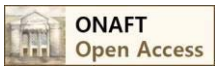
УДК 004.415.2.031.43

ДОСЛІДЖЕННЯ РОБОТИ ТА РОЗРОБКА ПРОТОТИПНОЇ МОДЕЛІ МОБІЛЬНОГО ДОДАТКУ ВИЗНАЧЕННЯ ОПТИМАЛЬНОГО ШЛЯХУ КАРЕТ ШВИДКОЇ ДОПОМОГИ

Балинський В.В.[1], Бодюл О.С.[2]

Одеська національна академія харчових технологій, Одеса, Україна
E-mail: [1]vadym.exi@gmail.com, [2]bodyulolena@ukr.net
ORCID: [2]https://orcid.org/0000-0001-9925-434X

Copyright © 2021 by author and the journal "Automation of technological and business – processes".
This work is licensed under the Creative Commons Attribution International License (CC BY).
http://creativecommons.org/licanses/by/4.0

DOI: 10.15673/atbp.v13i4.2221

Анотація. *Безумовно одною з найважливіших галузей впровадження інформаційних технологій є така, що тісно пов'язана з забезпеченням повсякденних потреб людини та її безпеки. Особу увагу привертають розробки, які спрямовані допомогти людині в екстреній ситуації. Прикладом може бути програмний продукт, який дозволить користувачам викликати швидку допомогу одним натисканням кнопки і відслідковувати її місцезнаходження в дорозі у разі погіршення стану здоров'я людини. Важливою функцією для користувача-пацієнта є забезпечення безперервного відстеження карети швидкої допомоги впродовж всього шляху пересування карети швидкої допомоги. Реалізація такого рішення потребує забезпечення надійної роботи, цілісності внутрішніх компонентів, легку доступність і зручне використання. Метою даної роботи є створення програмного продукту, який допомогає водіям бригад швидкої допомоги прокладати оптимальний маршрут до місця виклику, а, також, дозволяє пацієнтам спостерігати за пересуванням карети та визначати час прибуття медиків. В системі розроблено надійно-працюючий, виділений сервер, який обробляє дані, такі як: інформацію про користувачів, виклики, бригади, карети. Реалізовано можливість в режимі реального часу отримувати інформацію про місце перебування карети, яка знаходиться в дорозі до пацієнта. Положення об'єкта обчислюється завдяки використанню розміщеного на ньому GPS-приймача, який приймає та обробляє сигнали супутників космічного сегменту GPS – системи глобального позиціонування. Для реалізації була використана мова програмування C#, фреймворк ASP.NET Core, система управління базами даних PostgreSQL, середовище розробки JetBrains Rider, платформа для тестового хостингу сервера Heroku. Результатом розробки є мобільний додаток, який дозволить користувачу викликати швидку допомогу і відстежувати свою карету. Для водіїв карет забезпечується зручна робота при виконанні своїх обов'язків. Взаємодія між мобільними додатками здійснюється за допомогою серверу, яких працює з усіма користувачами мобільних додатків, забезпечуючи швидку і своєчасну передачу даних.*

Abstract: *Undoubtedly, one of the most important areas of information technology implementation is closely related to the provision of everyday human needs and security. Special attention is drawn to developments that are designed to help people in an emergency. An example is a software product that allows users to call an ambulance at the touch of a button and track its location on the road in the event of deteriorating human health. An important function for the patient-user is to ensure continuous tracking of the ambulance throughout the movement of the ambulance. The implementation of such a solution requires reliable operation, integrity of internal components, easy accessibility and ease of use. The aim of this work is to create a software product that helps ambulance crew drivers to plan the optimal route to the place of call, as well as allows patients to monitor the movement of the carriage and determine the time of arrival of medics. The system has developed a reliable, dedicated server that processes data such as: user information, calls, crews, carriages. The possibility of receiving real-time information about the location of the carriage on the way to the patient has been implemented. The position of the object is calculated by using a GPS receiver placed on it, which receives and processes signals from satellites of the GPS space segment - the global positioning system. The C # programming language, the ASP.NET Core framework, the PostgreSQL database management system, the JetBrains Rider development environment, and the Heroku server test hosting platform were used for implementation. The result of the development is a mobile application that will allow the user to call an ambulance and track their carriage. For drivers of carriages convenient work at performance of the duties is provided. Interaction between mobile applications is carried out by means of the server which works with all users of mobile applications, providing fast and timely data transfer.*

Ключові слова: додаток в режимі реального часу, швидка допомога, база даних, технологія «клієнт-сервер», інформаційна модель.
Key words: real-time application, ambulance, database, client-server technology, information model.

Вступ

Перспективи комп'ютерних технологій, їх розвиток мають величезний потенціал, який не обмежується нашим баченням напрямів сучасного прогресу. Розвиток технологій відбувається настільки швидко, що в недалекому майбутньому IT може постати перед нами в такому виді, в якому ми собі даже і не уявляли б. В теперішній час безумовно важливим є розвиток інформаційних технологій і галузей, які тісно пов'язані з забезпеченням повсякденних потреб людини та її безпеки. Величезне значення в нашому житті посіли мобільні пристрої. Вони є практично у кожної людини. У повсякденному житті смартфони допомагають нам спілкуватися з людьми, переглядати інформацію в інтернеті, користуватися зручними гаджетами і завантажувати додатки, які у деяких випадках стають незамінними для забезпечення комунікації. Розробка мобільних додатків набрала гігантських обертів. Каталог таких додатків налічує величезну кількість розроблених концепцій. У своїй більшості вони мають розважальний характер, але деякі програми можуть навіть допомогти людині в екстреній ситуації, прикладом якої є потреба у наданні швидкої медичної допомоги.

Швидкість реагування на виклик людини є найважливішим чинником роботи невідкладної швидкої допомоги. У свою чергу, людині потрібно мати інформацію про карету, що виїхала до неї, таку як її місцезнаходження та час прибуття. У такій ситуації не можна недооцінювати психологічний стан людей, який точно покращиться, якщо надати їм можливість бачити карету, що рухається до їх місця розташування. Можливість переглянути цю інформацію на підсвідомому рівні буде заспокоювати людину завдяки можливості спостерігати за місцем знаходження карети у додатку в реальному часі.

Теоретична частина

Основним завданням швидкої допомоги є своєчасна реакція на виклик і швидке надання допомоги людині, яка зробила виклик. Для обробки викликів від людей існують оператори. Вони обробляють всі дзвінки та передають інформацію бригадам швидкої допомоги. Бригади, в свою чергу, приступають до виконання, виїжджаючи на кареті з підстанції до пацієнта. Однією з організаційних проблем надання невідкладної допомоги є те, що на прийняття виклику від людини витрачається деякий час. Оператор повинен взяти трубку, записати виклик, повідомити бригаду про місцезнаходження людини, яка викликала швидку. Все це займає певний час і може мати непередбачені наслідки, коли мова йде про порятунок людини. У таких ситуаціях будь-яке зменшення витраченого часу може врятувати людині життя. Аналізуючи цю проблему, скорочення часу на обробку викликів можливе за допомогою виклику швидкої допомоги через мобільний додаток. Досить відкрити його і натисканням однієї кнопки викликати карету. Потрібно прийняти до уваги те, що швидкість роботи сервера в таких випадках є найважливішим фактором, тому що виклик повинен бути отриманий та оброблений негайно без будь-якої можливої затримки. Також важливою проблемою при виконанні своїх обов'язків бригадою є те, що маршрут, яким вони направляються до людини, може бути далеко не оптимальним. Рішенням такої проблеми є автоматичне прокладання шляху для карети швидкої допомоги від станції до пацієнта, який її викликає. В такому випадку, водій карети може дуже швидко дістатися до людини, витративши набагато менше часу.

Дане рішення дозволяє реалізувати також іншу функцію – відстеження карети швидкої допомоги в мобільному додатку. Людина, яка викликала швидку, може в режимі реального часу відслідковувати місце знаходження карети, що прямує до неї. Пацієнт також матиме можливість бачити оптимальний шлях та спостерігати за пересуванням карети в дорозі. В цьому допомагатиме GPS, сукупність радіоелектронних засобів, що дозволяє визначати положення та швидкість руху об'єкта на поверхні Землі або в атмосфері. Положення об'єкта обчислюється завдяки використанню розміщеного на ньому GPS-приймача, який приймає та обробляє сигнали супутників космічного сегменту системи глобального позиціонування GPS. Таким чином, людина володіє інформацією про реальний час прибуття карети швидкої, що може допомогти їй, як в психологічному плані, так і в рішенні самостійного усунення проблем.

Ще однією важливою особливістю є реалізація асинхронних методів для роботи програми в декількох потоках. Асинхронність в програмуванні – виконання процесу в неблокуючому (для головного потоку) режимі системного виклику, що дозволяє додатковим потокам програми продовжити обробку. Питання в тому, як реалізувати взаємодію пацієнта і водія карети, обмін даними між ними. Рішенням є створення так званих умовних кімнат – хабів (англ. hub) на стороні сервера, в яких знаходяться пацієнт і водій. Пацієнт отримує в режимі реального часу дані від водія швидкої. Таких хабів може бути скільки завгодно, по одному на пацієнта і водія. Самі по собі хаби існують окремо і не взаємодіють один з одним, тому робота потоків даних на такому сервері повинна проходити асинхронно, не зупиняючи інші потоки. Асинхронний код виносить операцію з основного потоку програми в додатковий потік, який відпрацьовує в паралельному режимі не зупиняючи при цьому основний послідовний процес обробки коду.

Якщо описати процес роботи таких хабів покроково, то він буде виглядати наступним чином:

- пацієнт викликає швидку допомогу в мобільному додатку;
- на сервері створюється новий хаб з поточним пацієнтом і чекає прийняття персоналом швидкої допомоги даного виклику;
- коли виклик приймається, в хаб додається водій карети;

- хаб обробляє дані місця розташування пацієнта і водія, тим самим створюючи оптимальний шлях за допомогою маршрутних даних *OpenStreetMap* або *GoogleDirectionsApi*;
- маршрут відображається як у водія швидкої допомоги, так і у пацієнта, для того щоб він бачив процес пересування карети швидкої допомоги;
- дані у пацієнта на екрані обробляються в режимі реального часу для постійного оновлення місця знаходження карети;
- коли карета прибуває, хаб на сервері видаляється.

Практична частина

Для вирішення поставленої проблеми були визначені наступні задачі:
- розробити архітектуру real-time мобільного додатку;
- розробити повнофункціональний сервер для взаємодії з мобільним додатком;
- створити зручну реєстрацію та авторизацію користувачів;
- забезпечити можливість зручного виклику швидкої допомоги через додаток;
- реалізувати можливість відстеження карети швидкої допомоги;
- надати користувачеві особистий профіль.

На рисунку 1 наведена інформаційна модель системи. На ній зображені модулі та їх взаємодія. Модулі розподілені на Сервер, API та Клієнт.

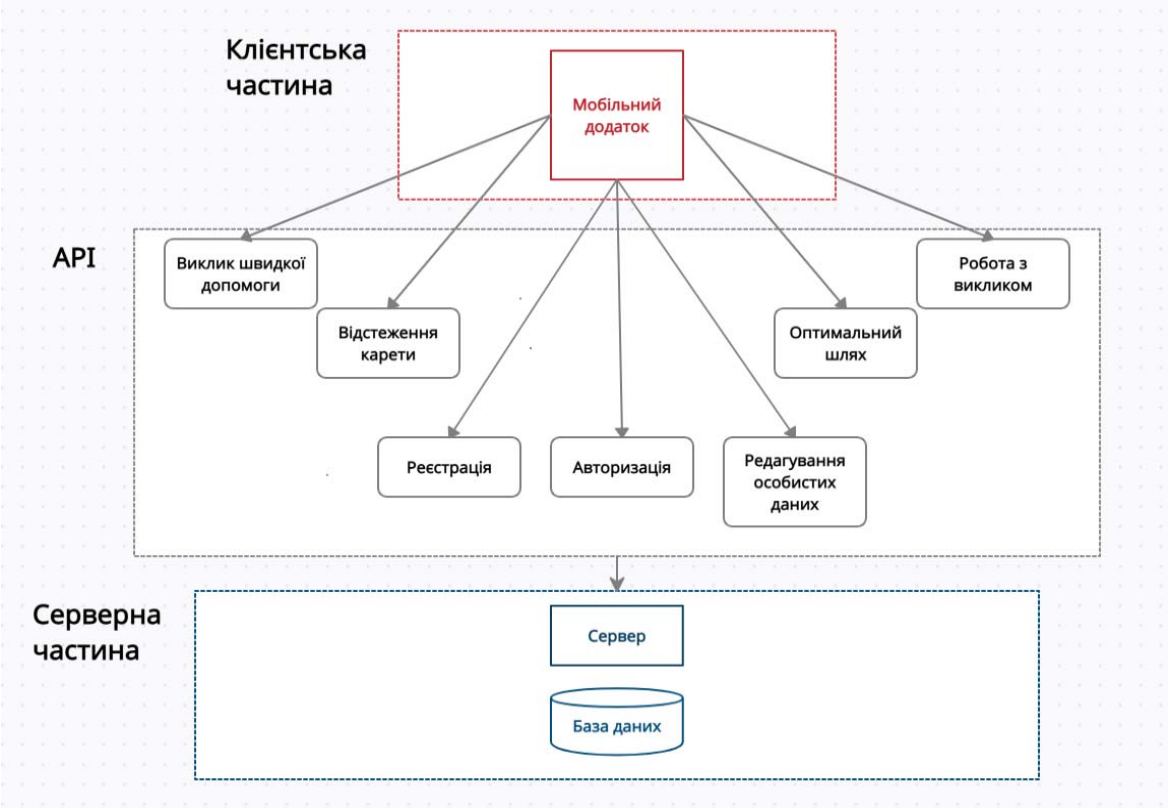


Рис. 1 – Інформаційна модель системи

В основі реалізації даного програмного продукту лежить архітектура клієнт-сервер. Клієнт-серверну архітектуру можна означити як концепцію інформаційної мережі, в якій основна частина її ресурсів зосереджена в серверах, які обслуговують своїх клієнтів. Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Така архітектура має наступні компоненти: набір серверів, які обслуговують інформацією програми, які до них звертаються; набір клієнтів, які звертаються до сервісів, що надаються серверами; мережу, яка забезпечує зв'язок між цими компонентами.

Оскільки розробка сервера є важливою частиною будь-який клієнт-серверної архітектури, для реалізації було використано фреймворк ASP.NET Core, якій надає відмінну функціональність для розробки високонавантажених серверів. Фреймворк побудований з набору незалежних компонентів і дає можливість або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або створити і застосовувати свої компоненти зі своїм функціоналом. Також використовується ряд бібліотек: SignalR – для реал-тайм реалізації клієнт-серверної архітектури, NewtonsoftJson – для роботи з json-форматом, JwtBearer – для роботи з токенами та інші компоненти.

Клієнтська частина програмного продукту являє собою мобільний додаток. У ньому користувач може зареєструватися, авторизуватися, редагувати свої персональні дані та викликати швидку допомогу. Якщо користувач

має роль водія карети, то він може бачити місцезнаходження пацієнта на мапі. Надають весь цей функціонал так звані API – програмний інтерфейс додатку для звернення до сервера. Через методи цього інтерфейсу відбувається взаємодія між сервером і клієнтом. У свою чергу на сервері знаходиться база даних, яка працює зі збереженням, редагуванням даних.

Для реалізації програмного продукту була використана мова програмування C#, система управління базами даних – PostgreSQL, середовище розробки – JetBrains Rider, платформа для тестового хостингу сервера Heroku, мобільний додаток використовує фреймворк Xamarin.Forms.

Для зв'язку бази даних PostgreSQL з сервером використовується технологія ORM під назвою Entity Framework Core. За допомогою даної ORM класи, описані програмним кодом можна зіставляти з таблицями в базі даних.

```
services.AddDbContext<AmbulanceSystemContext>(options =>
{
  var connUrl = Environment.GetEnvironmentVariable("DATABASE_URL");
  connUrl = connUrl.Replace("postgres://", string.Empty);
  var pgUserPass = connUrl.Split("@")[0];
  var pgHostPortDb = connUrl.Split("@")[1];
  var pgHostPort = pgHostPortDb.Split("/")[0];
  var pgDb = pgHostPortDb.Split("/")[1];
  var pgUser = pgUserPass.Split(":")[0];
  var pgPass = pgUserPass.Split(":")[1];
  var pgHost = pgHostPort.Split(":")[0];
  var pgPort = pgHostPort.Split(":")[1];
     string connStr = $"Server={pgHost};Port={pgPort};User
     Id={pgUser};Password={pgPass};Database={pgDb};sslmode=Prefer;Trust Server Certificate=true";
  options.UseNpgsql(connStr);
});
```

У даній ділянці коду, спочатку зчитується змінна навколишнього середовища системи, на якому сервер розгортається. Потім парсингом з цієї змінної зчитуються дані про порти і хости. На основі цих даних створюється рядок для підключення до бази даних. Параметр sslmode дозволяє підключатися до бази з різних адміністративних панелей для роботи з базою даних безпосередньо. Готовий контекст бази може працювати і взаємодіяти з даними за допомогою класів. Для зміни структури бази даних використовуються міграції. Вони створюються після зміни класів, які описують таблиці і їх атрибути. Міграції складаються з назви і версії Entity Framework Core, та за допомогою migrationBuilder послідовно описують всі зміни, які необхідно застосувати до бази.

Нижче представлені основні функції та процедури програмного продукту, із застосуванням яких реалізовано систему швидкої допомоги.

Реалізація одноразового пароля для підтвердження пошти. Даний метод обробляє створення і відправлення одноразових паролів. Він є асинхронним і на виході має строковий тип даних, який містить згенерований одноразовий пароль. Параметрами методу є пошта і логічне значення того, чи є це реєстрація або авторизація. Далі відбувається пошук користувача за поштою. Якщо логічне значення дорівнює *true*, але користувач не знайдений, метод видає «виняток». В іншому випадку, якщо змінна дорівнює *false* і користувач знайдений в системі, також буде видано «виняток». Далі конструктор *MimeMessage()* створює повідомлення, яке і буде відправлено як одноразовий пароль. За допомогою *SmtpClient()* створюються конфігурації для конекту. В результаті метод поверне згенерований пароль.

Логіка створення одноразового пароля винесена в окремий метод. Вказано масив всіх цифр, і за допомогою цього масиву створюється 6-значний одноразовий пароль. Сортування відбувається в циклі, а на виході створюється рядок з випадково відсортованих цифр.

```
public string GenerateOtp()
      {
       var chars = "1234567890";
       var stringChars = new char[6];
       var random = new Random();
       for (int i = 0; i < stringChars.Length; i++)
       { stringChars[i] = chars[random.Next(chars.Length)];
       }
       var str = new String(stringChars);
       return str;
      }
```

Створення та робота хабів. Хаби – це умовні кімнати, які містять з'єднання деякої кількості користувачів для обміну між ними даними в режимі реального часу. Хаби створюються на сервері, приймаючи всі повідомлення від одного користувача цього хаба і транслюючи це повідомлення іншим користувачам хаба.

```
public class HubContextGps : HubContext
   {// checking status
      public static bool ReadyForConnect { get; set; }
      public static bool IsConnection { get; set; }
      public HubContextGps()
      { HubConnection = new HubConnectionBuilder()
            .WithUrl(Url + "gps")
            .Build();
      }
        public async Task EnterToGroup(int callId)
      { await HubConnection.InvokeAsync("Enter", callId);
      }
                public async Task SendToServer(int callId, double latitude, double  longitude)
      { await HubConnection.InvokeAsync("SendGPS", callId, latitude, longitude);
      }}
```

Контекст хаба: змінні *ReadyForConnect* і *IsConnection* вказують стан підключення до хабу. Конструктор *HubContextGps()* за допомогою *HubConnectionBuilder()* створює фактично підключення до хабу. Метод *EnterToGroup(int callId)* додає користувача до групи в хабі. Назва хаба визначається ідентифікатором виклику. На сервері хаб створює групу з такою назвою. Група визначається ідентифікатором виклику, а дані про широту та довготу транслюються учасникам групи хаба.

Транслює дані користувач ролі «водій». Після прийняття виклику, в хабі знаходяться два учасники: користувач, який викликав швидку і водій карети, який виконує цей виклик. Дані про місцезнаходження карети водія транслюються потерпілому в режимі реального часу. Таким чином реалізується можливість відстеження карети швидкої допомоги в мобільному додатку.

На сервері використовується бібліотека *SignalR*, яка забезпечує роботу з хабами, групами і підключеннями.

```
public class GpsHub : Hub
    { public async Task Enter(int idCall)
      {
       await Groups.AddToGroupAsync(Context.ConnectionId, $"group{idCall}");
      }
      public async Task SendGPS(int idCall, double latitude, double longitude)
      {
       await Clients.OthersInGroup($"group{idCall}").SendAsync("ReceivingGPS", latitude, longitude);
       await Clients.All.SendAsync("ReceivingGPS", latitude, longitude);
     }}
```

Новий хаб створюється за допомогою класу *Hub*. Асинхронні методи *Enter* та *SendGPS* забезпечують роботу хаба з клієнтами мобільного додатка. *Groups.AddToGroupAsync* додає нові підключення в хаб, а *SendAsync* транслює повідомлення іншим підключеним клієнтам. *SignalR* сам надає кращий механізм для взаємодії для конкретних випадків. Найбільш оптимальним є *WebSockets*. У деяких випадках, коли *WebSockets* не підтримується, використовується *Server-Side Events* або *Long Polling*.

Реалізація програмного продукту

Методологія розробки. Було поставлено важливе завдання створення максимально точного, зручного та завершеного програмного забезпечення. Тому обрана методологія розробки передбачала побудову прототипної моделі системи, що дозволило в короткі терміни реалізувати мобільний додаток для пацієнтів та працівників швидкої допомоги. Оскільки програмний продукт мав проходити численні стадії перегляду та доопрацювання, обрана методологія виявилась вкрай єфективною та дозволила оперувати найзатребуванішими інструментами розробки.

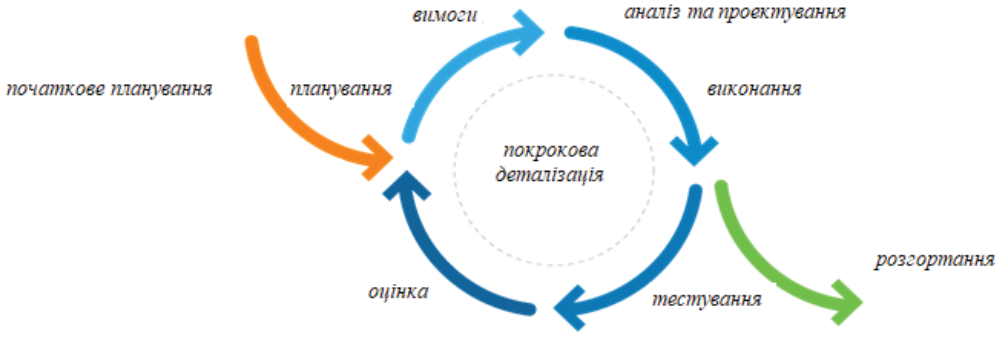


Рис. 2 – Прототипна модель розробки додатку

Функціанальні можливості та опис програмного продукту. Користувач може зареєструватися і увійти в мобільний додаток, щоб мати можливість викликати карету швидкої допомоги. Після виклику карети, користувач може відстежувати її місцезнаходження в режимі реального часу. На мапі показаний оптимальний шлях від місця виклику до поточного знаходження карети, а також відображається зміна положення карети під час її руху. Для користування мобільним додатком необхідно мати доступ до мережі і включений GPS. При першому вході відкривається форма входу в додаток. На рисунку 3 наведено сторінку авторизації користувача в додатку.

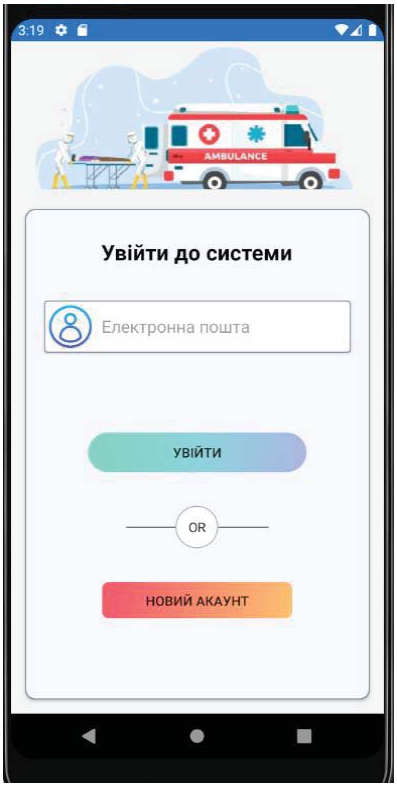


Рис. 3 – Сторінка авторизації

Для реєстрації нового користувача передбачено кнопку «Новий акаунт» після натискання якої відкриється нова сторінка, де створюється новий аккаунт. Користувач повинен ввести свої дані і натиснути на кнопку «Підтвердити». Після цього на пошту, яку було вказано, надсилається одноразовий пароль для підтвердження. Після введення одноразового пароля на сторінці реєстрації, система зареєструє користувача, вносить його дані. Далі відбувається перехід на головну сторінку програми. На рисунку 4 наведено меню користувача-пацієнта мобільного додатку.

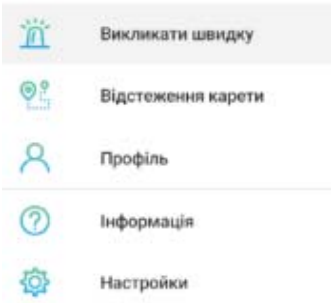


Рис. 4 – Меню користувача

На головній сторінці додатку можна відразу викликати швидку допомогу. Для цього є три опції: вибрати причину виклику і натиснути кнопку «Викликати швидку»; самому описати причину виклику і натиснути кнопку «Викликати швидку»; не вказувати нічого і натиснути кнопку «Викликати швидку». Далі користувач має можливість відстежувати свою карету, яка прямує до нього в даний момент. Для цього необхідно перейти на сторінку «Відстеження карети». На рисунку 5 наведено вікно додатка, де відображається шлях, місцезнаходження пацієнта та місцезнаходження карети в реальному часі. Маршрут також відображається у додатку водія швидкої, завдяки чому він витрачає набагато менше часу для того, щоб дістатися пункту призначення. Таким чином вирішується дуже важлива проблема економії часу, що може бути дуже важливим фактором для порятунку людини. В свою чергу людина також витрачає мінімальну

кількість часу для виклику швидкої допомоги через мобільний додаток. Користувач, який знаходиться в стані, коли він навіть не взмозі повідомити про свою проблему, одним кліком може здійснити виклик.

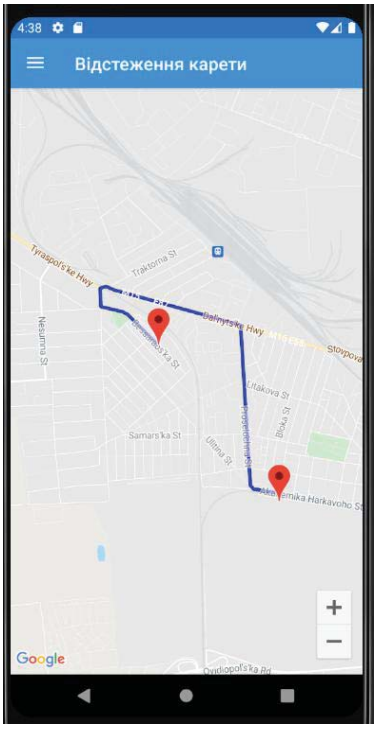


Рис. 5 – Вікно відстеження місця знаходження карети швидкої допомоги

Також мобільний додаток має сторінку особистого профілю пацієнта. У ньому користувач може переглядати свої дані, введені підчас реєстрації. За потребою користувач може змінювати інформацію натиснувши кнопку «Оновити дані». Після цього запит, в тілі якого містяться нові значення полів, відправляється на сервер. Після обробки даних і збереження їх в базі даних, інформацію буде оновлено. Підчас кожного входу в додаток авторизація зберігається. Натискання кнопки «Вийти з системи» призводить до зміни користувача.

В разі використання додатку користувачем – водієм карети швидкої допомоги, реалізовано наступний алгоритм дій. При виклику швидкої допомоги пацієнтом водієві карети приходить повідомлення про те, що в систему надійшов новий виклик. Після узгодження зі станцією водій приймає виклик у додатку, де одразу відобразиться точка на мапі – місце знаходження пацієнта та буде прокладено маршрут за цією адресою. В момент прибуття за місцем призначення користувачеві приходить повідомлення про те, що карета швидкої допомоги прибула за адресою.

Розроблений додаток протестовано на різних версіях Android. Доведена стабільна работа додатку на мобільних пристроях різних виробників.

Висновки

Важливою частиною даної роботи є реалізація мобільного додатку і серверу. Завдяки обраній методології розробки, а саме побудові прототипної моделі, мобільний додаток створено в короткі терміни, його роботу досліджено, швидко виявлені недоліки окремих модулів, внесено відповідні зміни до програмного коду. Розробка і врахування всіх нюансів роботи програми в режимі реального часу вимагає заздалегідь грамотного проектування всієї системи, оскільки додаток повинен моментально реагувати на зміни в координатах карети швидкої допомоги, а сервер додатку повинен мати високу стійкість до будь-яких навантажень. Ще однією важливою особливістю є реалізація асинхронних методів для роботи програми в декількох потоках. Поставали питання, як реалізувати взаємодію пацієнта і водія карети, обмін даними між ними. Рішенням стало створення умовних кімнат – хабів на стороні сервера, в яких знаходяться пацієнт і водій. Хаби існують окремо і не взаємодіють один з одним, тому робота потоків даних на сервері проходить асинхронно, не зупиняючи інші потоки. Для відправки на сервер місця розташування пацієнта автоматично використовується система глобального позиціонування (GPS) в мобільному телефоні. Дані місця розташування карети можуть передаватися або таким же шляхом за допомогою телефону водія, або через навігатор GPS, розташований в кареті швидкої допомоги

Список використаних джерел

[1]. Інформаційна технологія [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/suchasniinftechnology/home/1-1-so-take-informacijna-tehnologia. Дата звернення: 15.02.2021

[2]. GPS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/GPS. Дата звернення: 22.02.2021

[3]. Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019. Дата звернення: 05.02.2021
[4]. Rider [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/rider/. Дата звернення: 18.02.2021
[5]. SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sql/sqlserver/1.1.php. Дата звернення: 17.03.2021
[6]. Сервер баз даних SQLServer [Електронний ресурс] – Режим доступу до ресурсу: https://hsbi.hse.ru/articles/server-baz-dannykh-sql-server/. Дата звернення: 15.03.2021
[7]. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://static-course-assets.s3.amazonaws.com/ITE50UK/course/module6/ 6.2.1.7/6.2.1.7.html Дата звернення: 01.04.2021
[8]. ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/1.1.php. Дата звернення: 12.02.2021
[9]. Mark J. P. C# 8.0 and .NET Core 3.0. – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition. Packt Publishing, 2020. 818 p.
[10]. Jon Skeet. C# in Depth: Fourth Edition. Manning Publications, 2019. 528p.
[11]. Andrew Troelsen and Philip Japikse. Pro C# 7: With .NET and .NET Core. Apress, 2017. 1437 p.
[12]. Joseph Albahari and Ben Albahari. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2017. 1088 p.
[13]. Adam Freeman. Pro ASP.NET Core 3 (Develop Cloud-Ready Web Applications Using MVC 3, Blazor, and Razor Pages). Apress, 2020. 1109 p. https://doi.org/10.1007/978-1-4842-5440-0
[14]. Andrew Lock. ASP.NET Core in Action, Second Edition. Manning Publications, 2021. 832 p.
[15]. Ed Snider. Mastering Xamarin.Forms: App architecture techniques for building multi-platform, native mobile apps with Xamarin.Forms 4, 3rd Edition. Packt Publishing, 2019. 200 p.
[16]. Dan Hermes, Nima Mazloumi. Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals. Packt Publishing, 2019. 454p. https://doi.org/10.1007/978-1-4842-4030-4
[17]. Neil Middleton, Richard Schneeman. Heroku: Up and Running: Effortless Application Deployment and Scaling. O'Reilly Media, 2013. 100 p.

References

[1]. Information Technology [Electronic resource] – Available at: https://sites.google.com/site/suchasniinftechnology/home/1-1-so-take-informacijna-tehnologia. Accessed: 15.02.2021
[2]. GPS [Electronic resource] – Available at: https://uk.wikipedia.org/wiki/GPS. Accessed:22.02.2021
[3]. Visual Studio [Electronic resource] – Available at: https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019. Accessed:05.02.2021
[4]. Rider [Electronic resource] – Available at: https://www.jetbrains.com/ru-ru/rider/. Accessed:18.02.2021
[5]. SQL Server [Electronic resource] – Available at:https://metanit.com/sql/sqlserver/1.1.php. Accessed: 17.03.2021
[6]. Database server SQLServer [Electronic resource] – Available at: https://hsbi.hse.ru/articles/server-baz-dannykh-sql-server/. Accessed:15.03.2021
[7]. Client-server architecture [Electronic resource] – Available at: https://static-course-assets.s3.amazonaws.com/ITE50UK/course/module6/ 6.2.1.7/6.2.1.7.html Accessed:01.04.2021
[8]. ASP.NET Core [Electronic resource] – Available at: https://metanit.com/sharp/aspnet5/1.1.php. Accessed: 12.02.2021
[9]. Mark J. P. C# 8.0 and .NET Core 3.0. – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition. Packt Publishing, 2020. 818 p.
[10]. Jon Skeet. C# in Depth: Fourth Edition. Manning Publications, 2019. 528p.
[11]. Andrew Troelsen and Philip Japikse. Pro C# 7: With .NET and .NET Core. Apress, 2017. 1437 p.
[12]. Joseph Albahari and Ben Albahari. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2017. 1088 p.
[13]. Adam Freeman. Pro ASP.NET Core 3 (Develop Cloud-Ready Web Applications Using MVC 3, Blazor, and Razor Pages). Apress, 2020. 1109 p. https://doi.org/10.1007/978-1-4842-5440-0
[14]. Andrew Lock. ASP.NET Core in Action, Second Edition. Manning Publications, 2021. 832 p.
[15]. Ed Snider. Mastering Xamarin.Forms: App architecture techniques for building multi-platform, native mobile apps with Xamarin.Forms 4, 3rd Edition. Packt Publishing, 2019. 200 p.
[16]. Dan Hermes, Nima Mazloumi. Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals. Packt Publishing, 2019. 454p. https://doi.org/10.1007/978-1-4842-4030-4
[17]. Neil Middleton, Richard Schneeman. Heroku: Up and Running: Effortless Application Deployment and Scaling. O'Reilly Media, 2013. 100 p.

Отримана в редакції 19.11.2021. Прийнята до друку 30.11.2021. Received 19 November 2021. Approved Approved 30 November 2021. Available in Internet 04 December 2021.