Міністерство науки та освіти України
Харківський національний університет радіоелектроніки

Звіт до лабораторної роботи №4
з дисципліни «Аналіз та рефакторінг коду програмного забезпечення»
На тему: «Програмна система для виявлення задимлення у приміщенні та допомоги при евакуації»

Виконав:
ст. гр. ПЗПІ-19-3
Логвінов О. В.

Перевірив:
ст. викл. каф. ПІ
Сокорчук І.П

Харків 2021

Тема роботи: Розробка мобільної частини програмної системи для виявлення задимлення у приміщенні та допомоги при евакуації.

Мета роботи: Розробити мобільний застосунок на платформі Android використовуючи технологію Xamarin Forms. Створити розмітку екранів а також додаткові елементи відображення, забезпечити отримання даних з серверу за допомогою API та налаштувати передачу даних через Web Sockets. Відобразити отриманні данні на відповідних сторінках застосунку. Передбачити локалізацію та інтернаціоналізацію інтерфейсу користувача.

Посилання на файл з кодом на гугл-диску:
https://drive.google.com/file/d/1StgKSXOXyNQyqYl2mcfONxBTLZ_Y-_2C/view?usp=sharing

MD5 хеш для pzpi-19-3-lohvinov-oleksandr-lab4.zip:
2f921c79aeaf213572b9889ed605c18b

Хід роботи:

1) Створення розміток екранів (ContentPage) та представлень (ContentView), що можуть бути повторно використані у декількох ContentPage.
2) Написання http сервісів, що використовують та реалізують бізнес логіку програмної системи, а також сервісу, що відповідає за встановлення зв'язку використовуючи технологію Web Sockets.
3) Реалізація захисту системи шляхом використовування JWT-токену підчас запитів до сервера.
4) Локалізація та інтернаціоналізація інтерфейсу користувача
5) Побудова UML діаграм: UML діаграму прецедентів (Use case diagram), UML діаграму компонентів (Component Diagram), UML діаграму взаємодії (Interaction Overview Diagram), UML діаграму пакетів (Package Diagram);

Для відображення послуг та дій, які система може виконувати, була розроблена UseCase діаграма (див. додаток А). Згідно з цією діаграмою, видно,

що акторами можуть бути наступні користувачі: відповідальні особи, зареєстрований користувач та гість.

Відповідальна людина може налаштовувати масштаби приміщення для більш точного відображення користувачів на плані евакуації та перегляд усіх приміщень у будівлі Також даній ролі доступний увесь функціонал, що і для звичайного авторизованого користувача.

Авторизованому користувачу доступні такі дії як:

- Змінити свої дані
- Переглянути інформацію про поточне приміщення, у якому він знаходиться
- Увійти в інше приміщення. У разі наявності тесту, його потрібно пройти його перед входженням.
- Повідомити відповідальну людину про небезпеку у приміщення, шляхом надсилання повідомлення у додатку або подзвонивши йому.
- Отримати план евакуації у разі надзвичайної ситуації
- Під час евакуації, у разі блокування шляху, повідомити про це відповідальній особі.

Якщо у користувача немає профілю у додатку, він може зайти систему, використовуючи роль гостя. Після входу у додаток, користувачу доступний той же функціонал, що і авторизованій людині.

Для спрощення створення інтерфейсу та впровадження логіки для сторінки був використаний патерн проектування MVVM (Model – View - ViewModel). Цей патерн розділяє код на три частини – модель даних, що використовується на сторінці, представлення – розмітка сторінки та модель представлення, що виконує основну логіку сторінки а також забезпечує оновлення інтерфейсу у разі зміни даних у моделі. Для реалізації цієї можливості кожна модель даних спадкується від класу BaseViewModel. Код класу наведений у додатку Д.

Для зображення того, з яких елементів складається система, було створено діаграму компонентів (див. додаток Б). Зі схеми видно, що головним файлом у програмі є діяльність MainActivity.cs. У коді даного файлу ініціалізуються бібліотеки та модулі, що забезпечують коректну роботу додатка, після чого відображається вміст сторінок додатку. Зокрема для зменшення залежностей між моделями представлення та сервісами використовується IoC(Inversion of Control), що реалізується за допомогою використання бібліотеки TinyIoC.

Для використання доступні наступні сторінки:

- сторінка авторизації
- сторінка зміни персональних даних
- сторінка перегляду приміщень та налаштування їх масштабу, де відповідальна особа може вибрати приміщення, після чого їй буде наданий зображення плану евакуації з відміченими точками масштабу. Відповідальна особа може переназначити світові координати, ставши у потрібне місце у приміщенні та відправивши своє поточне місцезнаходження.
- сторінка проходження тесту перед входом до приміщення
- сторінка зображення інформації про приміщення. Зображується назва, опис приміщення та надається контактна інформація з відповідальними особами будівлі.
- сторінка навігації по додатку
- сторінка, що зображує план евакуації під час надзвичайної ситуації, де користувачу надаються плани евакуації та зображується шлях до виходу з приміщення

Класам ContentPage відповідають .xaml файли, що містять розмітку сторінок. Для зображення цих, а також допоміжних файлів, наприклад файлів, що містять переклад, була створена діаграма пакетів, що зображена у додатку В. На ній зображені пакети, що відповідають за створення розширених елементів

відображення таких як карта або текстовий елемент у якому можливе вирівнювання тексту (стандартний елемент цієї можливості не надає). Що стосується сторінок, то серед них можна виокремити навігаційні сторінки та сторінки, що забезпечують безпосередньо логіку програми. Також, для покращення взаємодії з додатком були створені спливаючі вікна. Вони розроблені за допомогою бібліотеки Rg.Plugins.Popup. Файли перекладу зберігаються у пакеті ресурсів (Resx).

Для опису взаємодії додатка з клієнтом під час евакуації буда розроблена діаграма взаємодій (див. додаток Г). Після отримання сигналу про тривогу, додаток автоматично відкриває сторінку з планом евакуації. Після чого отримується шлях евакуації та надається інтерфейс взаємодії.

Для спрощення створення запитів до серверу були розроблені методи, які інкапсулюють процес створення запиту а також додавання тіла та спеціальних заголовків, таких як Authorization, до запиту (див. додаток Е). Для організації з'єднання клієнту та додатку через Web Sockets використовувалася бібліотека Microsoft.AspNetCore.SignalR.Client. Код реалізації зображений у додатку Є.

Кожна сторінка сайту підтримує локалізацію та інтернаціоналізацію. Це зроблено шляхом створення окремих файлів AppResources.uk.resx та AppResources.resx, які використовує бібліотека Xamarin.CommunityToolkit. За перемикання мови відповідає кнопки мова у навігаційній панелі та на сторінці реєстрації

Висновки: під час лабораторної роботи була розроблена мобільна версія системи виявлення задимлення у приміщенні та допомоги при евакуації: спроектовано розмітку сторінок додатку, налаштована маршрутизація, налаштовано отримання та відправлення інформації на сервер, організовано отримання повідомлень з серверу через використання технології SignalR, передбачено інтернаціоналізацію та локалізацію інтерфейсу сайту

Додаток А

Діаграма прецедентів, що відображає відносини між акторами та прецедентами

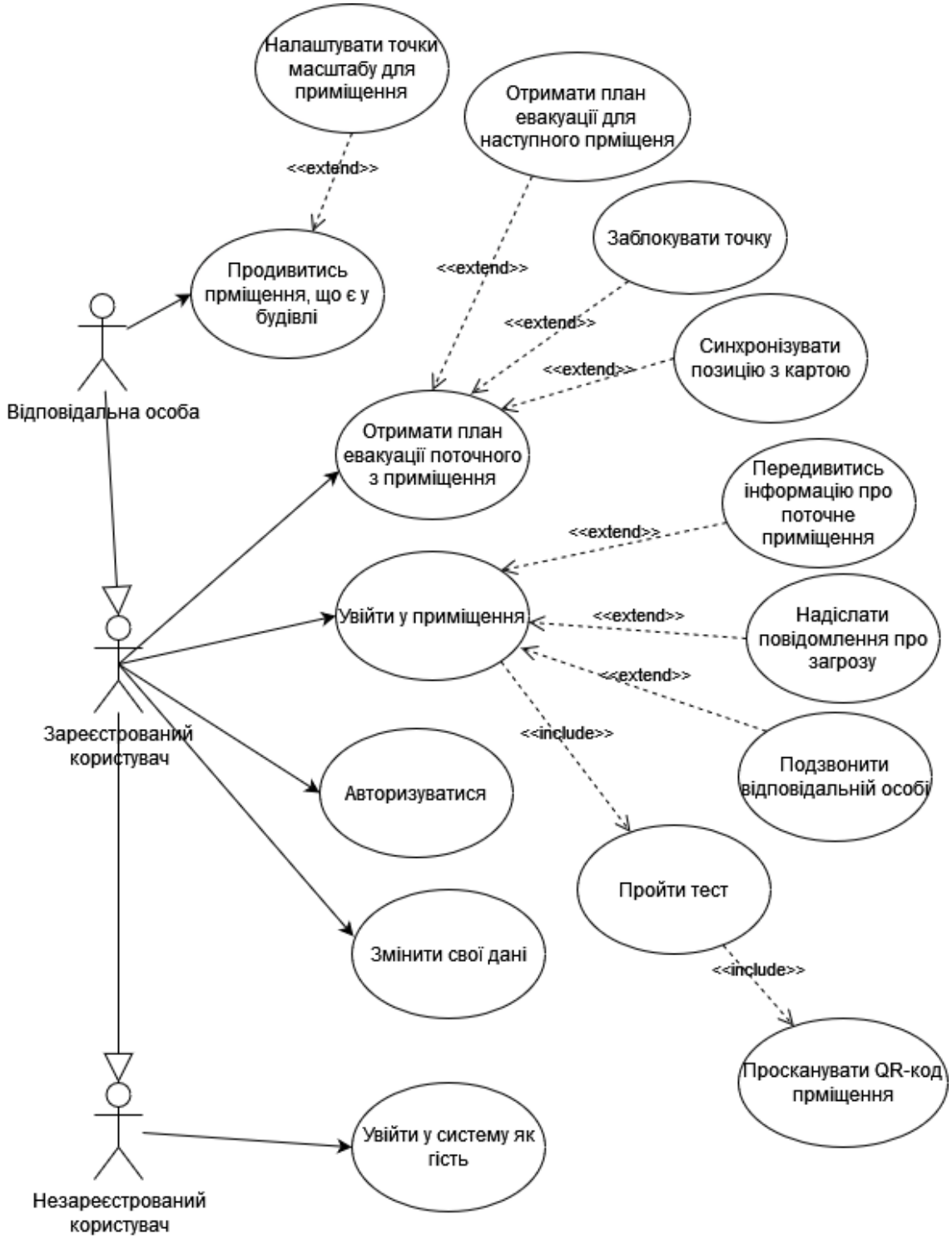

Рисунок А.1 – Діаграма прецедентів мобільного програмної система для виявлення задимлення у приміщенні та допомоги при евакуації

Додаток Б

Діаграма компонентів, що описує особливості фізичного представлення системи

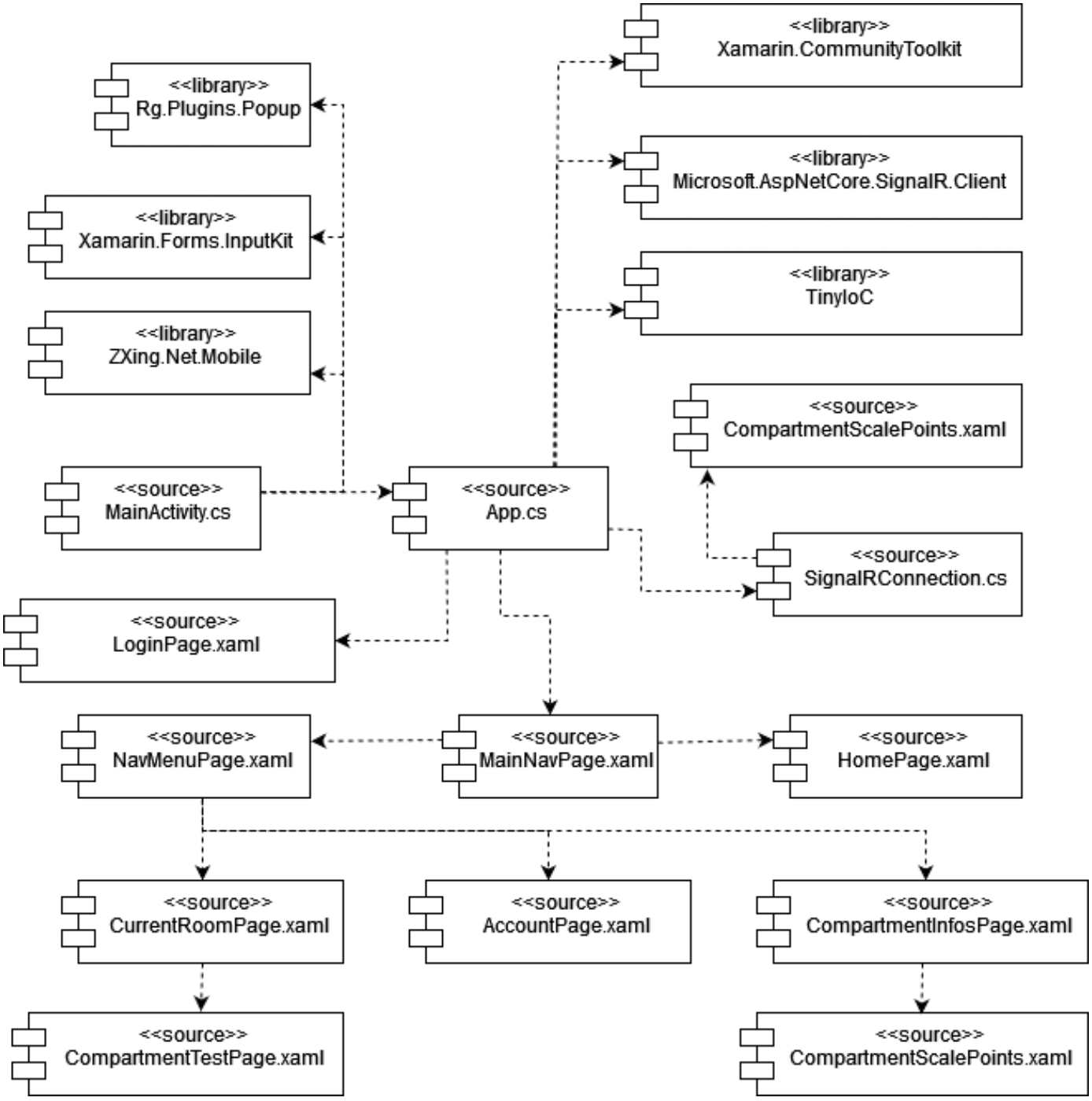

Рисунок Б.1 – Діаграма компонентів програмної системи виявлення задимлення у приміщенні та допомоги при евакуації

Додаток В

Діаграма пакетів, що відображає пакети класів та залежності між ними

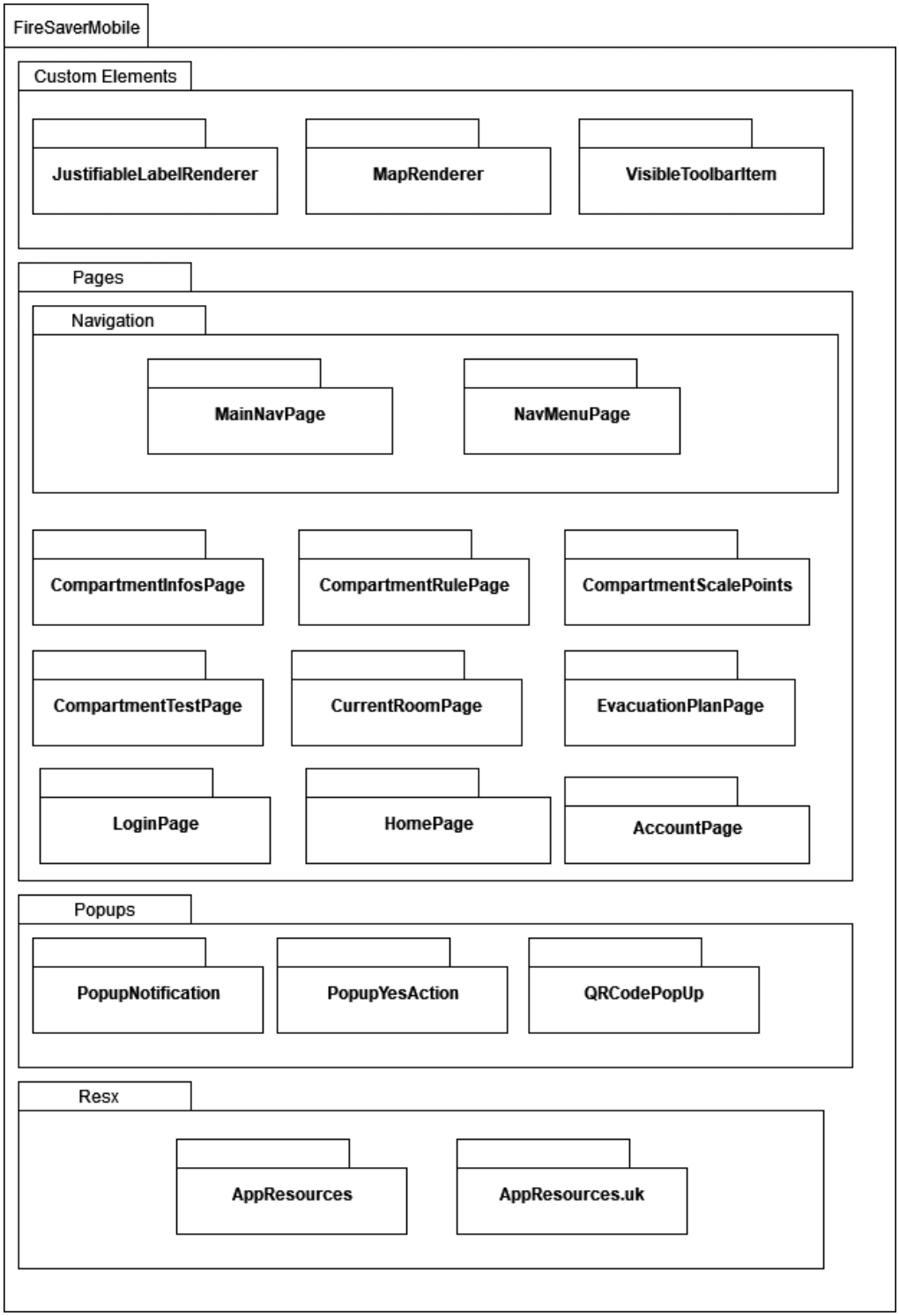

Рисунок В.1 – діаграма пакетів програмної системи програмної системи виявлення задимлення у приміщенні та допомоги при евакуації

Додаток Г

Діаграма взаємодії – зображує процес взаємодії додатка з користувачем під час евакуації

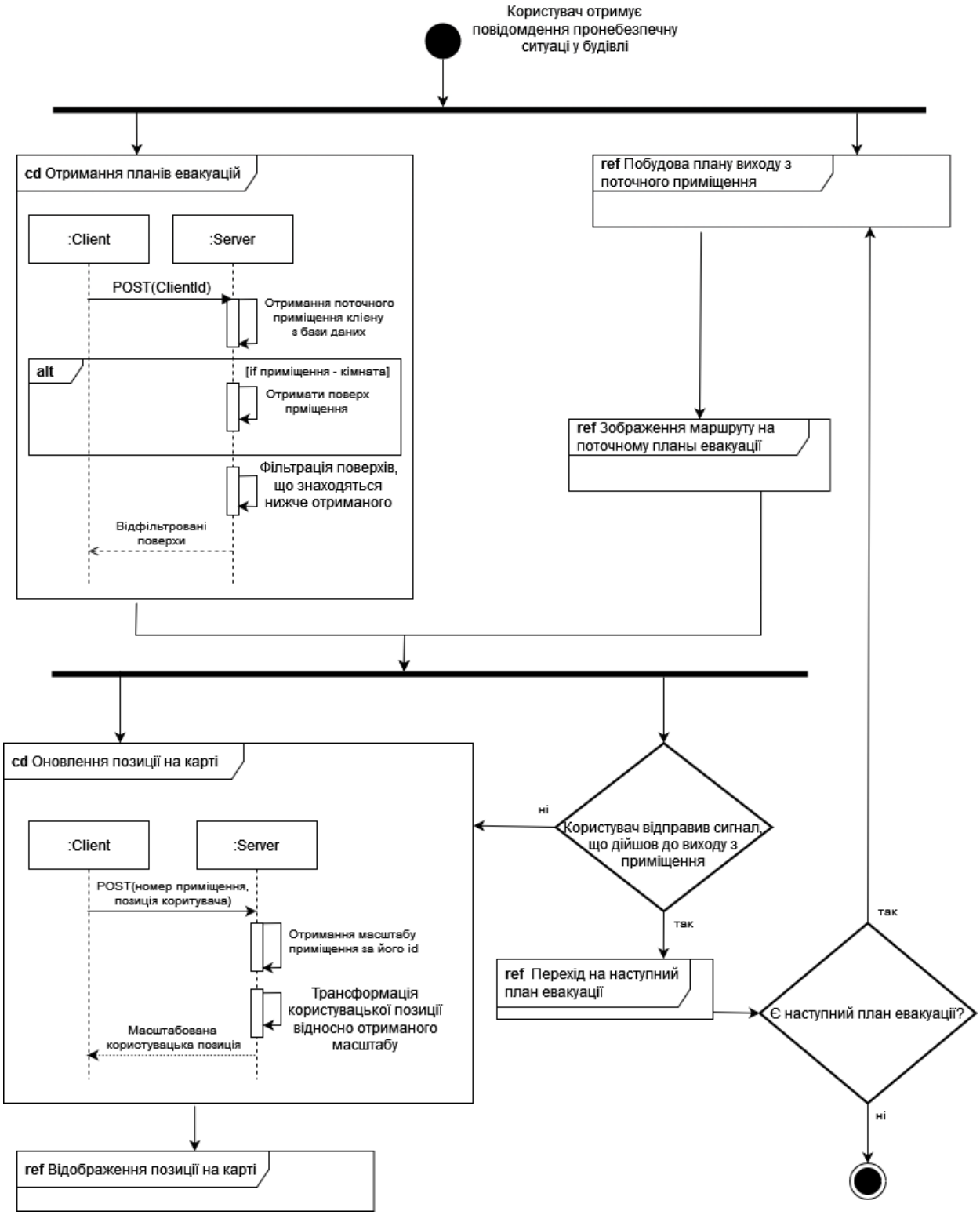

Рисунок Г.1 – Діаграма взаємодії мобільного додатка та користувача

Додаток Д

Код, що забезпечує прив'язку даних до інтерфейсу користувача

```
public class BaseViewModel: INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string
                                        ropertyName = null)
    {
        PropertyChanged(this, new
                                  PropertyChangedEventArgs(propertyName));
    }

    protected void SetValue<T>(ref T backingField, T value,
        [CallerMemberName] string propertyName = null)
    {
        if (EqualityComparer<T>.Default.Equals(backingField, value))
            return;

        backingField = value;

        OnPropertyChanged(propertyName);
    }
}
```

Приклад застосування прив'язки даних

```
public class AuthentificationInput:BaseViewModel
{
    private string mail = "";
    public string Mail
    {
        get { return mail; }
        set { SetValue(ref mail, value); }
    }
    private string pass = "";
    public string Password
    {
        get { return pass; }
        set { SetValue(ref pass, value); }
    }
}
```

Додаток Е

Код, для відправки POST запиту на сервер

```
1  public static async Task<HttpResponseMessage> PostRequest<T>(this T
2                                      model, HttpClient client, string url)
3  {
4      var token = await SecureStorage.GetAsync("token");
5      client.DefaultRequestHeaders.Authorization =
6          new AuthenticationHeaderValue("Bearer", token);
7
8      HttpContent content = await prePostPutRequestWork(model);
9
10     HttpResponseMessage response = await client.PostAsync(url, content);
11
12     return response;
13 }
14
15 private static async Task<HttpContent> prePostPutRequestWork<T>(T model)
16         {
17             var serializerSettings = new JsonSerializerSettings();
18             serializerSettings.ContractResolver =
19                 new CamelCasePropertyNamesContractResolver();
20
21             string json = await JsonConvert.SerializeObjectAsync(model,
22                 Formatting.None, serializerSettings);
23
24             HttpContent content = new StringContent(json, Encoding.UTF8,
25                 "application/json");
26
27             return content;
28         }
```

Додаток Е

Частина реалізації класу, що забезпечує Web Socket зв'язок із сервером

```
private SignalRConnectionHelper(string connectionUrl)
{
    hubConnection = new HubConnectionBuilder().WithUrl(connectionUrl,
Microsoft.AspNetCore.Http.Connections.HttpTransportType.WebSockets, (con)
    =>{
       con.SkipNegotiation = true;
       con.AccessTokenProvider = async () =>
   { var userData = await retrieveAuthValues(); return userData.Token; };
      }).Build();
      Task.Run(async () =>
      {
          await Connect();
      });
 }

public async Task Connect()
{
    if (IsConnected)
        return;
    try
    {
        await hubConnection.StartAsync();
        IsConnected = true;
     }
     catch (Exception ex)
     {
        await PopupNavigation.Instance.PushAsync(
            new PopupNotificationView("Unable to connect to server",
            MessageType.Warning));
      }
      hubConnection.Closed += async (error) => {
         IsConnected = false;
         await Task.Delay(5000);
         await Connect();
      }
```