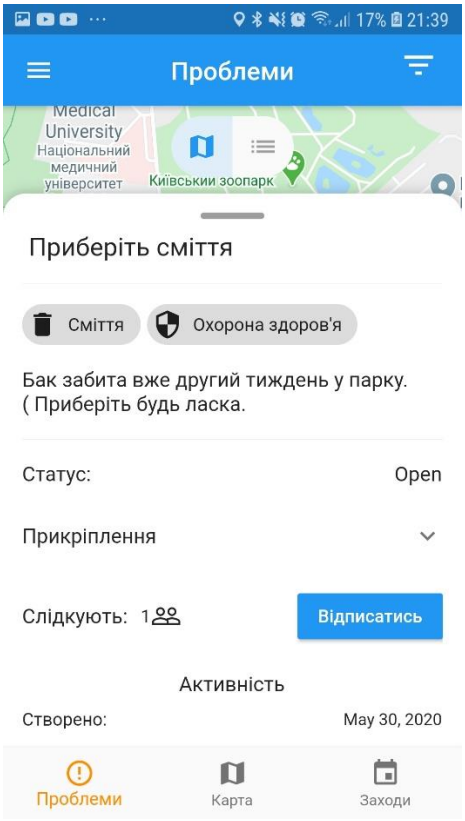

Рисунок 4.3.5.2 – Компонент деталізації в розгорнутому стані

Якщо цей користувач є власником мітки, то компонент деталізації буде дещо відрізнятися.(Рис. 4.3.5.3)

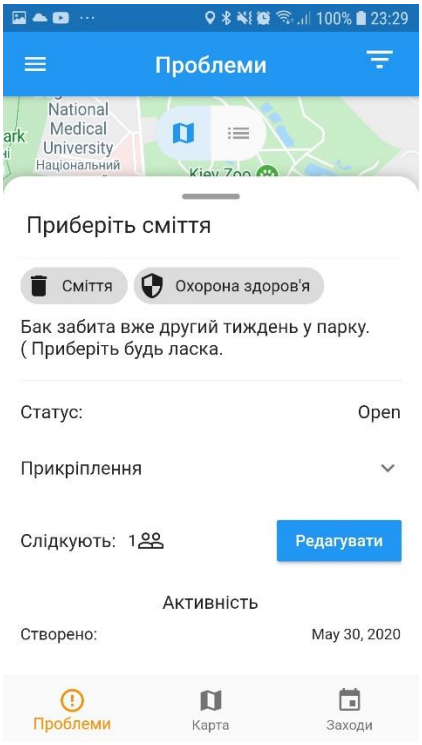

Рисунок 4.3.5.3 – Компонент деталізації для власника мітки

Власнику мітки не потрібна кнопка «Відписатись» чи «Підписатись», бо ця сутність завжди повинна бути розташована в розділі «Мої проблеми» або «Мої заходи» та сповіщення за цією сутністю повинні приходити завжди. Більш того, власнику потрібен функціонал редагування сутності. Щоб зберегти користувальницький інтерфейс цілісним, та видалити непотрібний функціонал, додавши новий, була створена замість звичних кнопок «Відписатись», «Підписатись» кнопка «Редагувати».

4.3.6 Компонент редагування сутності

Натискаючи на кнопку «Редагувати» користувач потрапляє на звичну йому форму. (Рис. 4.3.6)

На цій формі є можливість відредагувати свою сутність, змінити категорії, відредагувати назву або опис, видалити або додати новий файл. Наприклад якщо адміністрація запросить більш детальні докази або якіснішу фотографію, щоб користувач мав змогу це зробити.

За основу компонента було взято форму створення сутності, що дозволило зберегти однаковий інтерфейс на рівні користувача та зменшити кількість частин, що потрібно змінити при додаванні нового поля або редагуванні існуючих на рівні розробки додатка. Такі рішення дозволяють створювати більш вдалу та однозначну архітектуру мобільних додатків з мінімальними змінами в разі потреби.

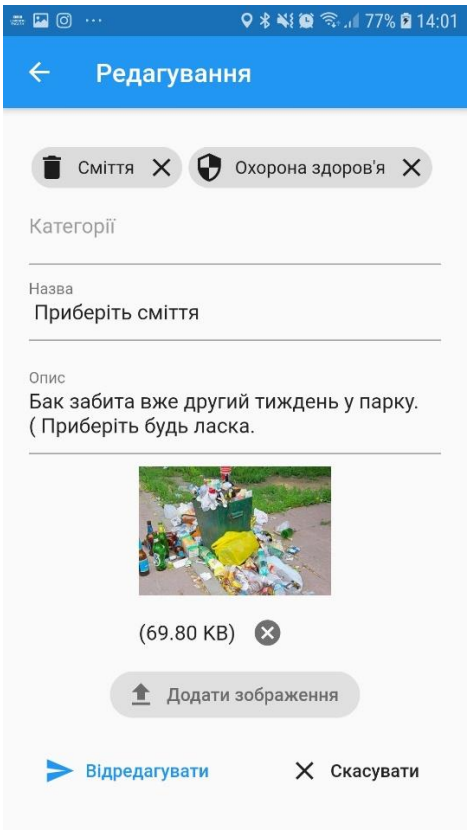

Рисунок 4.3.6 – Форма редагування

4.3.7 Компонент фільтрації

Компонент фільтрації був розроблений відповідно до технічного завдання.

Його спеціальне розташування у верхньому правому кутку зроблено для того, щоб при перемиканні з карти до списку, або навпаки, користувач очікував, що його фільтри досить застосовані. Завдяки цьому, додаток має змогу вдало переключатись між компонентами списку та карти і впроваджувати простий та зрозумілий інтерфейс при їх перегляді, створюючи однозначну функціональність.

Роздивимось фільтр для заходів детальніше на рисунку 4.3.7. Також було створено відображення аналогічно фільтру заходам фільтра для проблем, щоб зберегти цілісність інтерфейсу та знайомий варіант взаємодії додатку з користувачем.

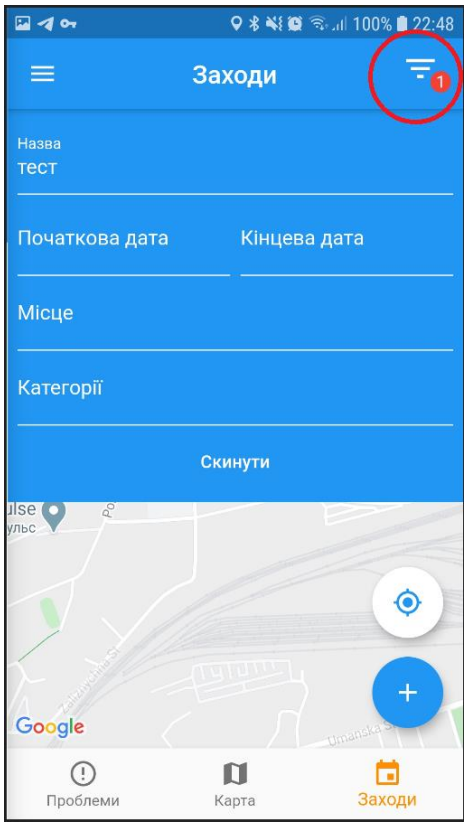

Рисунок 4.3.7 – Відкритий компонент фільтрації з одним заповненим елементом

Компонент фільтр має червоний символ «1», тому що заповнено одне поле для фільтрації, а саме - «Назва» із текстом «тест». Цей лічильник буде збільшуватись, доки користувач буде збільшувати поля за якими фільтрує сутність. В даному випадку, активне поле були лише одне, хоча доступні п’ять полів для фільтрації.

4.3.8 Компонент коментарів

Компонент коментарів інтегровано в кожен компонент деталізації проблеми або захода для того, щоб дізнатися більше інформації про цю сутність. (Рис 4.3.8)

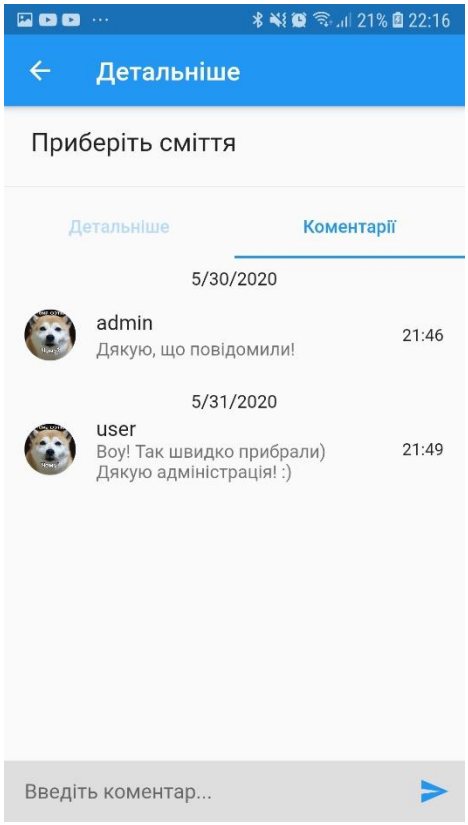

Рисунок 4.3.8 – Компонент коментарів

Насамперед, компонент можна побачити лише в деталізаціях сутностей, це зроблено щоб не розсіювати увагу користувача в інших компонентах, а керувати нею тільки, коли користувач дійсно зацікавився проблемою або заходом.

4.3.9 Компонент меню користувача

Меню користувача можна відкрити за допомогою кнопки, що розташована зверху зліва, біля заголовку або потягнувши за край лівого екрана вправо, доки меню не відкриється. (Рис. 4.3.9.1)

В меню зображено профіль користувача та кнопка з іконкою «Півмісяця» для темної теми (Рис. 4.3.9.2) або білого кола – світлої теми.

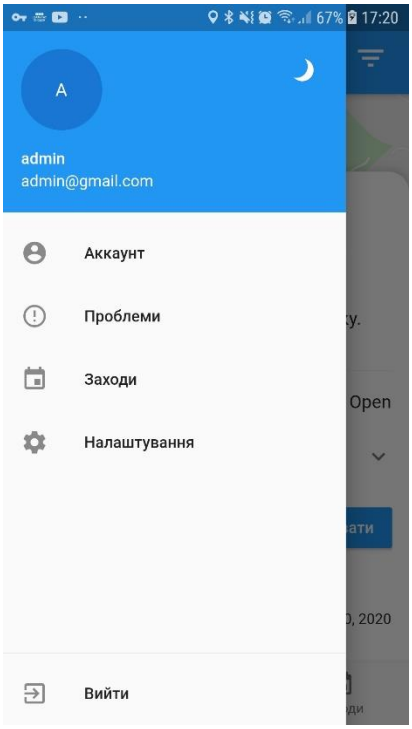

Рисунок 4.3.9.1 – Меню додатку

Усі компоненти мобільного додатку було розроблені з можливістю динамічної зміни теми, на будь-який колір в налаштуваннях.

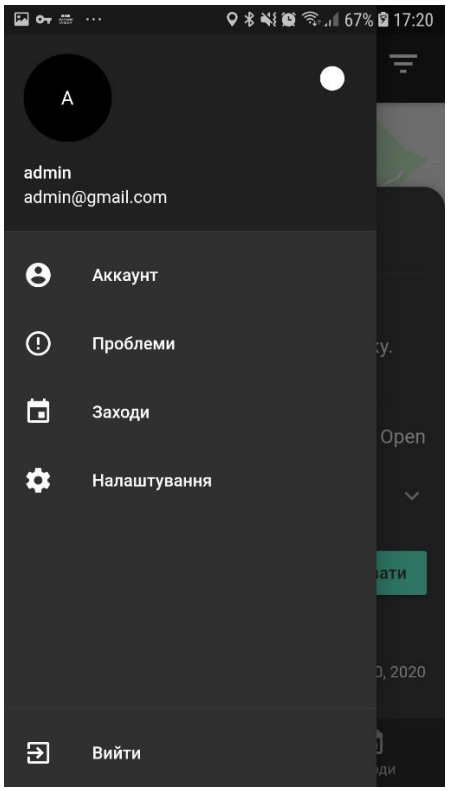

Рисунок 4.3.9.2 – Темна тема меню додатку

В меню також є пункти:

– аккаунт, де користувач має змогу змінити свої особисті дані;

– проблеми, де розташовується розділ «Мої проблеми», що дозволяє користувачу переглядати його створені проблеми та ті на які він вже підписався;

– заходи, де розташовується розділ «Мої заходи», що дозволяє користувачу переглядати його створені заходи та ті на які він погодився піти;

– налаштування, де розташовуються налаштування додатку;

4.4 Схема бази даних системи

Схема бази даних для серверної частини системи наведена на рисунках 4.4.1 та 4.4.2. Вона була створена для того, щоб показати які сутності існують в системі та як

вони взаємопов’язані. Більш того нижче описані деякі моменти, на котрі потрібно звернути увагу.

На діаграмі в додатковій колонці винесено тип полів та можливість поля приймати значення NULL.

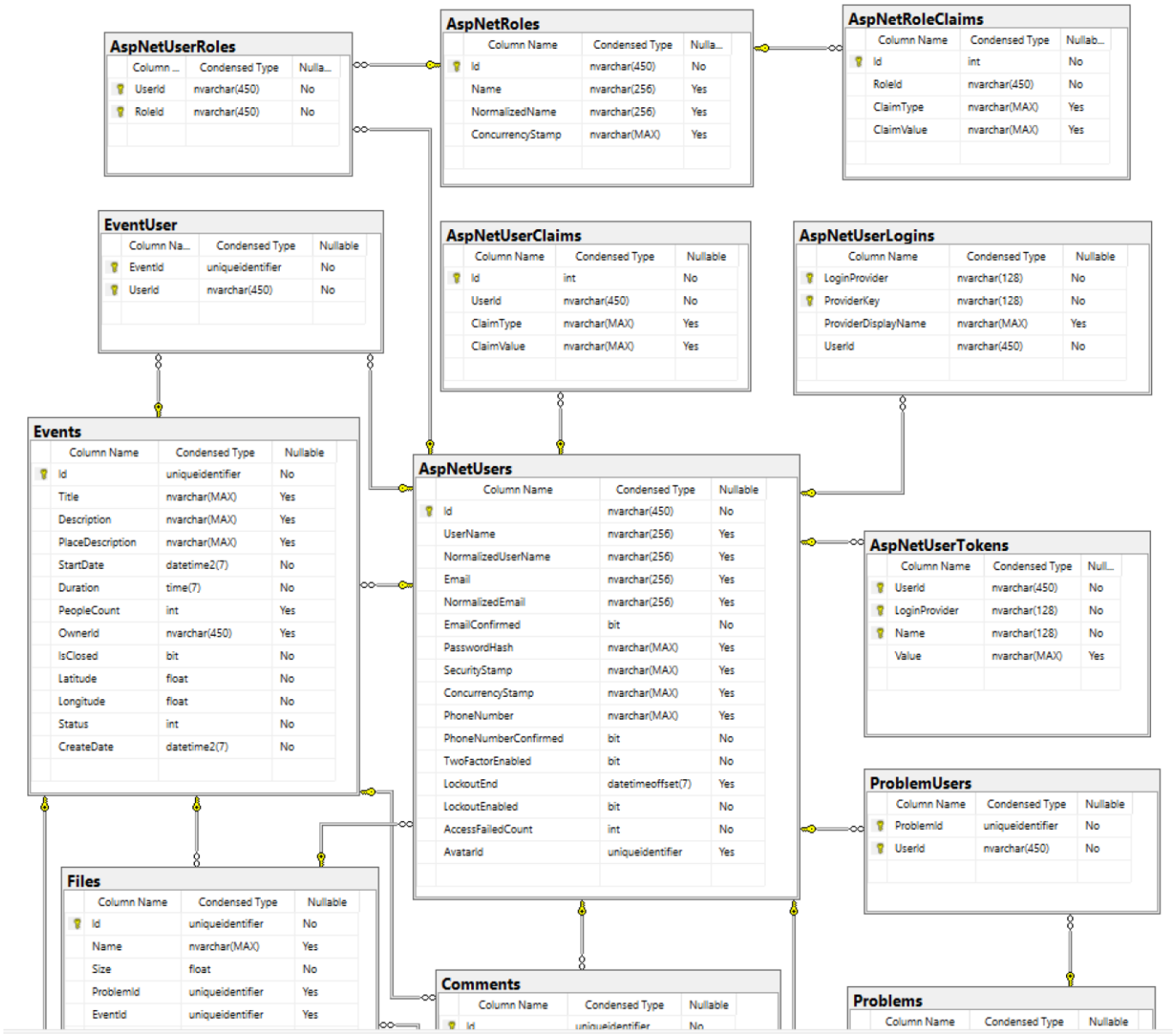

Рисунок 4.4.1 – Схема бази даних

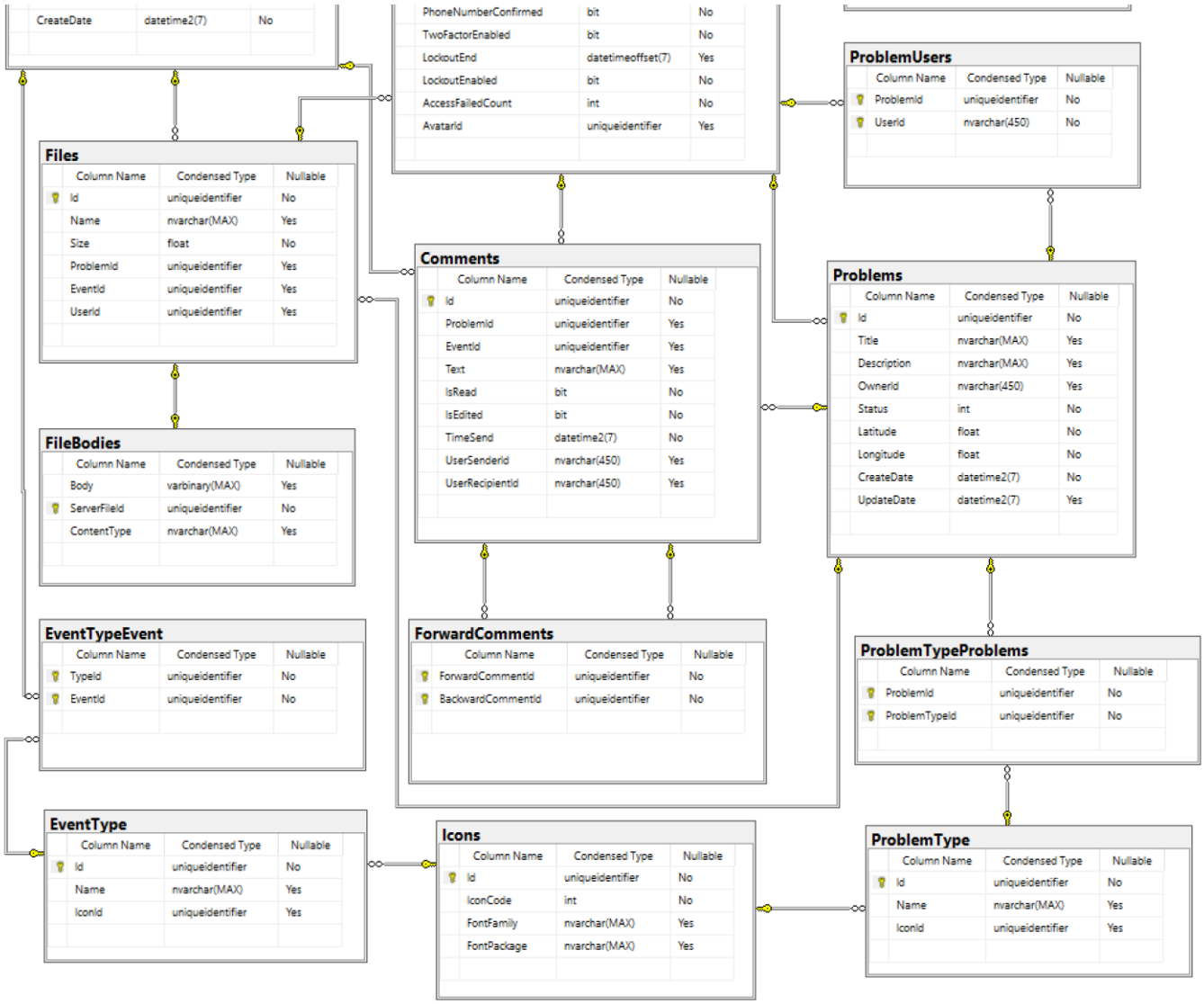

Рисунок 4.4.2 – Продовження схеми бази даних

На даній схемі показані зв’язки один до багатьох або один до одного через елементи: іконка ключа - один, нескінченність – багато.

В схемі бази даних, центральною сутністю можна виокремити AspNetUsers, яка являє собою головного користувача системи, та пов’язана з багатьма сутностями. Наприклад, у користувача є його створені проблеми та заходи, що пов’язує таблицю AspNetUsers з таблицями Problems та Events, як один до багатьох. Більш того, в системі передбачено підписування на чужі проблеми, або заходи, що потребує зв’язка багато до багатьох і в даному випадку реалізовано за допомогою додаткових таблиць ProblemUsers та EventUsers відповідно.

Треба звернути увагу на сутність Іконки (Icons), яка була створена спеціально для відображення типів проблем та заходів з відповідною іконкою, що потрібно лише для мобільного додатку. Вона приведена до базової сутності іконки у фреймворку Flutter, що дозволяє відображати будь-яку існуючу іконку в додатку, якщо цей пакет іконок встановлено заздалегідь.

Також виокремлено файл та тіло файла в дві окремі таблиці (Files та FileBodies відповідно), завдяки чому система набуває легкості при вилученні тільки текстової інформації та окремого завантаження файлу від мобільного додатку в єдиному незалежному запиті.

4.5 Реалізація сервера

Рухаючись від мобільного додатку, постає питання як саме повинен він взаємодіяти з сервером. Для цього була обрана REST-архітектура сервера, що описує простий та однозначний інтерфейс взаємодії двох систем.

4.5.1 REST-архітектура

REST (Representational state transfer) спочатку був задуманий як простий і однозначний інтерфейс для управління даними, який передбачав лише кілька базових операцій з безпосереднім мережевим сервером: вилучення даних (GET), збереження (POST), зміна (PUT / PATCH) і видалення (DELETE). Звісно ж цей перелік завжди супроводжувався такими опціями, як обробка помилок в запиті (чи коректно складений запит), розмежування доступу до даних і валідація вхідних даних, в

загальному, всіма можливими перевірками, які сервер виконує перед тим, як виконати бажання клієнта.

Крім цього REST має ряд архітектурних принципів, перелік яких можна знайти в будь-якій статті про REST. Якщо кількома словами, то:

– незалежність сервера від клієнта - сервери і клієнти можуть бути миттєво замінені іншими незалежно один від одного, так як інтерфейс між ними не змінюється. Сервер не зберігає станів клієнта;

– унікальність адрес ресурсів - кожна одиниця даних (будь-якого ступеня вкладеності) має свій власний унікальний URL, який, по суті, цілком є однозначним ідентифікатором ресурсу. Приклад: GET /api/v1/users/25/name;

– незалежність формату зберігання даних від формату їх передачі - сервер може підтримувати кілька різних форматів для передачі одних і тих же даних (JSON, XML і т.д.), але зберігає дані в своєму внутрішньому форматі, незалежно від підтримуваних;

– присутність у відповіді всіх необхідних метаданих - крім самих даних сервер повинен повертати деталі обробки запиту, наприклад, повідомлення про помилки, різні властивості ресурсу, необхідні для подальшої роботи з ним, наприклад, загальне число записів в колекції для правильного відображення посторінкової навігації.

Класичний REST має на увазі роботу клієнта з сервером як з плоским сховищем даних, при цьому нічого не говориться про пов'язаності і взаємозалежності даних між собою. Все це за замовчуванням цілком лягає на плечі клієнтського додатку. Однак сучасні предметні області, для яких розробляються системи управління даними, будь то соціальні сервіси або системи інтернет-маркетингу, мають на увазі складну взаємозв'язок між сутностями, що зберігаються в базі даних. Підтримка цих зв’язків, тобто цілісності даних, знаходиться в зоні відповідальності серверної сторони, в той час, як клієнт є тільки інтерфейсом для доступу до цих даних.

Щоб не міняти дані і зв'язку між ними вручну, ми просто викликаємо у ресурсі функцію і передаємо їй як аргумент необхідні дані. Ця операція не підходить під стандарти REST, для неї не існує особливого дієслова, що змушує розробників викручуватися хто на що здатний.

Найпростіший приклад - авторизація користувача. Викликаємо функцію login, передаємо їй як аргумент об'єкт, що містить облікові дані, і у відповідь отримуємо ключ доступу.

Ще варіант - створення і розрив зв'язків між даними. Наприклад, додавання користувача в групу. Викликаємо у сутності група функцію addUser, як параметр передаємо об'єкт користувач, отримуємо результат.

Ще бувають операції, які взагалі не пов'язані безпосередньо зі збереженням даних як таких, наприклад, розсилка повідомлень, підтвердження або відхилення будь-яких операцій (завершення звітного періоду і т.д.).

Часто буває так, що клієнтському додатку зручніше створювати / змінювати / видаляти / відразу кілька однорідних об'єктів одним запитом, і по кожному об'єкту можливий свій вердикт серверної сторони. Тут є як мінімум кілька варіантів: або всі зміни виконані, або вони виконані частково (для частини об'єктів), або сталася помилка. Ну і стратегій теж кілька: застосовувати зміни тільки в разі успіху для всіх, або застосовувати частково, або повертатися до попереднього стану в разі будь-якої помилки, а це вже тягне на повноцінний механізм транзакцій.

Приклад коду з REST архітектурою можно знайти в додатках (Додаток 2 ProblemController.cs).

4.5.2 Багаторівнева архітектура

В загальному вся система була розроблена за допомогою багаторівневої архітектури шарів.

Архітектурна структура «Шарів» або рівнів описана в різних публікаціях, але найпоширеніше вона представлена у вигляді чотирьох (Рис 4.5.2.1):

– рівень представлення;
– рівень застосунка або прикладний рівень;
– рівень бізнес логіки;
– рівень постійного сховища даних.

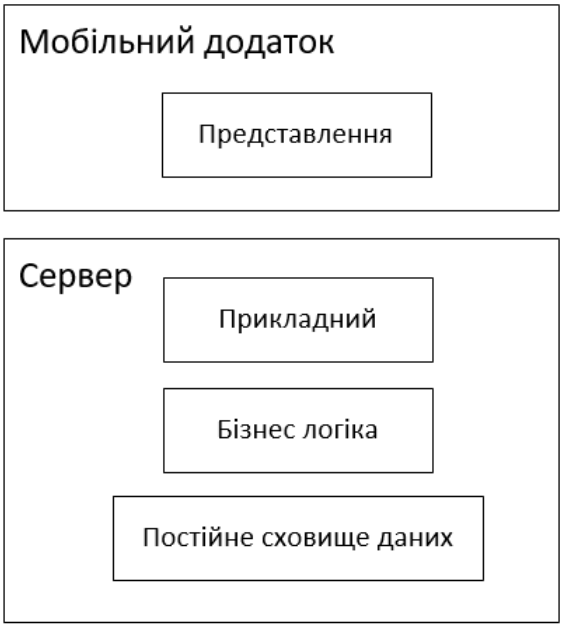

Рисунок 4.5.2.1 – Багаторівнева архітектура системи «Awesome Map»

Рівнем представлення виступає мобільний додаток, який відображає дані відповідно до всіх робочих процесів системи, та надає можливість користувачу взаємодіяти з системою.

Прикладним рівнем виступає частина сервера, що відповідає за валідацію та обробку http запитів. В більш звичному розумінні, цей рівень вважається підрівнем бізнес-рівня, як правило, інкапсулюючи визначення API, що створює додатково більш вдалу підтримку бізнес-функціональності в наступному рівні.

У системи «Awesome Map» цей рівень можна легко знайти, бо ним є весь застосунок на технології ASP.NET Core. В структурі проєктів серверної частини (Рис. 4.5.2.2) він зображений як проєкт з назвою «awesome_map_server».

В бізнес застосунках часто виходить, що цей рівень переплітається з рівнем бізнес логіки, тому часто прикладним рівнем виступають лише обробники http запитів, якими є контролери розташовані в теці Controllers, що можна також побачити на рисунку 4.5.2.2.

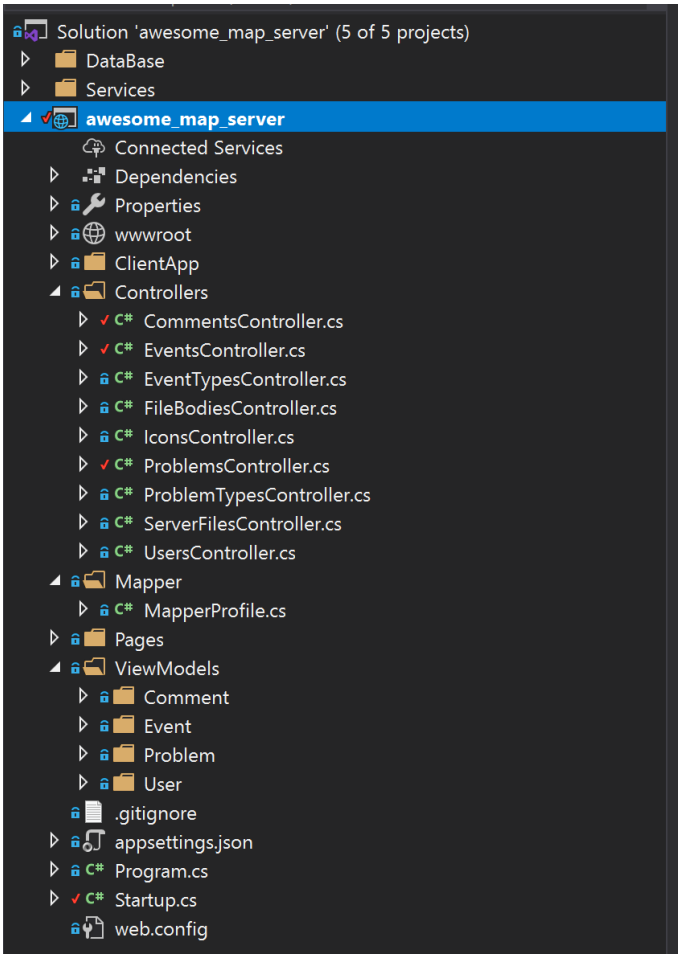

Рисунок 4.5.2.2 – Структура проєктів серверної частини

Рівень бізнес логіки реалізує робочі процеси системи. Він є основним в системі, адже він ідентифікує систему серед інших та впроваджує рішення для бізнесу замовника.

Гарним тоном архітектури є виокремлення рівня бізнес логіки від прикладного рівня, для того, щоб в сучасному світі з постійними оновленням фреймворків бізнес рішення не залежали від прикладного рівня, чим і виступають фреймворки.

Зменшив залежність цих рівнів, розробники мають можливість самі вирішувати чи потрібно безпечно оновлюватись до сучасної версії фреймворка чи ні, та бути впевненими, що в будь-якому випадку основна логіка бізнесу не постраждає.

У системі «Awesome Map» рівень бізнес логіки впроваджено у вигляді сервісів. (Рис. 4.5.2.3)

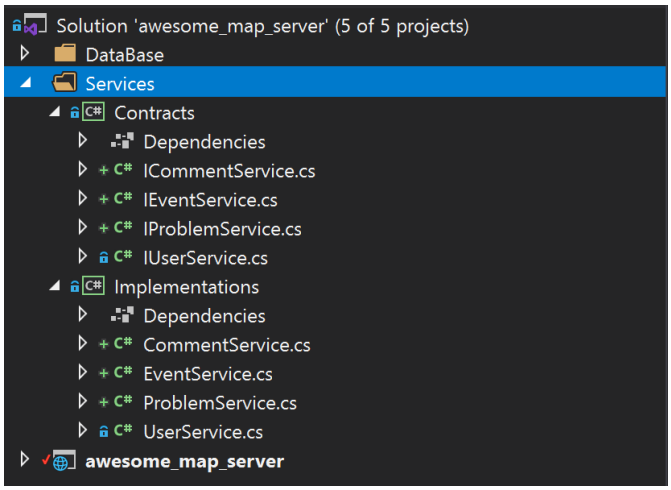

Рисунок 4.5.2.3 – Структура рівня бізнес логіки (сервісів)

За принципами SOLID, а саме принципом інверсії залежностей, що рекомендує створювати краще залежність на абстракціях, ніж на реалізаціях, рівень бізнес логіки (сервісів) був поділений на дві складові:

– Контракти, або інтерфейси («Contracts»);
– Реалізації («Implementations»).

Як висновок, вставка рівня бізнес логіки в систему встановлює низький рівень залежності від його реалізації, що дає змогу гнучко використовувати різноманітні реалізації для інших версій системи або проводити тестування деяких частин системи.

Оскільки багато інструментів тестування покладаються на успадкування, щоб здійснити мокінг, використання загальних інтерфейсів між класами (не лише між модулями, коли має сенс використовувати загальність) стало правилом.

Якщо використовуваний інструмент мокінга покладається лише на спадкування, може знадобитися широко застосовувати схему інверсії залежності. Це має основні недоліки:

– просто впровадження інтерфейсу над класом недостатньо для зменшення з'єднання; тільки думка про потенційну абстракцію взаємодій може призвести до менш пов'язаної конструкції;

– впровадження універсальних інтерфейсів скрізь у проекті ускладнює розуміння та підтримку. На кожному кроці читач запитає себе, які є інші реалізації цього інтерфейсу, і відповідь, як правило: лише моки;

– узагальнення інтерфейсу вимагає більшої кількості коду, зокрема фабрик, які, як правило, покладаються на впровадження залежностей;

– узагальнення інтерфейсу також обмежує використання мови програмування.

Поточний рівень бізнес логіки за допомогою налаштувань вбудованого механізму впровадження залежностей (Додаток 2 Startup.cs), використовується прикладним рівнем, а саме контролерами (обробниками http запитів). Треба зазначити, що взаємодія відбувається лише через абстрактну частину сервісів – інтерфейси.

Останнім рівнем архітектури є рівень зберігання даних або постійне сховище даних, що забезпечує доступ до даних, які зберігаються зазвичай в базі даних.

Цей рівень тісно пов’язан з рівнем бізнес логіки, тому рівень логіки знає, з якою саме базою даних він працює.

В застосунках ASP.NET цей рівень зазвичай представляють у вигляді Entity Framework.

Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну технологію, яка має змогу розшируватися, від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати з базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її

таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні оперують таблицями, індексами, первинними і зовнішніми ключами, то на концептуальному рівні, який пропонує Entity Framework, вже працють з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, можна через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

За замовчуванням на даний момент Microsoft надає ряд вбудованих провайдерів: для роботи з MS SQL Server, для SQLite, для PostgreSQL. Також є провайдери від сторонніх постачальників, наприклад, для MySQL.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо, наприклад, вирішуть змінити цільову СУБД, то основні зміни в проекті будуть стосуватися насамперед конфігурації і настройки підключення до відповідних провайдерам. А код, який безпосередньо працює з даними, отримує дані, додає їх в БД і т.д., залишиться колишнім.

Як технологія доступу до даних Entity Framework Core може використовуватися на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні додатки, WPF, UWP і ASP.NET Core. При цьому кроссплатформенная природа EF Core дозволяє задіяти її не тільки на ОС Windows, але і на Linux і Mac OS X.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людини, то можна виділити такі властивості, як ім'я, прізвище, зріст, вік. Властивості необов'язково представляють прості дані типу число або рядок, але можуть також представляти і більш комплексні типи даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому сутності можуть бути пов'язані асоціативної зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework Core, як технології ORM, є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо створювати різні запити на вибірку об'єктів, в тому числі пов'язаних різними асоціативними зв'язками. А Entity Framework при виконання запиту транслює вираження LINQ в вирази, зрозумілі для конкретної СУБД (як правило, в вирази SQL). [13]

Тому в системі «Awesome Map» рівень даних можна виокремити з набора сутностей, що представлені в проекті «DataBase» (Рис. 4.5.2.4).

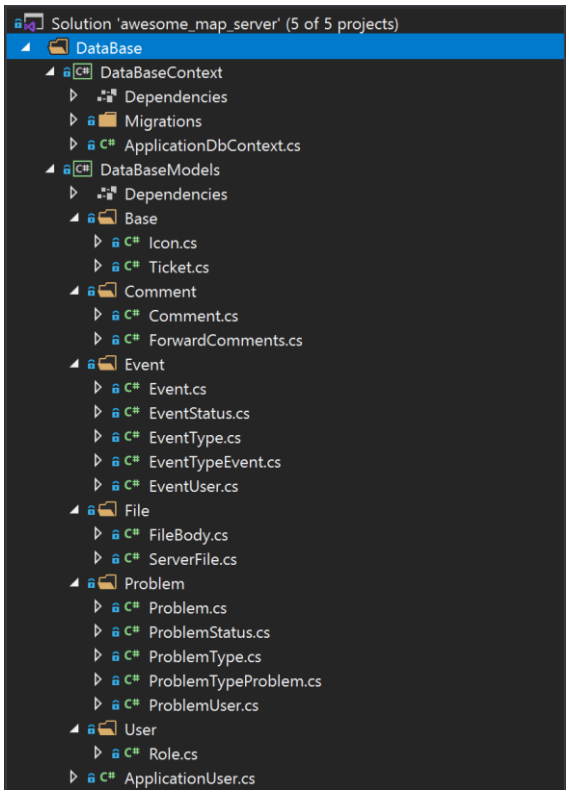

Рисунок 4.5.2.4 – Структура рівня даних

Вся архітектура системи працює завдяки виділенню сутностей в окремий проект, адже тепер цей рівень дозволено використовувати в рівні бізнес логіки, або в прикладному рівні безпосередньо.

4.5.3 Конфігурація сервера та конвеєр обробки http запитів

В цьому пункті розглядається налаштування сервера, конфігурація механізму впровадження залежностей та послідовний конвеєр обробки http запитів.

Вихідним код до цього пункту описаний в додатку 2 Startup.cs.

По перше, механізм впровадження залежностей частково вже працює надаючи в конструктор класу секретні конфігурації проекту, що створюються за допомогою інструменту Secret Manager. [14]

По друге, фреймворк ASP.NET Core автоматично виконує метод Configure та ConfigureService, де реалізована основна логіка налаштування.

В методі Configure проводяться налаштування Middleware (посереднє програмне забезпечення), що виступає конвеєром обробки http запитів (The request handling pipeline), що складається у вигляді серії проміжних програм.(Рис.4.5.3)

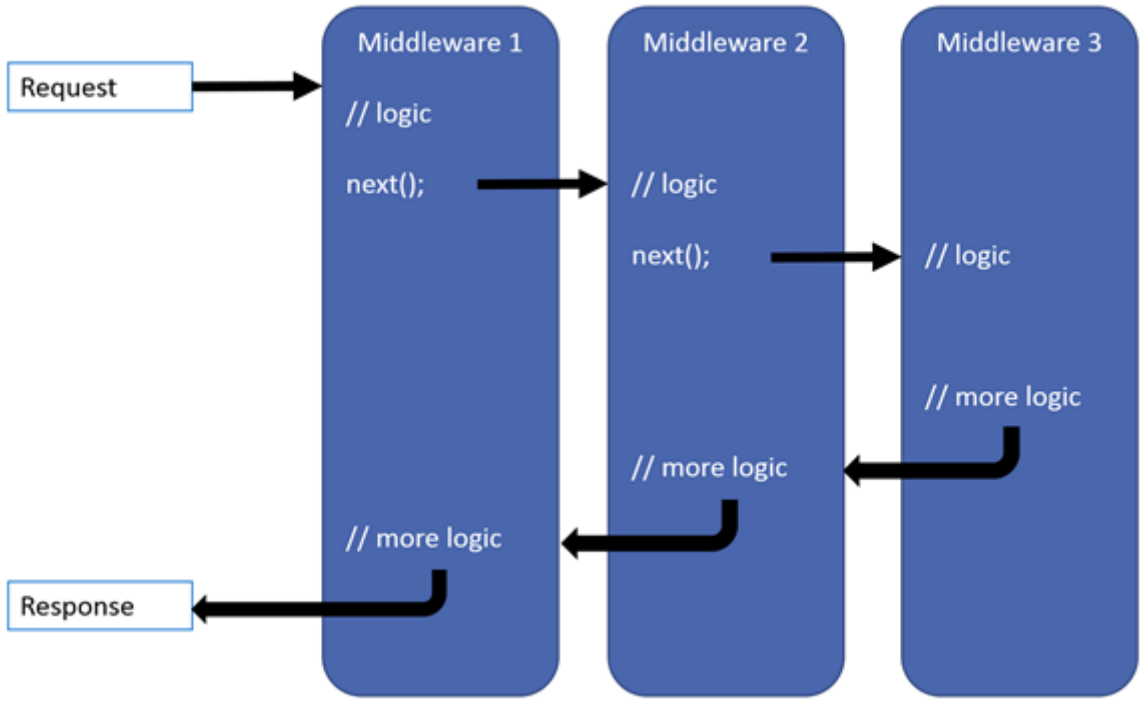

Рисунок 4.5.3 – Послідовність запитів, викликаних один за одним.

Потік виконання слідує за чорними стрілками.

Кожен компонент виконує операції над HttpContext і/або викликає наступне проміжне програмне забезпечення в конвеєрі, або припиняє запит. За умовою, компонент середнього програмного забезпечення додається до конвеєра шляхом виклику методу розширення Use ... у методі Startup.Configure. Наприклад, щоб увімкнути візуалізацію статичних файлів, викликається на об’єкті типу IApplicationBuilder метод UseStaticFiles.

Отже, в системі перш за все перевіряється в якому середовищі піднімається застосунок: якщо це в режимі розробника, то дозволяється відображати сторінку виключень та помилок, пов’язаних з базою даних; у противному випадку середовище є продуктове, де всі виключення перенаправляє на спеціально сторінку помилок, типу «Щось непередбачуване сталось, зверніться до адміністрації».

Далі за можливістю підключається переправлення всіх не https запитів.

Після цього, викликається вже знайома візуалізація статичних файлів.

Вагому частину в серверній частині відіграє наступний виклик включення маршрутизації, що несе відповідальність за відповідність вхідних запитів HTTP та передачу цих запитів до виконуваних кінцевих точок програми.

Наступним викликається UseAuthentication, що включає спроби автентифікувати користувача до того, як йому буде дозволений доступ до захищених ресурсів.

Далі включається стандартна реалізація сервера ідентифікації.

Нарешті, включається авторизація, що дає право користувачу на доступ до захищених ресурсів.

Наступним кроком проміжне програмне забезпечення для маршрутизації кінцевих точок (UseEndpoints з MapRazorPages) для додавання кінцевих точок сторінки Razor Pages до конвеєра запитів.

Та останнім виконується створення ролей та супер користувача з логіном та паролем з секретних налаштувань застосунку, якщо таке ще не виконувалось на поточній базі даних.

Інший метод ConfigureServices служить для налаштування служб програми. Сервіс - це багаторазовий компонент, який забезпечує функціональність додатків. Послуги реєструються в ConfigureServices і споживаються скрізь програму через ін'єкцію залежності (DI) або ApplicationServices.

Перш за все, в цьому методі додається контекст бази даних, якому в якості аргументів передається строка підключення до бази даних.

Далі налаштовується ідентифікація та ролі, конфігурації сервера ідентифікації та API авторизації.

Після створюється конфігурація JWT токенів та інші стандартні параметри.

В кінці метода було додано до механізма впровадження залежностей екземпляри сервісів рівня бізнес логіки: UserService, ProblemService, EventService та CommentService.

ІНСТАЛЮВАННЯ СИСТЕМИ

Для встановлення та використання системи звичному користувачу потрібно встановити мобільний додаток на телефон з Play Market, що носить назву додатку «Awesome Map». Відкрити додаток та зареєструватись.

Якщо ви розробник і намагаєтесь встановити систему собі локально, то вам потрібно:

1) Завантажити вихідний код. [12]
2) Налаштувати сервер, указавши рядок підключення в application.config файлі, що розташований у корні проєкту сервера.
3) Запустити сервер.
4) Налаштувати мобільний додаток, вказавши нову адресу сервера у файлі за шляхом <мобільний додаток>/lib/env/dev.json.
5) Запустити мобільний додаток.

Не потрібно створювати схему бази даних, тому що сервер автоматично створює її при першому запуску.

ВИСНОВКИ

В ході виконання дипломної роботи:

1. Розглянуті загальні задачі системи. Впроваджені головні та додаткові вимоги, які повинен вирішувати мобільний додаток в рамках робочих процесів системи, а також розглянуті вимоги до архітектури серверної частини та загальна роль сервера в системі «Awesome Map».

2. Проаналізовано три популярних рішення на базі GIS систем, такі як: «MyLa311», «Pakistan Citizen Portal» та «2GIS». Більш того, виділені їх недоліки та переваги. Результатом було створено порівняння, відображене у розділі 2 таблиці 2.1 та вдосконалена ця система.

3. Обрані та описані найновітніші технології для реалізації системи, що дозволяє мати підтримку та впровадження новітніх функцій від майбутніх розробників ще багато років, дозволяючи зменшити ризики використання даного продукту в бізнес проєктах та будувати свої додаткові системи.

4. Описані усі можливі взаємодії користувача з системою на діаграмі прецедентів, що дозволило вдало виділити головний функціонал системи, для простішого старту реалізації всієї системи.

5. Реалізована система та описано майже весь користувальницький інтерфейс мобільного додатку, що був розбитий на головні функціональні компоненти. Виділено багато цікавих моментів дизайну та обґрунтовано їх доцільне використання. Також описана та реалізована багаторівнева архітектура для всієї системи, дозволяючи збудувати гнучкий продукт, де легко видалити та замінити одну з частин системи на іншу.

6. Створена діаграма послідовності, на якій продемонстрована взаємодія частин систем та потоку даних між ними в часі.

7. Створена та описана діаграма схема бази даних системи, де обґрунтовані основні моменти побудови схеми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільний додаток «MyLa311» від City of Los Angeles [Електронний ресурс] - Режим доступу: https://play.google.com/store/apps/details?id=com.LA.MyLA311
2. Мобільний додаток «Pakistan Citizen Portal» від National IT Board, Government Of Pakistan [Електронний ресурс] - Режим доступу: https://play.google.com/store/apps/details?id=com.govpk.citizensportal
3. Мобільний додаток «2GIS» від DoubleGIS, LLC [Електронний ресурс] - Режим доступу: https://play.google.com/store/apps/details?id=ru.dublgis.dgismobile
4. Грамотная клиент-серверная архитектура: как правильно проектировать и разрабатывать web API, Владимир, web-developer in Noveo [Електронний ресурс] – Режим доступу: https://tproger.ru/articles/web-api
5. Flutter FAQ [Електронний ресурс] – Режим доступу: https://flutter.dev/docs/resources/faq
6. Dart (programming language) [Електронний ресурс] – Режим доступу: https://dart.dev/
7. Visual Studio Code [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/docs/supporting/faq
8. Developer Servey Results 2019 [Електронний ресурс] – Режим доступу: https://insights.stackoverflow.com/survey/2019#development-environments-and-tools
9. ASP.NET Core [Електронний ресурс] – Режим доступу: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet
10. MS SQL [Електронний ресурс] – Режим доступу: https://www.microsoft.com/en-us/sql-server

11. A tour of the C# language [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
12. Гавриляк О. В. Awesome Map [Електронний ресурс] / Олександр Володимирович Гавриляк. – 2020. – Режим доступу до ресурсу: https://github.com/defa808/awesome_map.
13. Введеня в Entity Framework Core [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/entityframeworkcore/1.1.php
14. By Rick Anderson, Kirk Larkin, Daniel Roth, and Scott Addie. Safe storage of app secrets in development in ASP.NET Core [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-3.1&tabs=windows.

ДОДАТОК 1

Засоби вирішення господарських проблем кампуса КПІ на базі ГІС систем

Специфікація

УКР.НТУУ «КПІ». ТІ-6281_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ТІ6281_20Б	ТІ-62 Гавриляк Пояснювальна записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ТІ6281_20Б	Startup.cs	Клас конфігурацій та конвеєра обробки запитів сервера
УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ТІ6281_20Б	ProblemsController.cs	Обробники http запитів для сутності «Проблем»
УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ТІ6281_20Б	ProblemService.cs	Взаємодія з базою даних з реалізованою бізнес логікою системи для сутності «Проблем»
УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ТІ6281_20Б	createProblemItemContent.dart	Відображення форми створення з валідацією для сутності «Проблем»

ДОДАТОК 2

Засоби вирішення господарських проблем кампуса КПІ на базі ГІС систем

Текст програми

УКР.НТУУ «КПІ». ТІ-6281_20Б

Аркушів 10

Київ 2020

Startup.cs

```
public class Startup {
      public Startup(IConfiguration configuration) {
         Configuration = configuration;
      }
      public IConfiguration Configuration { get; }
      public void ConfigureServices(IServiceCollection services) {

         services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(
               Configuration.GetConnectionString("DefaultConnection")));

         services.AddDefaultIdentity<ApplicationUser>(options => {
            options.SignIn.RequireConfirmedAccount = false;
            options.SignIn.RequireConfirmedPhoneNumber = false;
            options.Password.RequireDigit = false;
            options.Password.RequireNonAlphanumeric = false;
            options.Password.RequireDigit = false;
            options.Password.RequireUppercase = false;
            options.Password.RequiredUniqueChars = 0;
            options.Password.RequireLowercase = false;
         }
         ).AddRoles<IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>();

         services.AddIdentityServer()
            .AddApiAuthorization<ApplicationUser,
ApplicationDbContext>().AddJwtBearerClientAuthentication();

         services.AddAuthentication(o => {
            o.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
            o.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
         }).AddJwtBearer(options => {
            options.TokenValidationParameters = new TokenValidationParameters() {

               ValidateIssuerSigningKey = true,
               ValidateIssuer = false,
               ValidateAudience = false,
               ValidateLifetime = true,
               IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["JWTTokenSecret"]))
            };
         }).AddIdentityServerJwt().AddGoogle(options => {
            options.SignInScheme = JwtBearerDefaults.AuthenticationScheme;
            options.SaveTokens = true;
            options.CallbackPath = new PathString("/signin-google");
            IConfigurationSection googleAuthNSection =
         Configuration.GetSection("Authentication:Google");
            options.ClientId = googleAuthNSection["ClientId"];
            options.ClientSecret = googleAuthNSection["ClientSecret"];
         });
```

```
            services.AddControllersWithViews();
            services.AddRazorPages();
            // In production, the Angular files will be served from this directory
            services.AddSpaStaticFiles(configuration => {
                configuration.RootPath = "ClientApp/dist";
            });
            services.AddAutoMapper(typeof(MapperProfile));


            // настройка вбудованого механізму впровадження залежностей
            services.AddScoped<IUserService, UserService>();
            services.AddScoped<IProblemService, ProblemService>();
            services.AddScoped<IEventService, EventService>();
            services.AddScoped<ICommentService, CommentService>();
        }

        private async Task CreateRolesAndUser(IApplicationBuilder app, IServiceProvider serviceProvider)
{
            //ініціалізація ролей користувачів системи
            using (var scope = app.ApplicationServices.CreateScope()) {
                var RoleManager =
scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
                var UserManager =
scope.ServiceProvider.GetRequiredService<UserManager<ApplicationUser>>();
                string[] roleNames = { "Admin", "User" };
                IdentityResult roleResult;

                foreach (var roleName in roleNames) {
                    var roleExist = await RoleManager.RoleExistsAsync(roleName);
                    if (!roleExist) {
                        //create the roles and seed them to the database: Question 1
                        roleResult = await RoleManager.CreateAsync(new IdentityRole(roleName));
                    }
                }

                //Тут створюється супер користувач, з ролею адміністратора.
                //Логін и пароль беруться з файлу налаштувань сервера
                var poweruser = new ApplicationUser {
                    UserName = Configuration["AppSettings:UserName"],
                    Email = Configuration["AppSettings:UserEmail"],
                };
                //Ensure you have these values in your appsettings.json file
                string userPWD = Configuration["AppSettings:UserPassword"];
                var _user = await UserManager.FindByEmailAsync(Configuration["AppSettings:UserEmail"]);

                if (_user == null) {
                    var createPowerUser = await UserManager.CreateAsync(poweruser, userPWD);
                    if (createPowerUser.Succeeded) {
                        //here we tie the new user to the role
                        await UserManager.AddToRoleAsync(poweruser, "Admin");
```

```
            }
        }
    }
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
    if (env.IsDevelopment()) {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    } else {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    //app.UseHttpsRedirection();
    app.UseStaticFiles();
    if (!env.IsDevelopment()) {
        app.UseSpaStaticFiles();
    }

    app.UseRouting();

    app.UseAuthentication();
    app.UseIdentityServer();
    app.UseAuthorization();
    app.UseEndpoints(endpoints => {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller}/{action=Index}/{id?}");
        endpoints.MapRazorPages();
    });

    app.UseSpa(spa => {
        spa.Options.SourcePath = "ClientApp";
        if (env.IsDevelopment()) {
            spa.UseAngularCliServer(npmScript: "start");
        }
    });

    var serviceProvider = app.ApplicationServices.GetService<IServiceProvider>();
    CreateRolesAndUser(app, serviceProvider).Wait();
}
}
```

ProblemsController.cs

```
namespace awesome_map_server.Controllers {

    [Authorize]
    [Route("api/[controller]")]
    [ApiController]
    public class ProblemsController : ControllerBase {
        private IMapper _mapper;
        private IProblemService _problemService;
        public ProblemsController(IMapper autoMapper, IProblemService problemService) {
            _mapper = autoMapper;
            _problemService = problemService;
        }

        // GET: api/Problems
        [HttpGet]
        public async Task<ActionResult<IEnumerable<ProblemViewModel>>> GetProblems() {
            List<Problem> problems = await _problemService.GetProblems();
            List<ProblemViewModel> problemsViewModel = new List<ProblemViewModel>();
            foreach (var item in problems) {
                ProblemViewModel viewModel = _mapper.Map<ProblemViewModel>(item);
                viewModel.ProblemTypes =
_mapper.ProjectTo<ProblemTypeViewModel>(item.ProblemTypeProblems.Select(x =>
x.ProblemType).AsQueryable()).ToList();
                viewModel.SubscribersCount = item.Subscribers.Count;
                viewModel.CommentsLength = item.Comments.Count;
                problemsViewModel.Add(viewModel);
            }
            return problemsViewModel;
        }

        // GET: api/Problems/5
        [HttpGet("{id}")]
        [Authorize(Roles = "Administrator")]
        public ActionResult<ProblemViewModel> GetProblem(Guid id) {
            Problem problem = _problemService.GetProblem(id);

            if (problem == null) {
                return NotFound();
            }

            return _mapper.Map<ProblemViewModel>(problem);
        }

        // PUT: api/Problems/5
        [HttpPut("{id}")]
        public async Task<IActionResult> PutProblem(Guid id, [FromBody] ProblemViewModel problem)
{
            if (id != problem.Id) {
                return BadRequest();
            }
```

```
            try {
                Problem newProblem = _mapper.Map<Problem>(problem);
                foreach (var item in problem.ProblemTypes)
                    newProblem.ProblemTypeProblems.Add(new ProblemTypeProblem() { Problem =
newProblem, ProblemTypeId = item.Id });
                await _problemService.Change(newProblem);
                newProblem = _problemService.GetProblem(newProblem.Id);
                ProblemViewModel loadedProblem = _mapper.Map<ProblemViewModel>(newProblem);
                loadedProblem.ProblemTypes = problem.ProblemTypes;
                loadedProblem.SubscribersCount = newProblem.Subscribers.Count;
                return CreatedAtAction("GetProblem", new { id = newProblem.Id }, loadedProblem);
            } catch (DbUpdateConcurrencyException) {
                if (!_problemService.Exist(id)) {
                    return NotFound();
                } else {
                    throw;
                }
            }
        }

        // POST: api/Problems
        [HttpPost]
        public async Task<ActionResult<ProblemViewModel>> PostProblem(ProblemViewModel
problem) {
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

            Problem newProblem = _mapper.Map<Problem>(problem);
            foreach (var item in problem.ProblemTypes)
                newProblem.ProblemTypeProblems.Add(new ProblemTypeProblem() { Problem =
newProblem, ProblemTypeId = item.Id });
            newProblem.OwnerId = userId;
            await _problemService.Save(newProblem);
            _problemService.Subscribe(newProblem, userId);

            ProblemViewModel loadedProblem = _mapper.Map<ProblemViewModel>(newProblem);
            loadedProblem.ProblemTypes = problem.ProblemTypes;
            loadedProblem.SubscribersCount = newProblem.Subscribers.Count;
            return CreatedAtAction("GetProblem", new { id = newProblem.Id }, loadedProblem);
        }

        // DELETE: api/Problems/5
        [HttpDelete("{id}")]
        public async Task<ActionResult<Problem>> DeleteProblem(Guid id) {
            var problem = _problemService.GetProblem(id);
            if (problem == null) {
                return NotFound();
            }
            _problemService.Delete(problem);
            return problem;
        }
```

```
        [HttpPost("Subscribe")]
        public async Task<IActionResult> Subscribe([FromBody] Guid problemId) {
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

            Problem problem = _problemService.GetProblem(problemId);
            if (problem == null)
                return NotFound();
            try {
                _problemService.Subscribe(problem, userId);

            } catch (Exception e) {
                return BadRequest();
            }
            return Ok(true);
        }

        [HttpPost("Unsubscribe")]
        public async Task<IActionResult> Unsubscribe([FromBody] Guid problemId) {
            string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (!_problemService.Exist(problemId))
                return NotFound();
            try {
                if (await _problemService.Unsubscribe(problemId, userId))
                    return Ok(true);
                else
                    return BadRequest();
            } catch (Exception e) {
                return BadRequest();
            }

        }
    }
}
```

ProblemService.cs

```
public class ProblemService : IProblemService {
        private ApplicationDbContext _context;
        public ProblemService(ApplicationDbContext context) {
            _context = context;
        }

        public async Task Change(Problem problem) {
            problem.UpdateDate = DateTime.Now;
            _context.Entry(problem).State = EntityState.Modified;
            await _context.SaveChangesAsync();
        }

        public async Task Delete(Problem problem) {
            _context.Problems.Remove(problem);
            await _context.SaveChangesAsync();
```

```
    }

    public bool Exist(Guid id) {
       return _context.Problems.Any(e => e.Id == id);
    }

    public Problem GetProblem(Guid id) {
       return _context.Problems.Include(x => x.Files)
        .Include(x => x.Subscribers)
        .Include(x => x.Comments)
        .Include(x => x.ProblemTypeProblems).ThenInclude(x => x.ProblemType)
        .ThenInclude(x => x.Icon).FirstOrDefault(x => x.Id == id);
    }

    public async Task<List<Problem>> GetProblems() {
       return await _context.Problems
        .Include(x => x.Files)
        .Include(x => x.Subscribers)
        .Include(x => x.Comments)
        .Include(x => x.ProblemTypeProblems).ThenInclude(x => x.ProblemType)
        .ThenInclude(x => x.Icon)
        .ToListAsync();
    }

    public async Task Save(Problem newProblem) {
       newProblem.CreateDate = DateTime.Now;

       _context.Problems.Add(newProblem);
       await _context.SaveChangesAsync();
    }

    public void Subscribe(Problem problem, string userId) {
       problem.Subscribers.Add(new ProblemUser() { ProblemId = problem.Id, UserId = userId });
       _context.SaveChanges();
    }

    public async Task<bool> Unsubscribe(Guid problemId, string userId) {
       Problem problem =  GetProblem(problemId);
       if (problem.OwnerId == userId)
          return false;
       _context.ProblemUsers.RemoveRange(_context.ProblemUsers.Where(x => x.ProblemId ==
problemId && x.UserId == userId).ToList());
       _context.SaveChanges();
       return true;
    }
  }
```

createProblemItemContent.dart

```
class CreateProblemItemContent extends StatefulWidget {
  CreateProblemItemContent(
```

```
      {Key key,
      this.isEditMode = false,
      this.problem,
      this.btnOk,
      this.btnCancel,
      this.files,
      this.addFile,
      this.removeFile,
      this.formKey,
      this.first })
      : super(key: key);
  final Problem problem;
  final List<File> files;
  final void Function(File) addFile;
  final void Function(File) removeFile;
  final Widget btnOk;
  final Widget btnCancel;
  final GlobalKey<FormState> formKey;
  final bool isEditMode;
  final bool first;
  @override
  _CreateProblemItemContentState createState() =>
      _CreateProblemItemContentState();
}

class _CreateProblemItemContentState extends State<CreateProblemItemContent> {
  TextEditingController nameController;
  TextEditingController descriptionController;

  @override
  void initState() {
    super.initState();
    nameController = TextEditingController(text: widget.problem.title);
    descriptionController =
        TextEditingController(text: widget.problem.description);
  }

  @override
  void dispose() {
    nameController.dispose();
    descriptionController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      key: widget.formKey,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: <Widget>[
```

```
if (!widget.isEditMode)
  Center(
    child: Container(
        height: 5,
        width: 50,
        decoration: BoxDecoration(
            color: Colors.grey,
            borderRadius: BorderRadius.circular(18))),
  ),
if (!widget.isEditMode) SizedBox(height: 5),
if (!widget.isEditMode)
  Row(
    children: <Widget>[
      Flexible(
        child: TextFormField(
            decoration: InputDecoration(
                border: UnderlineInputBorder(),
                labelText: "Широта",
                hintText: "Широта"),
            keyboardType: TextInputType.number,
            controller: TextEditingController()
              ..text = widget.problem.latitude.toString(),
            onSaved: (String value) {
              widget.problem.latitude = double.parse(value);
            }),
      ),
      SizedBox(width: 50),
      Flexible(
        child: TextFormField(
            decoration: InputDecoration(
                border: UnderlineInputBorder(),
                labelText: "Довгота",
                hintText: "Довгота"),
            keyboardType: TextInputType.number,
            controller: TextEditingController()
              ..text = widget.problem.longitude.toString(),
            onSaved: (String value) {
              widget.problem.longitude = double.parse(value);
            }),
      ),
    ],
  ),
if (widget.problem.problemTypes.length > 0)
  SizedBox(
    height: 10,
  ),
Wrap(
    direction: Axis.horizontal,
    runAlignment: WrapAlignment.start,
    alignment: WrapAlignment.start,
    runSpacing: 0.0,
```

```
    spacing: 0.0,
    crossAxisAlignment: WrapCrossAlignment.start,
    children: <Widget>[
      for (Category item in widget.problem.problemTypes)
        Padding(
          padding: const EdgeInsets.only(right: 5.0),
          child: CategoryItem(
              label: Text(item.name),
              icon: item.icon != null
                  ? Icon(IconData(item.icon.iconCode,
                      fontFamily: item.icon.fontFamily,
                      fontPackage: item.icon.fontPackage))
                  : null,
              onDelete: () {
                widget.problem.removeCategory(item.id);
                setState(() {});
              }),
        )
    ]),
ChooseCategoryAutoComplete(
    getStore: GetIt.I.get<ProblemService>().getCategories,
    selectedCategories: widget.problem.problemTypes,
    validate: widget.problem.problemTypes.length != 0,
    first: widget.first,
    addCategory: (Category category) {
      widget.problem.addCategory(category);
      setState(() {});
    }),
TextFormField(
    maxLines: null,
    keyboardType: TextInputType.multiline,
    controller: nameController,
    decoration: InputDecoration(
        border: UnderlineInputBorder(),
        labelText: "Назва",
        errorText: widget.first && nameController.value.text.isEmpty
            ? "Введіть назву."
            : null),
    onSaved: (String value) {
      widget.problem.title = value;
    }),
SizedBox(height: 10),
TextFormField(
    controller: descriptionController,
    maxLines: null,
    keyboardType: TextInputType.multiline,
    decoration: InputDecoration(
        border: UnderlineInputBorder(),
        labelText: "Опис",
        errorText:
            widget.first && descriptionController.value.text.isEmpty
```

```
                    ? "Введіть опис."
                    : null),
            onSaved: (String value) {
              widget.problem.description = value;
            }),
        SizedBox(height: 10),
        FilePicker(
          first: widget.first,
          validate: widget.problem.files.length != 0,
          addFile: widget.addFile,
          removeFile: widget.removeFile,
          files: widget.files,
        ),
        SizedBox(height: 10),
        Divider(height: 1),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[widget.btnOk, widget.btnCancel],
        ),
      ],
    ),
  );
 }
}
```

ДОДАТОК 3

Засоби вирішення господарських проблем кампуса КПІ на базі ГІС систем

Опис програми

УКР.НТУУ «КПІ». ТІ-6281_20Б

Аркушів 5

Київ 2020

АНОТАЦІЯ

Додаток містить опис основних програмних класів серверної та мобільної частин, що виконують деякі головні функції системи, а саме:

- створює налаштування для серверного застосунку та впроваджує конфігурацію конвеєра обробки запитів;
- реалізує взаємодію з сутностями за допомоги обробки http запитів згідно з REST архітектурою серверів
- реалізує алгоритм створення сутності на серверній та мобільній частині

Мобільний застосунок розроблений за допомогою мови програмування Dart з використанням інструментальних засобів для створення UI користувача – Flutter та серверної частини за допомогою технології ASP.NET Core, яка взаємодіє з базою даних SQL Server.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку розміщено опис одних з основних компонентів системи «Awesome Map», що реалізують алгоритм створення сутності, в даному випадку «Проблеми», в розділі 4, рисунка 4.2 та описують налаштування сервера, включаючи конвеєр обробки http запитів сервера.

Система має три окремих компонента, що взаємодіють між собою:

- клієнт - мобільний додаток, що реалізований на фреймворку Flutter;
- сервер, що реалізований на фреймворку ASP.NET;
- база даних, на SQL Server, створена методом Code First за допомогою Entity Framework Core.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Додаток надає реалізацію функціональності створення проблеми в системі, впроваджуючи:

- форму на мобільному застосунку;
- обробники http запита за REST архітектурою;
- реалізацію сервісу (бізнес логіки), що зберігає сутність до бази даних.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Згідно з багаторівневою архітектурою системи, що складається з 4 рівнів (детальніше в розділі 4 підрозділі 4.5 пункті 4.5.2).

Мобільний додаток впроваджує форму для заповнення та створення, так званої, view model – абстрактне представлення сутності, що відкриває загальнодоступні властивості. Після валідації даних, якщо все заповнено вірно, застосунок створить view model проблеми та за допомогою http клієнта відправляє POST запит на сервер, включаючи view model, як тіло запита, та додаючи в заголовки запита JWT токен користувача.

На сервері запит обробляється відповідним методом контролера, в даному випадку ProblemsController та методом PostProblem.

Згідно багаторівневої архітектури описаній в розділі 4, підрозділі 4.5 пункті 4.5.2, це вважається прикладним рівнем.

В цьому методі, за допомоги заздалегідь правильних налаштувань авторизації сервера, система має доступ до JWT токена з заголовку запита, який дозволяє дізнатися ідентифікатор користувача. Якщо цей токен невалідний, то система автоматично відхилить запит з відповіддю 403 – Forbidden (Заборонено), в якості статусного кода, про що дізнається мобільний додаток та відреагує відповідно.

Якщо автентифікація буде успішною, то за допомогою бібліотеки AutoMapper та її налаштувань, створюється з view model сутність Entity Framework-а, тобто модель постійного сховища даних. Після ця модель передається на обробку відповідної реалізації контракту IProblemService – сервісу ProblemService в метод Save, що виступає вже рівнем бізнес логіки.

В сервісі проблем модель зберігається до бази даних, додаючи необхідне поле - дату створення. Після вдалого зберігання управління системи передається знову до контролера, що повинен ще підписати користувача (власника) на цю проблему, який в подальшому не зможе від неї відписатись, бо він є її створювач. Тому контролер

знову передає оновлену вже сутність проблеми до сервісу, в метод Subscribe, додаючи ідентифікатор користувача отриманий з JWT токена.

В реалізації сервісу, система додає користувача до сутності в якості підписника та зберігає знову до бази даних.

Після цього, контролер отримує управління ресурсами та за допомогою бібліотеки AutoMapper, оновлена вже двічі сутність проблеми переводиться до представлення view model, яка вже буде повертатися в мобільний додаток в якості відповіді на http запит зі статусним кодом 200.

Нарешті мобільний додаток отримує створену модель та відображає її на карті, додаючи її в своє сховище сутностей.

ДОДАТОК 4

Засоби вирішення господарських проблем кампуса КПІ на базі ГІС систем

Публікації

18 - Міжнародна науково-практична конференція

УКР.НТУУ «КПІ». ТІ-6281_20Б

Аркушів 2

Київ 2020

Система «Awesome Map KPI» сучасний засіб моніторингу господарських проблем університету

Гавриляк Олександр Володимирович
Гагарін Олександр Олександрович

В наш час діджеталізації створюється потреба в моніторингу господарських проблем та публічних заходів промислового району для ефективного вироблення управлінських рішень. Системи що вирішують зазначену проблему все більш реалізуються у вигляді сервісних центрів з віддаленим доступом з спеціалізованих мобільних додатків.

Пропонуєма система «Awesome Map KPI» вирішить, як муніципальні проблеми університету, наприклад, повний бак сміття біля гуртожитку, так і життєво-небезпечні проблеми забезпечення порядку та закону, наприклад, група п'яних чоловіків пристає до людей в парку.

Система що розробляється має клієнт серверну архітектуру та вирішує задачі моніторингу наявних господарських проблем (порив водо-,тепло-, електричних мереж, наявність незручностей, сміття та таке інше), а також планування та проведення публічних заходів на території університету. Серверна частина організує ведення бази даних проблем що виникають, їх фіксацію та доступ до інформації необхідної для вироблення управлінських завданий. Збір проблемної інформації та зведення її до єдиного сервісного центру виконується користувачами системи за допомогою мобільного додатку.

Подібні мобільні додатки починаються розробляти не тільки в Україні, а й по всьому світу, наприклад додаток для Los Angeles «MyLA311»[1], або для всього Пакістану – «Pakistan Citizen Portal»[2].

Створення системи потребувала розроблення концептуальної схеми бази даних, розробки алгоритмів відбору потрібної по запиту інформації та створення форм для забезпечення UX-інтерфейсу користувача. Система розроблялась за допомогою сучасних технологій: Google Cloud Platform, сучасного інструмента користувальницького інтерфейсу для мобільних додатків від Google – Flutter, та останніх технологій від Microsoft, для створення серверної частини системи – ASP.NET Core

Бета версія системи та апробація всіх її компонентів у наступний час завершені. З повною версією опису системи можна ознайомитися на сайте apeps.kpi.ua у розділі студентські випускні роботи.

Література:

1.Мобільний додаток «MyLa311» від City of Los Angeles [Електроний ресурс] Режим доступу: https://play.google.com/store/apps/details?id=com.LA.MyLA311

2.Мобільний додаток «Pakistan Citizen Portal» від National IT Board, Government Of Pakistan [Електроний ресурс] Режим доступу: https://play.google.com/store/apps/details?id=com.govpk.citizensportal