МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Веб-сервіс з використанням алгоритму оповіщення клієнтів на основі фреймворку React Native»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Проценко О.Б.**

Студента групи ІН.мз – 92с — **Сокура І.А.**

СУМИ 2021

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп ютерних наук*
Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
“_____” _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Сокури Інни Андріївни
(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) *Веб-сервіс з використанням алгоритму оповіщення клієнтів на основі фреймворку React Native*
затверджую наказом по інституту від “____” _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____
3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Огляд методів розробки застосунку; 2)Постановка задачі; 3) Огляд та вибір програмних засобів; 4) Проектування застосунку; 5) Розробка алгоритму оповіщення; 6) Програмна реалізація.*
_____
_____
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.			
2.			
3.			
4.			
5.			

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 63 стор., 21 рис., 1 додаток, 16 джерел.

Об’єкт дослідження – інформаційна система розроблення мобільного застосунку для роботи зі сповіщеннями.

Мета роботи— проаналізувати та порівняти особливості функціонування об`єкту дослідження. Створення мобільного застосунку - для інформування користувачів за допомогою push-сповіщень на основі фреймворку React Native.

Результати — описаний процес проектування алгоритму роботи оповіщення клієнтів. Також проведено порівняння існуючих застосунків зі схожим функціоналом. Розроблена API за допомогою фреймворку Laravel. Результатом роботи став мобільний застосунок для інформування користувачів.

REACT NATIVE, LARAVEL, PUSH-NOTIFICATION, NOTIFICATION, RESTFUL API, MYSQL, MVC, МОБІЛЬНИЙ ЗАСТОСУНОК

ЗМІСТ

ВСТУП

Мобільні сповіщення є важливою функцією мобільних обчислювальних служб, і вони широко застосовуються у мобільних застосунках. Це покращило якість спілкування між людьми, а також є ефективним способом доставки рекламної інформації потенційним клієнтам. Крім масової реклами, останнім часом популярність придбала персоналізована передача інформації. Персоналізована передача це – доставка певної інформації конкретним споживачам.

Довгий час основою такого роду систем доставки інформації були короткі повідомлення (SMS) і їх просунутий варіант MMS. Саме SMS використовувалися (використовуються зараз) як транспортний механізм в інформаційних системах для мобільних користувачів. У теперішній же час, наприклад, цілком очевидно зміщення інтересів мобільним користувачів від SMS до так званого Instant Messaging.

Управління повідомленнями може бути більш зручно для користувачів, ніж операції з SMS. Вартість доставки інформації, тобто SMS є основним джерелом доходу телекомунікаційних компаній, і ціни на цю послугу не знижуватимуться. Все це призводить до того, що відмова від передачі SMS та MMS повідомлень та заміна їх іншими способами обміну інформацією в режимі реального часу між мобільними користувачами становляться актуальними.

В результаті проведеної роботи буде отримано повністю працездатну програму, яку можна буде повноцінно використовувати в якості застосунку для інформування користувачів за допомогою push-сповіщень.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Основні характеристики push-сповіщень

Push-сповіщення – це повідомлення, що з'являються на екрані мобільного пристрою, де є область сповіщення й інформують про важливі оновлення та події[1]. Практично кожна операційна система підтримує push-сповіщення. Вони є одним з інтерактивних елементів, що надає можливість переглянути докладнішу інформацію натиснувши на спливаюче повідомленням та є засобом представлення інформації для користувача. Вони не залежать від відсутності чи наявності Інтернету, і наприклад, відображаються в залежності від геолокації, або просто інформують користувача телефону про точний час та дату. Однак в даній роботі розглядається push-сповіщення в роботі з сервером, що буде забезпечувати інформування клієнтів.

У системах push-сповіщення доставлення повідомлень здійснюється за допомогою мережі Інтернет, використовуючи модель клієнт-сервер. Користувачі мобільних пристроїв за допомогою встановленої програми підписуються на розсилку повідомлень, а сервер публікації, коли настає певна подія, здійснює відправлення повідомлень клієнтам, що підписалися на розсилку. Таким чином, адресат повідомлення це завжди застосунок. Push-сповіщення дозволяють серверам повідомити користувачів про певну подію, навіть якщо застосунок неактивний в даний момент. Достатньо того, щоб застосунок був встановлений.

Відправлення повідомлень на сервері, здійснюється коли настає деяка подія. За допомогою застосунку відстежується подія, це може бути отримання повідомлення від адміністратора, або поблизу знаходиться деяка кількісті клієнтських пристроїв (це може бути Wi-Fi або Bluetooth моніторинг).

Спочатку технологія push мала відношення не до мобільних застосунків, а до мережі PointCast, що займалася розсилкою новин та даних фондового

ринку. Дану систему використовують у США, для розсилки передплатникам даних про процеси. Пізніше Netscape і Microsoft включили цю технологію в свої браузери, але через низьку швидкість підключення користувачів в той час вона була витіснена pull-технологією RSS. Та лише потім термін став широковідомим після впровадження мобільного сервісу Google Cloud Messaging (GCM) створений компанією Google в операційній системі Android та Apple Push Notification Service створений компанією Apple в iOS 3. На рисунку 1.1 розглянемо елементарну схему роботи push-сповіщення.

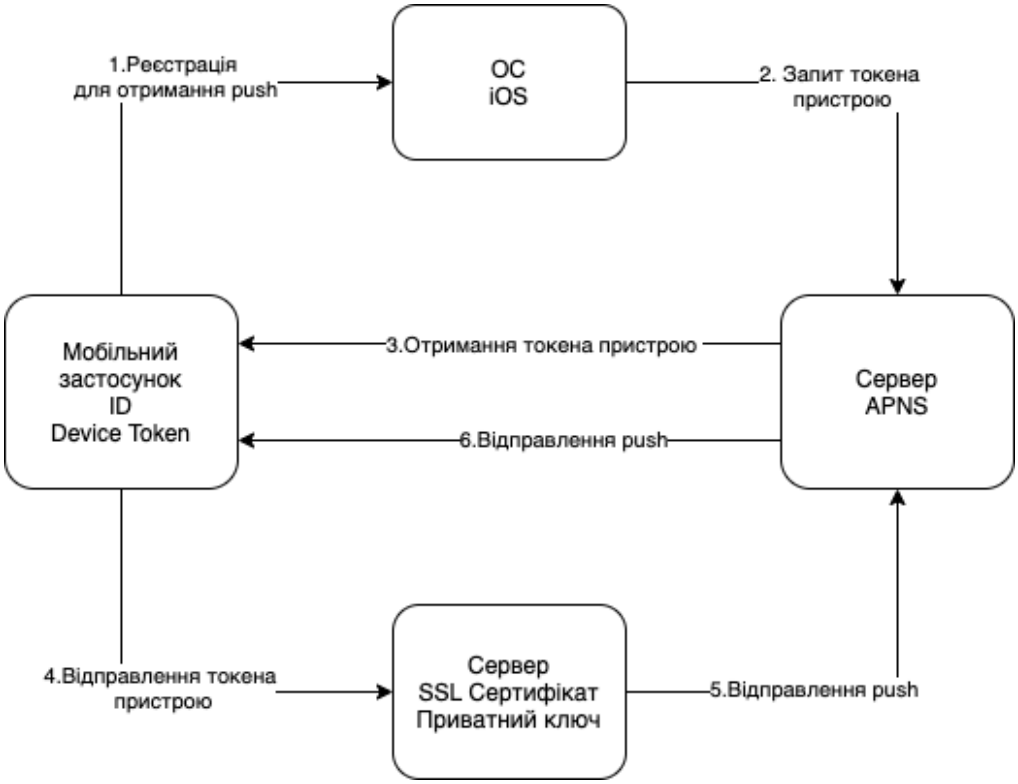

Рисунок 1.1 – Схема роботи Push Notification на прикладі сервісу APNS

Для того, щоб повідомлення відобразилося на екрані пристрою, сам застосунок не обов'язково має бути ввімкненим – саме для реалізації цієї переваги посередником тут виступає OS. До речі, такий підхід дозволяє економити заряд батареї смартфона, і трафік. Схема роботи сервісу:

1. Для отримання push-сповіщень OS повинна зареєструвати мобільний застосунок;

2. OS запитує у APNS ідентифікатор пристрою (токен);

3. Застосунок отримує токен від сервера APNS;

4. Застосунок відправляє токен назад на сервер, щоб далі сервер користувався ним для відправки push-сповіщень;

5. При настанні певної події, сервер, використовуючи маркери, відправляє push-сповіщень через APNS;

6. APNS робить розсилку повідомлень у застосунок.

Існують нюанси у розсилках push-сповіщень для різних операційних систем таких як, Android, iOS, Windows Phone. Припустимо, якщо користувач видалив застосунок, то всі сервіси повідомляють про те, на які пристрої не слід більше посилати повідомлення. Здійснюється цей процес за допомогою повідомлення серверу токенів цих пристроїв. Але якщо у GCM відправка ідентифікаторів відбувається відразу, то у APNS є спеціальний сервер зворотного зв'язку, з якого список таких токенів забирається раз на добу. Для роботи з цими відмінностями і потрібні проміжні сервіси.

У разі розробки мобільного застосунку [7] за допомогою будь-якого кросплатформного рішення, наприклад Appcelerator, такий проміжний сервіс, як правило, інтегрований в нього. В тому ж Appcelerator це Appcelerator Cloud Services (ACS), що представляє собою додатковий сервіс каналів повідомлень. Такий канал об'єднує кілька пристроїв, будучи своєрідним ідентифікатором, що складається з цифр і букв. ACS дає можливість відправляти пуші і по токену пристроїв. Отже, даний проміжний сервіс бере на себе функцію оновлення інформації про пристрої і взаємодіє з APNS і GCM.

Схема такої взаємодії виглядає розглянута на рисунку 1.2:

1. При розробці мобільного застосунку до нього впроваджується ключ, який видає ACS;

2. Будь-яке повідомлення є словником формату JSON, що складається з токена девайса, деякої службової інформації та дані, які відправляються на телефон.

3. Сервер, користуючись ключем:

3.1. отримує список каналів і пристроїв, підписаних на канали;

3.2. підписує (і відписує) пристрої на певні канал;

3.3. відправляє push-сповіщення на всі пристрої або тільки по певним токенам або каналам пристроїв.

4. Пристрої, в залежності від їх операційної системи, отримують push-сповіщення від GCM або APNS.

Рисунок 1.2 – Схема взаємодії серверу з мобільним застосунком

1.2 Сервіси push-сповіщень

Існує досить велика кількість інформаційних систем, які так чи інакше намагаються надати інструмент одним користувачам розсилати інформацію іншим користувачам за допомогою push-сповіщень. Для доставлення push-сповіщень від сервера-відправника до клієнта-одержувача в цілях економії ресурсів клієнтських пристроїв всі існуючі системи використовує проміжну

ланку, як було сказано раніше. Цією ланкою є служба повідомлень (Push Service Provider). Ось декілька прикладів основних служб:

- Google Cloud Messaging для мобільних пристроїв під Android;
- Apple Push Notification Service для мобільних пристроїв під iOS;
- Microsoft Push Notification Service для мобільних пристроїв під Windows.

Дані служби підтримують економічне з'єднання з мобільними пристроями і здійснюють доставлення push-сповіщень. Операційна система мобільного пристрою перенаправляє повідомлення до відповідного застосунку. У застосунках повідомлення з'являються у верхній частині екрана в панелі повідомлень.

Інформаційні системи [2] в даній галузі ставлять перед собою такі основні завдання:

- надати користувачу інструмент для створення розсилки;
- надати можливість здійснювати будь-який з видів розсилок;
- надати користувачу більш прозорий спосіб підписки на розсилку.

Для вирішення завдань зі списку, користувачам надається доступ до віртуального особистого кабінету, де у користувача є можливість управління розсилками. В особистому кабінеті є можливість створення розсилки певної тематики, перегляд статистики відкритих push-сповіщень, відправлення повідомлення і т.д.

Для роботи з розсилками використовуються так звані, мітки, які присвоюються кожному одержувачу. Це універсальні характеристики, або довільні теги, універсальні характеристики, такі як версія операційної системи мобільного пристрою. Для персоналізованої розсилки можуть використовуватися унікальні ідентифікатори, які генеруються мобільним застосунком, або ж персональні параметри, що налаштовуються користувачем – наприклад, день народження власника телефону.

Як правило, також є можливість вказати заголовок повідомлення і посилання для переходу при натисканні одержувачем на повідомлення, коли воно з'являється в області сповіщень. Може вказуватися і зображення, що з'являється разом з повідомленням, та звуковий сигнал. Для того, щоб одержувач міг підписатися на розсилку, необхідно, по-перше, створити програму, яка б була передплатником в технічному сенсі, і по-друге доставити застосунок до кінцевого споживача. Для цього застосовуються найрізноманітніші рішення.

У таблиці 1.1 зображено порівняльну характеристику деяких відомих інформаційних систем, які використовуються для проведення інформаційних розсилок на основі push-сповіщень.

Таблиця 1.1 – Порівняльна характеристика існуючих систем

Назва сервісу	ОС	Вміст повідомлень
Amazon Simple Notification Service	iOS, Android, Fire OS, Windows	Є можливість вказання теми, тексту, звуку, вставлення посилання для переходу та зображення
Windows Azure Notification Hubs	Windows 8, Windows Phone, iOS, Android	
Urban Airship Push-notifications	Windows 8, iOS, Windows Phone, Android	
Jeapie	iOS, Android, Windows та інші	Відмінною особливістю є візуальний конструктор повідомлень, який дозволяє акцентувати увагу на візуальній складовій повідомлення
Push2Press	iOS, Android	Є можливість вказання теми,

OpenPush	iPhone, Android, Windows Phone	тексту, звуку, вставлення посилання для переходу та зображення

У підсумку перераховані завдання сервісу. Для створення розсилки на основі push-сповіщень необхідно створити одну-дві облікові записи, після чого контент розсилки вказати. Для роботи сервісу з мобільним застосунком є готові рішення: бібліотеки, що вбудовуються у вихідний код програми, та готовий застосунок, налаштований на взаємодію з конкретним сервісом.

Супровід розсилки: будь то платний веб-сервіс або сервіс з відкритим вихідним кодом, все зберігання повідомлень, гарантію їх доставки і захист інформації він бере на себе.

1.3 Порівняння SMS-розсилки з push-сповіщенням

Попередником, і в деякому сенсі конкурентом інформаційних систем на базі push-сповіщень є системи, засновані на SMS-розсилках. Крім переваг веб-сервісу push-сповіщень перед іншими аналогічними інформаційними системами, проведено порівняння з сервісом, що здійснює SMS розсилку. У порівнянні розглядатимуться відмінності технологій push і SMS самих по собі. Перераховані переваги однієї системи є недоліками іншої системи.

Переваги SMS-розсилок:

1. Не треба ставити спеціальний застосунок.
2. Немає обмежень на ОС пристрою – підтримуються абсолютно всі моделі мобільних телефонів.
3. Для того, щоб підписати власника пристрою до розсилки, потрібен тільки його номер телефону.
4. Можна відправити повідомлення «наосліп» за випадковим номером – іноді це буває корисним.
5. Немає залежності від інтернету.

6. Простота організації – крім підписки на сервісі розсилок взагалі ні про що більше не треба думати.

7. Потрібно платити за кожне SMS-повідомлення.

Переваги розсилок на основі push-сповіщень:

3. Простий доступ до персональних даних користувача.

4. Зрозумілий спосіб підписатися/відписатися від розсилок.

5. Можливість організувати зворотний зв'язок за допомогою застосунку.

6. Потенційно більш широкі можливості по відображенню повідомлень – наприклад, іконка, попередньо встановлена публікатором.

7. Вимога до розвиненої ОС мобільного телефону дозволяє гарантувати наявність таких даних, як, наприклад, GPS-координати. Звідси всілякі переваги на зразок можливості проведення гео-позиціонування і відправки повідомлень в залежності від місцезнаходження користувача.

Як видно з цих переліків, кожна технологія все ж має право на життя, володіючи своїми унікальними можливостями. Вибір повинен проводитися в залежності від конкретної галузі, в якій необхідно використовувати розсилку повідомлень.

1.4 Аналіз існуючих рішень

У процесі дослідження існуючих рішень буди виявлені два мобільних застосунка, та телеграм бота:

1. “Блэкаут”. Планові відключення.

2. “1580”. Гаряча лінія міста Львова.

3. “CityBot Назар”

Кожен з програмних продуктів має свій унікальний функціонал та особливість роботи. Розглянемо кожен з цих продуктів.

Мобільний застосунок “Блэкаут” [10] (рис. 1.3) для ОС Android дозволяє отримувати інформацію про заплановані відключеннях електроенергії в

будинках Одеси. У застосунку можна зберегти будь-яку кількість адрес, що цікавить користувача. І отримувати повідомлення про майбутні ремонтні роботи заздалегідь. Звичайно ж, як і на сайті blackout.today, можна переглянути повний список запланованих робіт на найближчі дні. І здійснити пошук по адресам. Інформація на сайті blackout.today в автоматичному режимі збирається з сайту ПАТ «Одессаоблэнерго». Таблиці містять тільки інформацію про заплановані ПАТ «Одессаоблэнерго» відключеннях, але не про аварійні. До наданої інформації слід ставитися як до довідкової, а не 100% надійної. Сайт та застосунок працюють в режимі бета-тестування. Є відомі помилки, пов'язані з машинної обробкою інформації наданої на сайті енергокомпанії. Тому надійна обробка даних – це ще великий обсяг роботи.

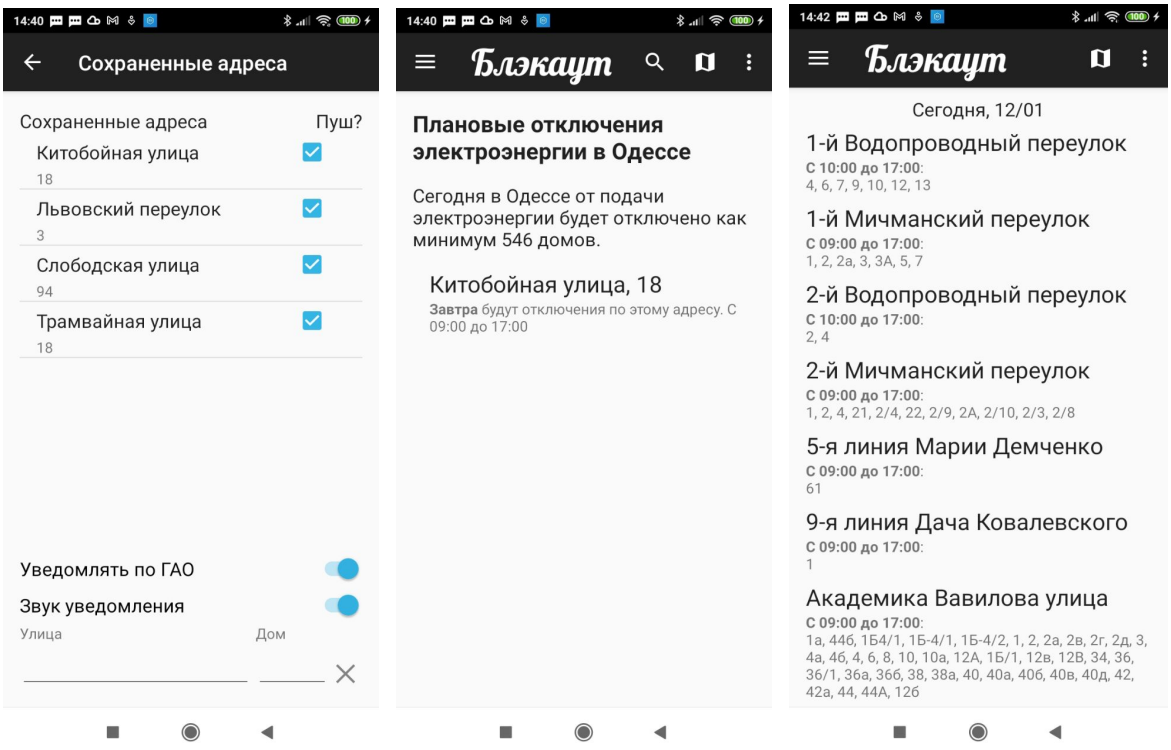

Рисунок 1.3 – Мобільний застосунок “Блэкаут”

Застосунок “1580” [11] (рис. 1.4) створено для того щоб, було зручно відправляти ініціативи та пропозиції до Львівської міської ради. Гаряча лінія

міста Львів –це міська телефонна служба, де кожен мешканець міста Львів може отримати консультацію, що йому потрібна. Ця служба контролює, як вирішуються конкретні проблеми.

За допомогою даного мобільного застосунку, після обов'язкової реєстрації, буде можливість користуватися наступними функціями:

– отримувати повідомлення або сповіщення від міських комунальних служб, що стосуються будинку, який було вибрано в самому початку. В даному застосунку можна вибрати лише один будинок за яким користувач буде отримувати сповіщення;

– можливість надіслання звернень та додавання фотографій проблемної ділянки;

– слідкувати за розглядом та опрацюванням звернення;

Команда 1580 створена задля покращення благоустрою міста.

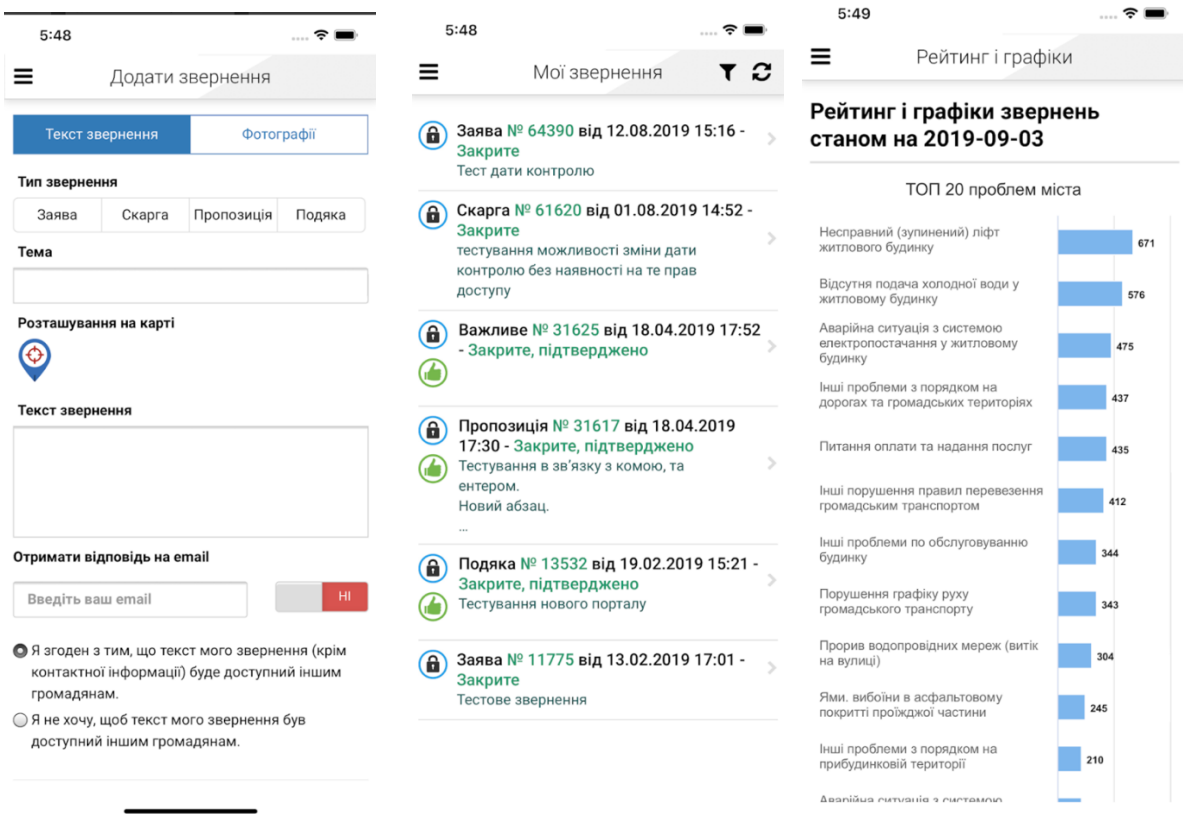

Рисунок 1.4 – Мобільний застосунок "1580"

У Тернополі працює сіті-бот на ім'я Назар [12], який зможе відповісти на всі їхні запитання про комунальні послуги у місті Тернопіль. Назар буде сповіщати містян про всі планові і аварійні відключення водопостачання, електроенергії, газу або опалення на вашу адресу. Знайти сіті-бота Назара можна через Viber або Telegram. Далі ввести адресу, за якою користувач хоче отримувати сповіщення, у даному застосунку можна вибрати лише одну адресу за якою приходитимуть сповіщення. А в інформаційному розділі "Довідка" можна знайти телефони всіх комунальних, державних підприємств і соціальних служб міста. Назар бот кожного місяця повідомляє користувачів про те що через нього можна передати показники лічильників та до якої дати це потрібно зробити. Та інформує про будь-які інші технічні несправності.

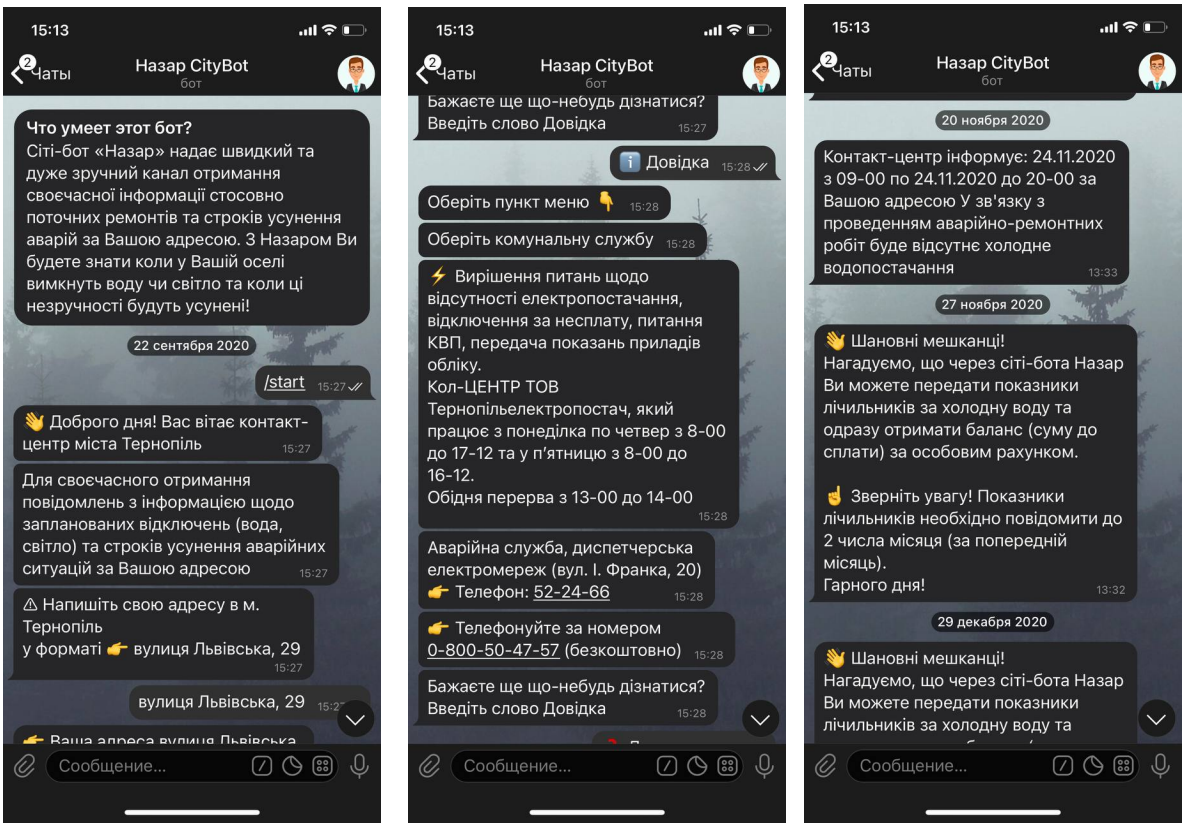

Рисунок 1.5 – CityBot Назар

1.5 Постановка задачі

Виконати проектування та програмну реалізацію мобільного застосунку з такими функціями:

1. Авторизація користувачів у системі.
2. Можливість перегляду подій та новин.
3. Особистий кабінет користувача з можливість внести особисті дані.

Виконати проектування та програмну реалізацію веб-сервісу з такими функціями:

1. Авторизація адміністратора у системі.
2. Можливість додавання редагування та видалення подій та новин.
3. Виконання персоналізованої розсилки сповіщень користувачам.

Застосунок призначений для сповіщення користувачів та нагадування про планові відключення води та світла. Передбачити можливість масштабування застосунку.

2 ВИБІР МЕТОДІВ РІШЕННЯ

2.1 Мови програмування

Для реалізації мобільного застосунку були обрані такі технології, як JavaScript [3], Bootstrap [4], PHP [5], MySQL. Розглянемо кожну з них більш детально.

JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування. Є реалізацією стандату ECMAScript. JavaScript [8] зазвичай використовується, як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить у браузерах, як мова сценаріїв для надання інтерактивності веб-сторінкам.

Дана мова була обрана за простоту використання і доопрацювання. Ця мова дозволить керувати каскадними таблицями CSS і розміткою HTML, роблячи сторінки більш «живими», що дозволить користувачу комфортніше працювати з застосунком.

Bootstrap — вільний набір інструментів для створення сайтів і веб застосунків. Включає в себе HTML і CSS шаблони оформлення для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсів, включаючи JavaScript розширення [4]. Це доповнення до каскадних таблиць CSS дозволить підтримати весь проект в одному стилі, організовуючи при цьому зручний інтерфейс.

PHP – скриптова мова загального призначення, інтенсивно застосовується для розробки веб-застосунків. В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів [5].

Дана мова була обрана за простоту, легкість вивчення, веб-спрямованості, швидкості виконання скриптів, захищеності, об'єктно-орієнтованості.

MySQL — вільна реляційна система управління базами даних. Обрана за свою гнучкість, простоту використання і легкість в масштабуванні проекту при необхідності.

2.2 Фреймворк Laravel

Мова програмування PHP в даний час використовується для розробки веб-сайтів. За допомогою мови PHP написано приблизно 80% сайтів. Головна особливість – це те, що виконується потрібний файл. Тобто, якщо розробник допустить помилку в якомусь файлі, то сайт продовжить працювати, поки до файлу що містить помилку не буде виконаний запит.

PHP на відміну від наведених інших мов має великий вибір фреймворків та систем керування контентом, що дає змогу розробникам вибирати різні рішення для виконання задач.

Для розробки веб-сервісу було вирішено використовувати фреймворк Laravel [13]. Даний фреймворк використовує архітектурний шаблон MVC, який передбачає поділ застосунку на три пов’язані частини: модель, контролер та відображення.

В моделях описується структура таблиць, що знаходяться в базі даних, методи, що дають можливість отримати дані з бази даних та встановити зв’язки між різними таблицями. При виконанні запиту до бази даних ми отримуємо об’єкт або масив об’єктів. За допомогою ORM Eloqument відбувається обмін даним через об’єкти. ORM Eloqument зв’язує програмний код та базу даних та здійснює обмін даними у зручному для нас вигляді. За логіку роботи застосунку відповідає контролер. Там знаходяться виклики методів та здійснюється обмін даними між користувачем та застосунком. Користувачу із контролера повертається масив даних, або частина HTML верстки із новими даними. В даному фреймворку використовується шаблон Blade, для того щоб, показати верстку та дані, що динамічно підставляються у необхідні блоки, та даний шаблонізатор має деякі елементи синтаксису, подібного до php.

Laravel має модульну систему та надає дозвіл завантажувати та використовувати різні бібліотеки. Завантаження бібліотек або фреймворку здійснюється за допомогою менеджера залежностей Composer, в якому для завантаження необхідного компонента достатньо написати його назву та версію і виконати команду оновлення.

2.3 Система управління базами даних MySQL

Система управління реляційними базами даних MySQL [15] має відкритий програмний код та використовує мову SQL(для виконання запитів Перевага даної системи – вона працює майже на всіх платформах такі, як Linux, Unix та Windows. Використовується MySQL найчастіше у веб-застосунках. MySQL базується на моделі клієнт-сервер. Ядром цієї системи є сервер.

Дана система була розроблена для швидкої обробки великих баз даних, тому вона і набула значної популярності у веб-застосунках, оскільки вони забезпечують їх належну швидкодію. MySQL дозволяє створювати два типи таблиць InnoDB та MarinaDB. Основна відмінність між цими типами таблиць полягає в тому, що InnoDB підтримує зовнішні ключі між таблицями, а MarinaDB – ні. Відсутність зв’язків спричиняє пришвидшення виконання запитів до бази даних, оскільки в такому випадку відсутні перевірки можливих зв’язків та обмежень перед записом та оновленням даних. Це призводить до значного пришвидшення роботи з базою даних, але недоліком є те, що в такому випадку, у базі даних можуть зберігатися некоректні дані або може відбутися дублювання даних, тому доцільність застосування такого типу таблиць є досить спірним рішенням.

Система управління базами даних MySQL надає змогу зберігати та отримувати доступ до даних із використанням різних драйверів, які виконують операції над даними. Дана система дозволяє виконувати додаткове копіювання даних та розподіляти їх по різним таблицям. Така необхідність часто виникає в

досить складних та навантажених системах, які оброблюють велику кількість даних. В таких системах виникають проблеми, пов’язані з тим, що певна таблиця починає містити занадто велику кількість даних і виникає необхідність вилучити певні дані з цієї таблиці та розподілити по декількох таблицях для пришвидшення пошуку. MySQL має певні інструменти та команди, які дозволяють виконати операції з розподілу даних.

Важливою особливістю MySQL є те, що вона не потребує від користувачів вивчення додаткових мов для виконання запитів. Для користування цією системою та виконання запитів до бази даних, вони можуть використовувати стандартні команди SQL.

2.4 Фреймворк React Native

React Native схожий на React [9], але в якості будівельних блоків він використовує власні компоненти замість веб-компонентів. Щоб зрозуміти базову структуру додатків React Native, потрібно розуміти деякі основні концепції React, такі як JSX, компоненти state та props.

Програми, створені за допомогою React Native, не є мобільними веб-застосунками, тому що React Native використовує ці самі компоненти, що звичайні застосунки для iOS та Android. Замість того, щоб використовувати мову Swift, Kotlin або Java, ці компоненти збираються за допомогою JavaScript і React.

React Native [6] – це фреймворк, на основі якого лежить React.js, що дозволяє розширювати кроссплатформенні застосунки як для Android, так і для iOS.

Переваги React Native:

- Кросплатформність. Можна використовувати React Native для створення власних програм для платформ Android та iOS.
- Велике співтовариство. У Telegram є спільнота React Native на 3000 розробників.

- Великий вибір доступних компонентів і готових рішень.

З React Native [16] є два способи розроблення застосунку це – використовування Expo або React Native CLI.

Expo це – набір інструментів, бібліотек і сервісів для швидкого запуску. Це деякий API, який «з коробки» дає доступ до деяких можливостей пристрою: до камери, геолокації, push-сповіщень.

В Expo є інструменти налагодження і тестування програми. Алгоритм простий: встановлюємо на телефон додаток Expo Client, підключаємо телефон з комп'ютером в одну локальну мережу, заходимо в Expo на телефоні. Всі зміни в коді відображаються миттєво. З Expo не потрібні Xcode і Android Studio. Пишемо в середовищі розробки, збираємо застосунок за допомогою сервісу Expo, підписуємо і оновлюємо на льоту.

Але є і мінуси. Не можна додавати нативні модулі. Не можна оновитися на нову версію фреймворку React Native, поки не оновиться Expo.

React Native CLI. Його перевага в можливості кастомізації, та додаванні нативних модулів.

Але для підпису і збірки застосунку потрібен Xcode чи Android Studio. Щоб провести тестування застосунку, необхідно зробити збірку і завантажити на телефон, наприклад, за допомогою TestFlight для iOS або за допомогою прямого встановлення з ПК на телефон. Стандартна схема, але незручно і довго.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Мобільний застосунок для сповіщення клієнтів повинен мати функціонал авторизації користувача, функцію нагадування паролю, система має містити дві ролі – клієнта та адміністратора. Модуль адміністратора повинен виконувати функції створення, редагування, видалення подій та повідомлень, прийняття заявок від клієнтів. Модуль клієнта повинен мати функції перегляду доступних новин та сповіщення по заданій адресі, подачі заявок. Для реалізації всіх поставлених задач була обрана компонентна архітектура, яка є основним патерном розробки застосунків на React Native.

3.1 Архітектура програмної системи

Результатом розроблення успішного продукту є правильне будування архітектури програмної системи. Оскільки мобільний застосунок є тільки частиною системи, до якої входять API, яке надає методи редагування, створення, та видалення об’єктів даних. API написано за допомогою веб-фреймворку Laravel.

Веб-фреймворк Laravel представляє широкий набір функцій для веб-застосунків. Система базуватиметься на RESTful [14] архітектурі, що дозволить легко розділить клієнтський застосунок від програмних інтерфейсів для роботи з базою даних.

Серед переваг даної архітектури є простота уніфікованого інтерфейсу, прозорість зв'язків між компонентами системи для сервісних служб, переносимість компонентів системи шляхом переміщення програмного коду разом з даними. Побудована модель клієнт-сервер, тому частини системи працюють самостійно. У ролі системи керування базою даних було обрано MySQL. Це реляційна СКБД, що підтримує структуровану мову запитів SQL і

може застосовуватися у ролі SQL-сервера. Серед переваг можна відзначити безпечність, швидкодію, надійність та переносимість.

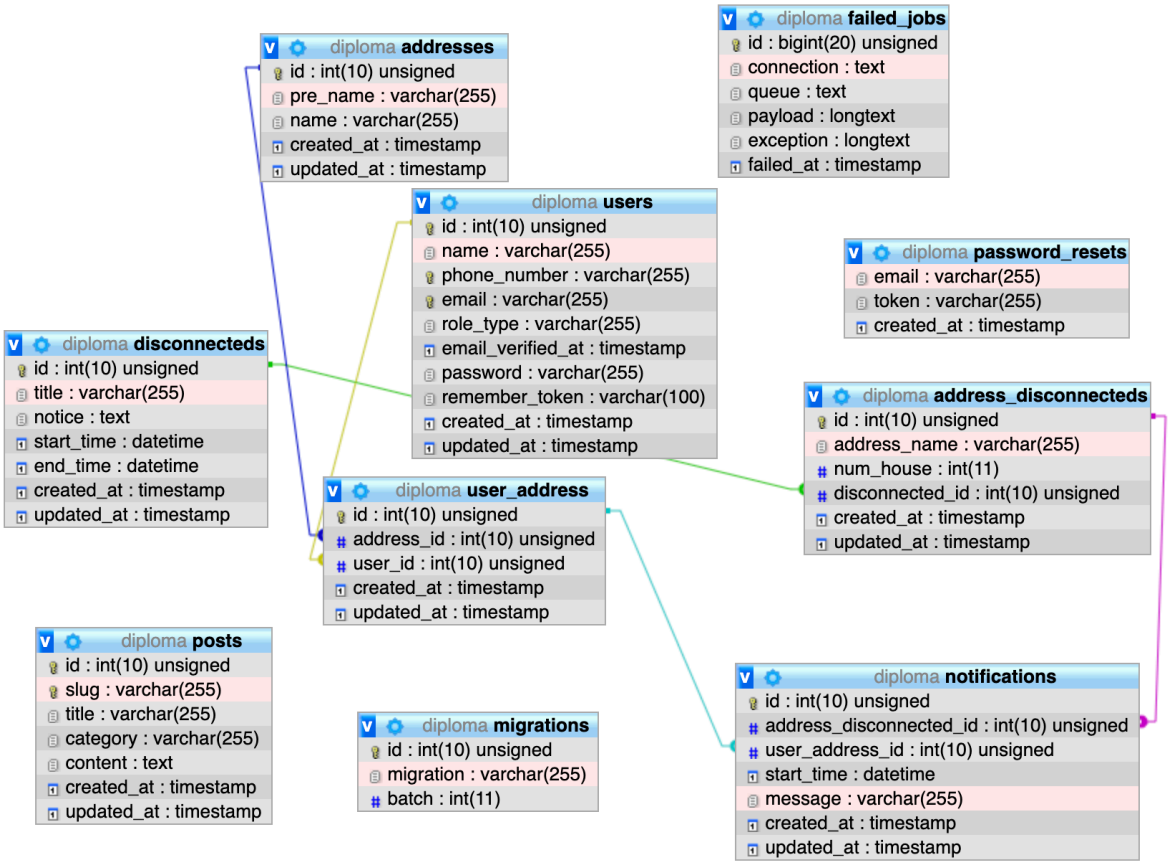

Рисунок 3.1 – Структура бази даних

Дана система має таку структуру бази даних, таблиці якої зображено на рисунку 3.1.

Таблиця users призначена для зберігання даних про користувачів системи. Дана таблиця має такі поля, як ім’я, поштова адреса, номер телефону, тип користувача та поля за допомогою яких можна переглянути коли він був створений, та коли останній раз відновлював свої дані користувач.

Таблиця addresses призначена для зберігання назв вулиць та дату створення чи оновлення даних певної адреси.

Disconnecteds зберігає дані, що стосуються відключення. Вона складається з наступних полів: з причини відключення, тобто з інформаційної частини, з дати та часу відключення та фактичної дати та часу усунення

проблеми, та поля що вказуватимуть дату та час розміщення або оновлення оголошення.

Таблиця address_disconnects призначена для зберігання адрес в яких планується відключення. Вона має такі поля: дані про відключення, назва вулиці, номер будинку, та інформація коли внесли чи оновили дані.

Проміжна таблиця user_address. Вона призначена для зберігання адрес користувачів. Кожен користувач може одночасно отримувати сповіщення з декількох адрес.

Posts призначена для збереження даних про новини в системі. В даній таблиці зберігаються привітання та новини. Таблиця складається з таких полів, як назві новини або привітання, контент, категорія новини, дату створення та оновлення публікації.

Та таблиця notification, що зберігає дані про сповіщення користувача по певним адресам. Дана таблиця містить інформацію по адресі, даті та часу відключення, адресу користувача за яким відбувається алгоритм виконання розсилки сповіщення та повідомлення, що відправляється.

Іншою частиною системи є веб-застосунок, який надає подібний функціонал, але у виконані веб-сайту. Веб-застосунок написаний за допомогою фреймворку Laravel на мові програмування PHP. Він також містить роль адміністратора та клієнта. Ролі адміністратора надано багато можливостей такі як, слідкування за роботою системи, створення та редагування події. Ця роль несе собою функції супер-адміна, який повністю володіє системою. Зв'язок між компонентами проектованої системи відображено на рисунку 3.2.

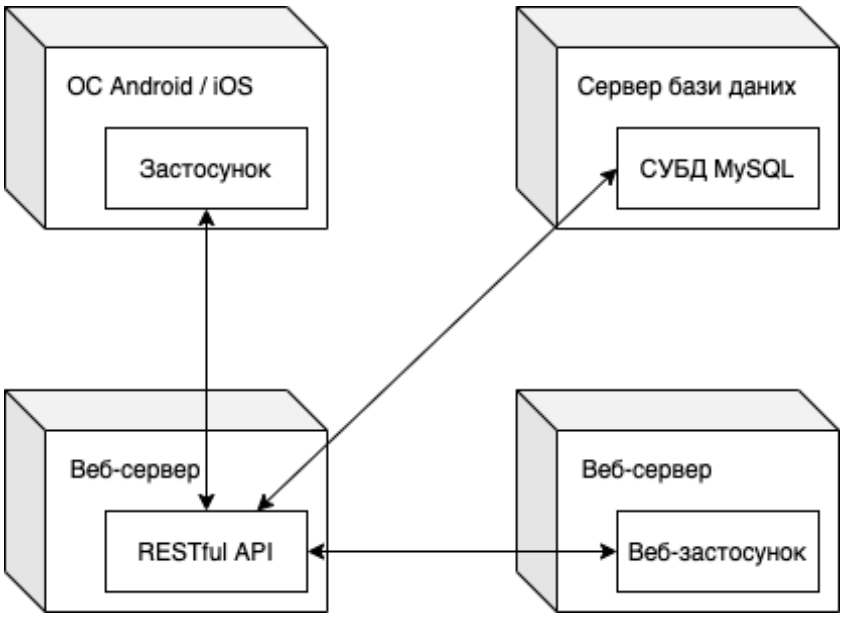

Рисунок 3.2 – Зв’язок між компонентами проектованої системи

Розглянемо детальніше архітектуру мобільного застосунку, та функціонал, який реалізовано для двох ролей користувачів. Створення системи для інформування користувачів за допомогою push-сповіщень передбачає насамперед визначення внутрішньої функціональності системи, тобто описання функції, які система повинна робити. Для визначення функціональних вимог була розроблена діаграма варіантів використання адміністратора системи та клієнта. Діаграма варіантів використання представлена на рисунках 3.3 та 3.4. Основними функціями для адміністратора є: авторизація, формування новин, визначення ролі користувача, створення, реагування та видалення подій. Тоді як для клієнта: авторизація, та підписка на розсилку.

Рисунок 3.3 – Діаграма варіантів використання

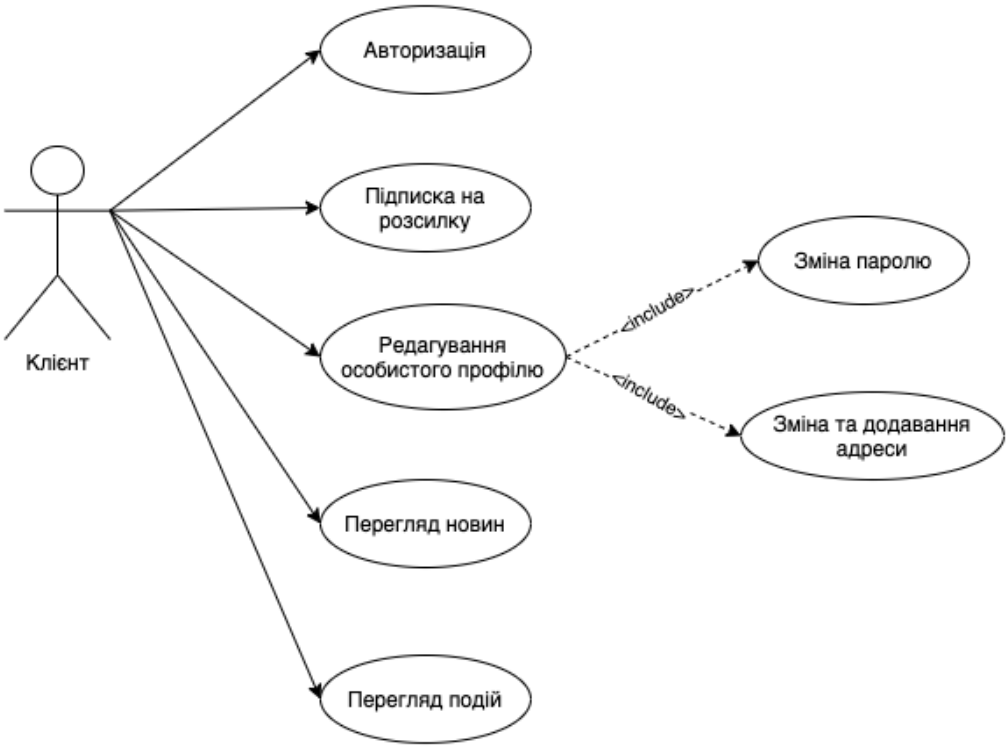

Рисунок 3.4 – Діаграма варіантів використання

Алгоритми створення, редагування, видалення новин, подій та інших об’єктів системи мають спільні методи та дії. Наприклад, для того, щоб виконати авторизацію користувача у систему необхідно зібрати дані з форми, які ввів користувач, в один об’єкт. Далі викликається метод, який робить запит на відповідну адресу та пересилає зібрані дані. Коли сервер отримує дані, він виконує роботу авторизації, в результаті чого повертає на клієнтську частину токен з об’єктом користувача, де знаходиться його інформація. Цей алгоритм зображено на діаграмі послідовностей (рисунок 3.5), що влучно підходить для опису подій.

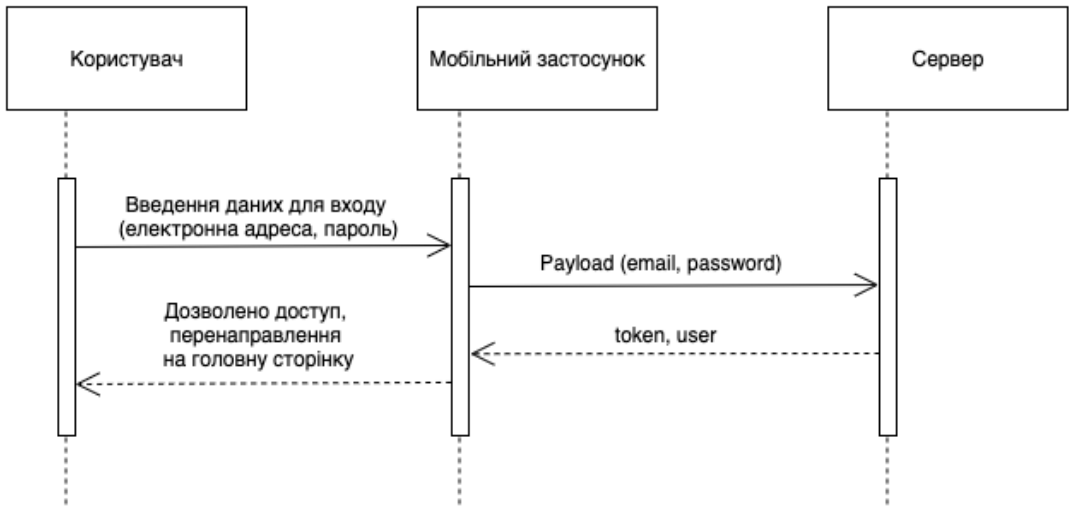

Рисунок 3.5 – Діаграма послідовності для функції “Авторизація”

Схожий алгоритм застосовується до усіх функції, які працюють даними, за винятком того, що при наступних запитах обов’язково потрібно слати токен та роль користувача, що повернувся нам від сервера при авторизації.

3.2 Розробка алгоритму виконання розсилки сповіщень

Персоналізація повідомлення передбачає використання персональної інформації отримувача. Для цього необхідно отримати інформації від користувача. Найкращим способом збору персональної інформації є

застосунок. Якщо при встановленні користувач підтвердив дозвіл на збирання персональної інформації, застосунок може зібрати інформацію у фоновому режимі та відіслати на сервер. Альтернативний спосіб – це використання форми для вводу даних, в якому потрібну інформацію користувач заповнить самостійно.

Всі сповіщення одного користувача надсилаються синхронно – поки не відправиться перше повідомлення, неможливо надіслати друге і так далі. Це неминуче впливає на пропускну здатність сервісу. Або можна запустити відправлення повідомлень в асинхронному режимі, наприклад, за допомогою технологій AJAX.

Для виконання розсилки буде використовувати наступний алгоритм. Є чотири параметри, якими можна керувати для вказання часу виконання команди. Це хвилини, години, дні місяця, та номер місяця. Та особиста інформація користувача, що вказуватиме потрібність відправки сповіщення, це є адреса користувача. Для того, щоб знати кому відправлено повідомлення створюється ще одна додаткова таблиця, куди записуватимуться користувачі, які вже отримали повідомлення. Таким чином, при виконанні запиту до бази даних для отримання користувачів додається перевірка на те, чи отримав даний користувач повідомлення, яке відноситься до поточної розсилки.

В логіку роботи скрипта додається перевірка, чи немає на даний момент запущених скриптів, які виконують надсилання сповіщень. Якщо такі скрипти є, то нічого не виконується, якщо немає – виконується розсилка.

Адміністратору потрібно обрати, за якою адресою надсилати повідомлення і натиснути кнопку «Розіслати». Після натискання на кнопку в базу даних записується подія, що потрібно виконати розсилку. На рисунку 3.6 зображений алгоритм виконання розсилки сповіщень.

Рисунок 3.6 – Алгоритм виконання розсилки сповіщень

3.3 Розробка веб-застосунку

Даний веб-застосунок використовує методологію MVC. Було створено моделі для роботи з таблицями та прописані зв’язки з іншими моделями. Та система має чітку валідацію даних, це дозволить уникнути непередбачуваних помилок. Було проведено розробку Restful API, що зображено на рисунку 3.7. Даний веб-застосунок має закриту систему, тобто доступ має тільки адміністратор, який додаватиме інформацію. Інші користувачі матимуть доступ тільки до мобільного застосунку. Всі можливі URL системи матимуть middleware, який перед виконанням певної логіки, виконає перевірку прав користувача.

```php
Route::group(['prefix' => 'admin', 'middleware' => 'admin', 'as' => 'admin.'], function () {
    Route::get( uri: '/home', action: 'HomeController@index')->name( name: 'home');
    Route::get( uri: '/admin', function () {
        return view( view: 'admin.index');
    });
    Route::get( uri: '/news', function () {
        return view( view: 'admin.news');
    });
    Route::get( uri: 'admin/user', action: 'Admin\UserController@index')->name( name: 'user.index');

    Route::resource( name: 'admin/news', controller: 'Admin\PostController')
        ->except(['show'])
        ->names( names: 'news');

    Route::resource( name: 'admin/address', controller: 'Admin\AddressController')
        ->except(['show'])
        ->names( names: 'address');

    Route::resource( name: 'admin/disconnected', controller: 'Admin\AddressDisconnectedController')
        ->except(['show'])
        ->names( names: 'disconnected');

    Route::get( uri: 'address/query', action: 'Admin\AddressDisconnectedController@queryAddress')->name( name: 'address.search');
});
```

Рисунок 3.7 – Restful API системи

Головний функціонал веб-застосунку, який відповідає за додавання, видалення та відображення відключень відповідає контролер Disconnected. В даному контролері присутні наступні методи: destroy, store, update.

Відповідно store, створює нові дані про відключення. В даному методі перевіряємо правильність введених даних та результаті помилки відбувається відображення повідомлення з відповідною помилкою.

```
public function store(Request $request) {
        $disconnected = Disconnected::create([
            'title' => $request->title,
            'start_time' => $request->start_time,
            'end_time' => $request->end_time,
            'notice' => $request->notice,
        ]);
        $disconnectedAddress = AddressDisconnected::create([
            'address_name' => $request->address_name,
            'num_house' => $request->num_house,
            'disconnected_id' => $disconnected->id,
        ]);
        if($disconnected && $disconnectedAddress) {
            return redirect()->route('disconnected.index',
[$disconnected->id])->with(['success' => 'Успешно сохранено']);
        } else {
            return back()->withErrors(['msg' => 'Ошибка
сохранения'])->withInput();
        }
```

```
    }
```

Для редагування даних використовується метод edit, якщо дані знайдені то відбувається оновлення даних, а інакше з’являється повідомлення про помилку. В даному методі створено набір правил, що дає можливість провести перевірку даних, щоб уникнути помилок у системі. Якщо дані були введені неправильно, відбувається відправлення повідомлення про помилку.

```
public function edit($id)
    {
        $disconnected = Disconnected::find($id);
        $disconnectedAddress = AddressDisconnected::find($id);
        return view('admin.disconnected.edit',
compact('disconnected', 'disconnectedAddress'));
    }
    public function update(Request $request, $id)
    {
        $disconnected = Disconnected::find($id);
        $disconnectedAddress = AddressDisconnected::find($id);
        $rules = [
            'title' => 'required|min:5|max:200',
            'notice' => 'required|string|min:5|max:10000',
            'start_time' => 'required',
            'end_time' => 'required',
        ];
        $message = [
            'title.required' => 'Введите заголовок статьи',
            'notice.min' => 'Минимальная длина статьи [:min]
символов',
        ];
        $data = $this->validate($request, $rules, $message);
        if(empty($disconnected && $disconnectedAddress)) {
            return back()->withErrors(["msg" => "Запись id
=[{$id}] не найдена"])->withInput();
        }
        $resultOne = $disconnected->fill($data)->save();
        $resultTwo = $disconnectedAddress->fill($data)->save();
        if($resultOne && $resultTwo) {
            return redirect()->route('disconnected.index',
[$disconnected->id])->with(['success' => 'Успешно сохранено']);
        } else {
            return back()->withErrors(['msg' => 'Ошибка
сохранения'])->withInput();
        }
    }
```

Відповідно метод destroy видаляє дані з бази даних. Спочатку знаходимо відповідну запис у таблиці, а потім видаляємо і в разі успіху отримаємо повідомлення, що запис успішно видалено.

```
    public function destroy($id)
    {
        $disconnectedAddress = AddressDisconnected::find($id);
        $disconnectedAddress->delete();
        return redirect()->route('disconnected.index')-
>with(['success' => 'Успешно удалено']);
    }
```

Додатково для перевірки правильності введеної адреси створено функцію queryAddress, яка при введені адреси відображає користувачу знайдені співпадіння з базою даних Addresses.

```
    public function queryAddress(Request $request) {
        $input = $request->all();
        $data = Address::select("name")
            ->where("name", "LIKE", "%{$input['query']}%")
            ->get();
        return response()->json($data);
    }
```

Для виконання всіх інших дій в системі створено відповідні контролери, аналогічного контролеру Disconnected.

Даний веб-застосунок має адміністративну частину, яка дозволяє додавати певну інформацію до веб-застосунку, для відображення інформації користувачу. Доступ до адміністративної частини має тільки адміністратор. На рисунках 3.8 – 3.9 зображена сторінка “Відключення”. На даній сторінці адміністратор може додавати причину, дату відключення та адресу.

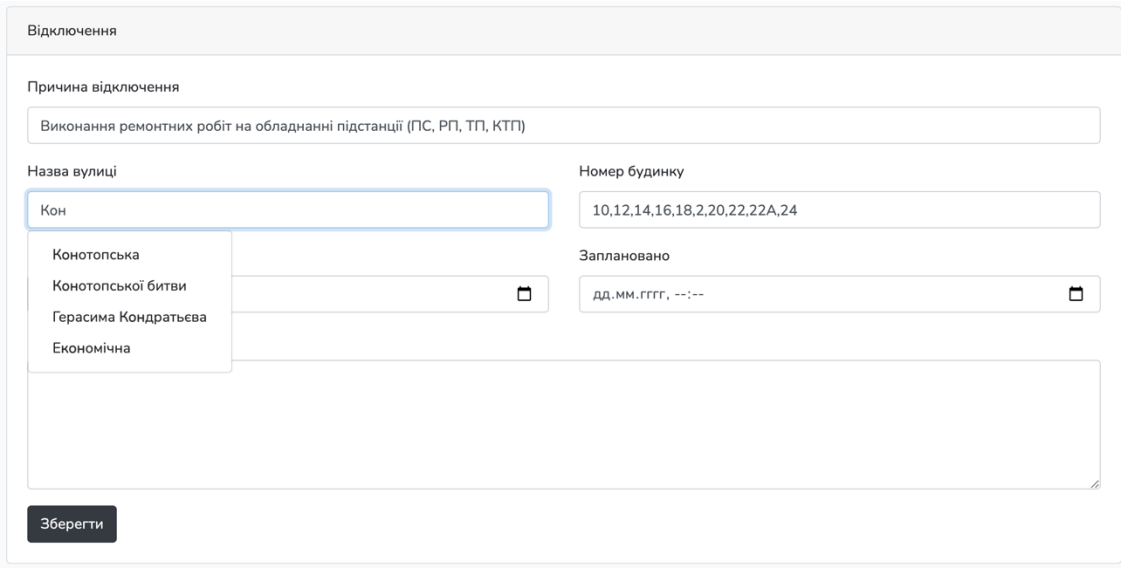

Рисунок 3.8 – Сторінка “Відключення”

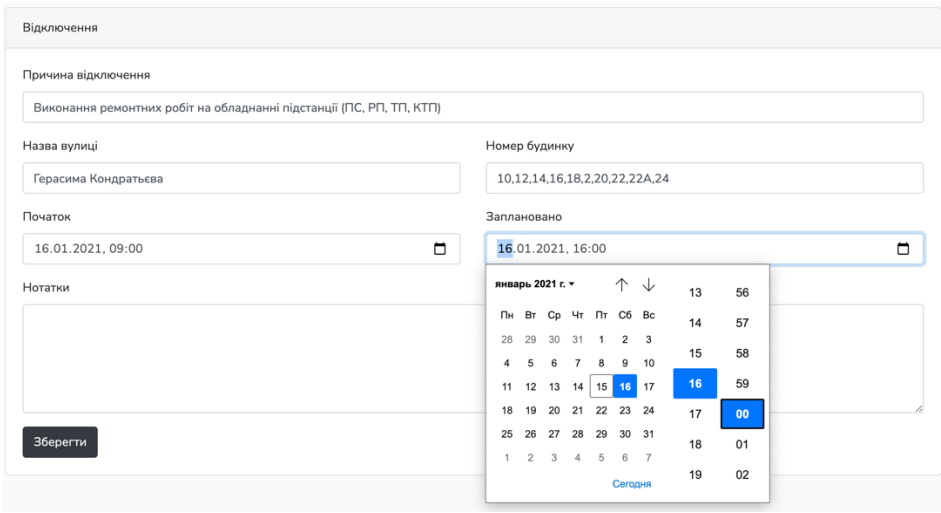

Рисунок 3.9 – Сторінка “Відключення”

3.4 Опис основного функціоналу застосунку

При розробці важливо підтримувати прозору та чітку архітектуру, та структуру директорій та файлів. На рисунку 3.10 зображено список директорій, які входять до даного проекту.

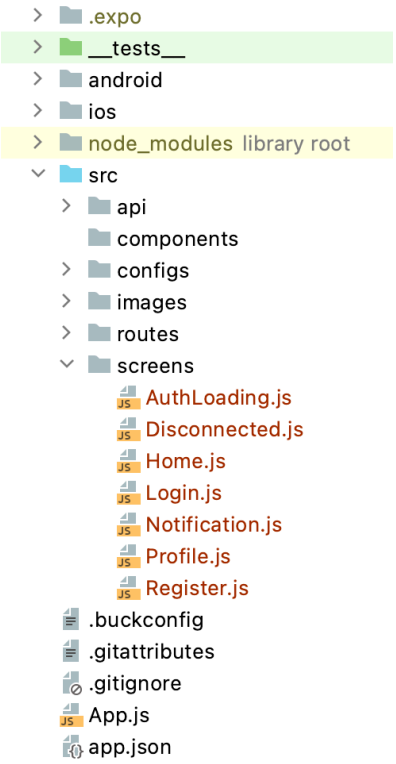

Рисунок 3.10 – Структура директорій

Директорія .expo є опціональною. Директорія tests відповідно зберігає тести. У директоріях android та ios містяться проекти на відповідних мовах, це Java та Swift, відповідно. Саме у них відбувається побудова проекту та його запуск в емуляторі.

Основною директорією, що використовується в роботі є src. В api директорії зберігаються запити до API. Директорія images відповідно зберігає картинки. У директорії components розміщаються спільні модулі, в більшості це модулі UI, наприклад, поля для вводу даних, кнопки, тощо. У configs директорії зберігаються дані кольорів, регулярних виразів. Головною директорією є routes, де розміщені сторінки та маршрути. Усі екрани знаходяться у директорії screens.

Таким чином, архітектура мобільного застосунку побудована вдало, вона складається з незалежних модулів та здатен до масштабованості та додавання нового функціоналу.

Як у більшості застосунків, робота у системі починається з авторизації користувачів (рис. 3.11) чи реєстрації нового користувача (рис.3.12).

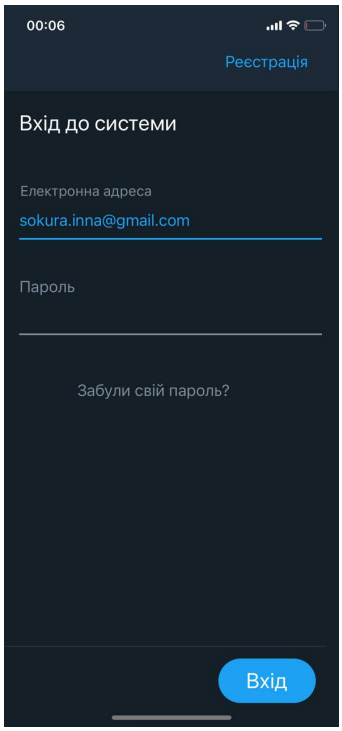

Рисунок 3.11 – Авторизація користувача

Даний метод викликається коли клікнути на кнопку, далі відбувається перевірка правильності написання даних, і виконується збір даних та відправка їх на сервер. Якщо відповідь позитивна – зберігаємо токен, і дані користувача (електронну пошту та пароль).

```
submitHandler = () => {
    if (!checkRequired(this, REQUIRED_FIELDS) &&
!checkPattern(this, PATTERN_FIELDS)) {
        const { login, navigation } = this.props;
        const { form } = this.state;
        login({
                email: form.email.trim(),
                password: form.password.trim(),},
            (res: Object) => {
                this.saveUser(form. email.trim(),
form.password.trim());
                this.saveToken(res.token);
                navigation.navigate('Home');
                69},
```

```
                () => this.setState(() => ({ errorLogin: true })),);
        }
}
```

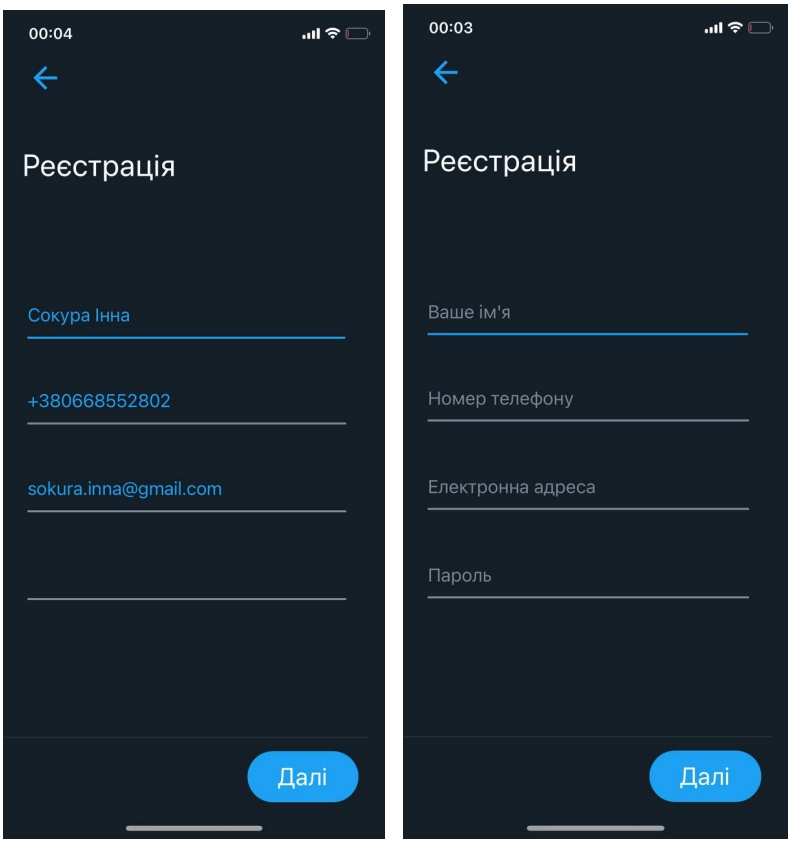

Рисунок 3.12 – Реєстрація нового користувача

Та створено авторизацію користувача з можливістю відновлення паролю. Даний функціонал є важливим у будь-якому застосунку. Для того щоб відновити пароль, потрібно перейти на сторінку відновлення паролю, натиснувши посилання “Забули свій пароль?”, після чого ввести свою електронну адресу, і далі натиснути кнопку “Отримати пароль”. Далі на електронну адресу, що ввів користувач, прийде повідомлення з паролем.

Користувачі, що авторизовані у системі мають доступ до новин (рис. 3.13) та інформації по всім відключенням(рис. 3.14).

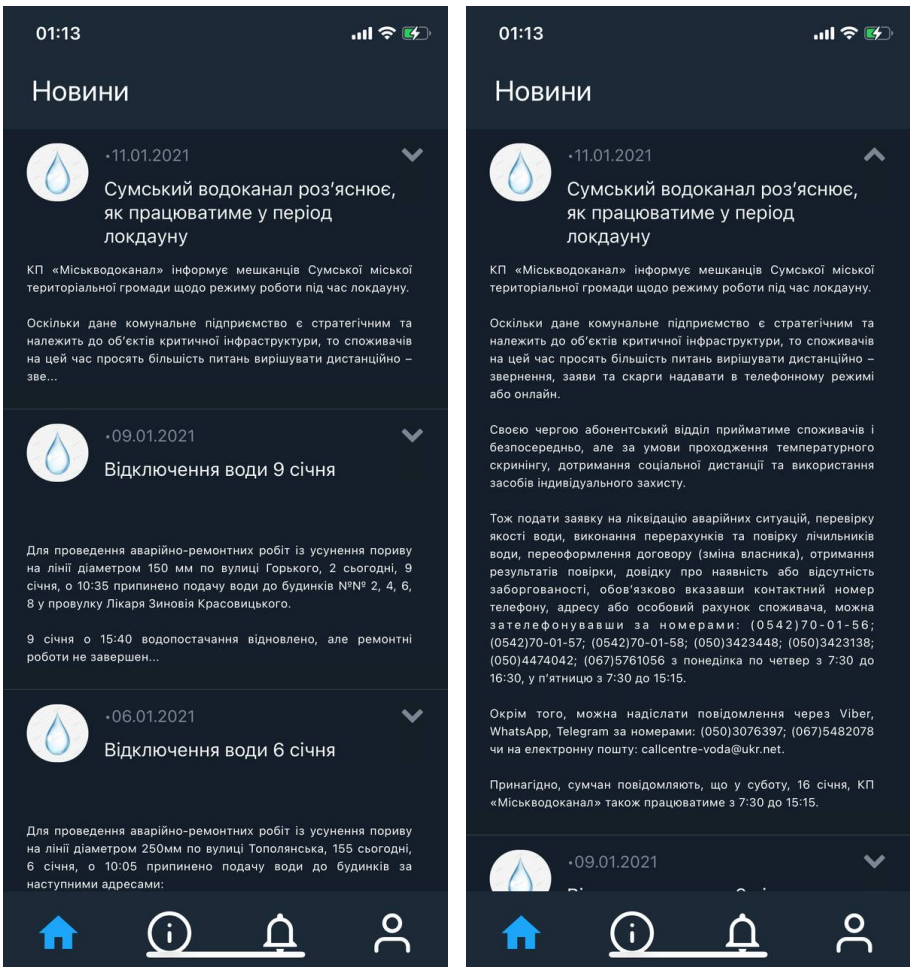

Рисунок 3.13 – Новини

Щоб загрузити дані у мобільний застосунок, в даному випадку по новинам, потрібно зробити запит на сервер. React Native надає Fetch API для мережевих потреб. Fetch API надає інтерфейс для отримання для отримання ресурсів.

```
const baseUrl = http://192.168.31.227:8888/diplom/;
constructor(props) {
        super(props);
        this.state = {
            dataNews:[],
        }
    }
    componentDidMount(){
        return fetch(baseUrl + "api/news")
            .then((response) => response.json())
            .then((responseJson) => {
                this.setState({
                    isLoading: false,
                    dataNews: responseJson,
                });
            })
            .catch((error) =>{
                console.error(error);
            })
    }
```

За допомогою даного коду отримуємо JSON дані по новинам. Далі визначаємо метод renderItem для підставки у FlatList. І отримуємо відформатований список новин та стилізуємо його.

```
renderItem = ({item}) => {
     return (
          <View style={styles.news}>
               <Text style={styles.date}>•{new
Date(item.created_at).toLocaleDateString()}</Text>
               <Text style={styles.title}>{item.title}</Text>
               <Text style={styles.content}>{item.content}</Text>
          </View>
     )
};
<FlatList
     data={this.state.dataNews}
     extraData={this.selectedId}
     keyExtractor={(item)=>item.id}
     renderItem={this._renderItem}
/>
```

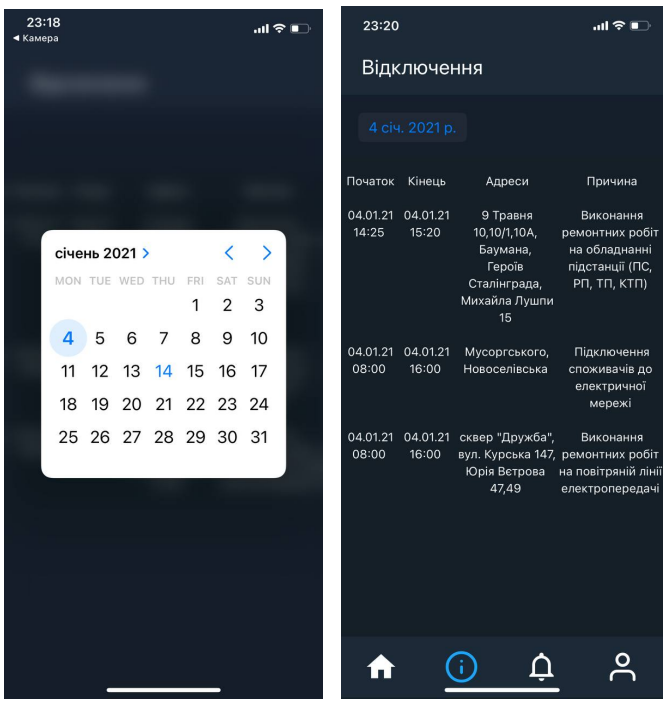

Рисунок 3.14 –Відключення

Сторінка профілю (рис. 3.15) містить його персональну інформацію, його ім’я та електронна адреса. Існує можливість редагування профілю, а саме зміна пошти та номеру телефону. На сторінці профілю можна одразу додати адресу по якій відбуватимуться сповіщення на мобільний застосунок.

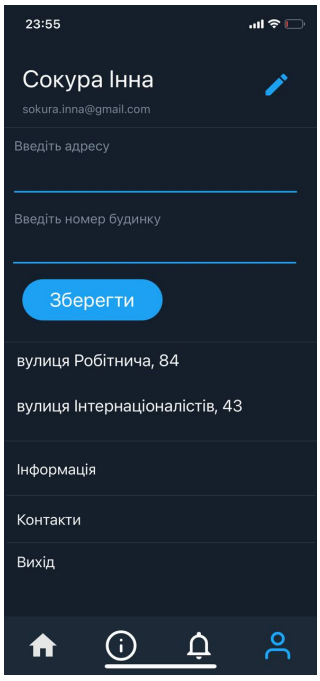

Рисунок 3.15 –Профіль користувача

Сторінка “Сповіщення” містить інформацію, яка сповіщає користувача про відключення води чи світла, яке відбується чи буде відбуватися найближчим часом за певною адресою, яка була введена на сторінці профілю користувача. Якщо адреса не збережена на даній сторінці нічого не виводиться. Далі дане сповіщення буде виведено на екран за межами звичайного застосунку, як зображено на рисунку 3.17.

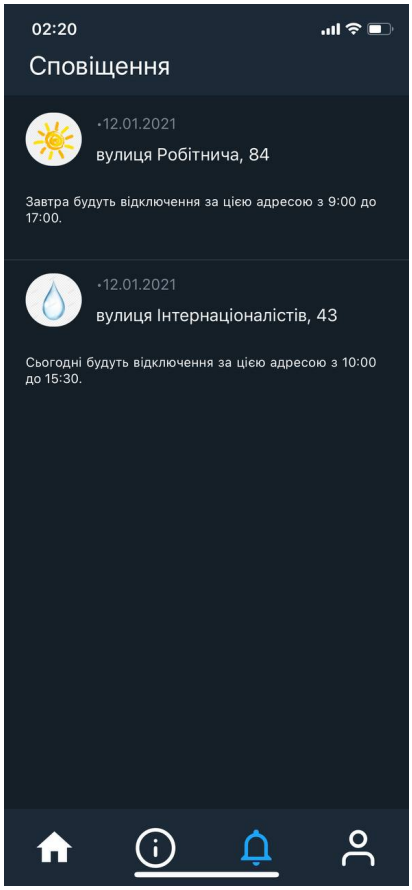

Рисунок 3.16 – Сторінка “Сповіщення"

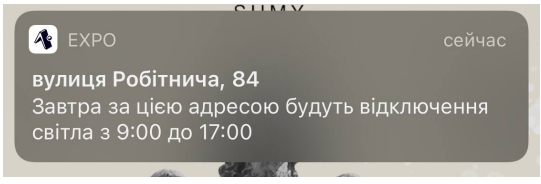

Рисунок 3.17 – Сповіщення

За допомогою функції _setNotificationHandler() можна встановити зворотній зв'язок, який вирішить, відображати сповіщення користувачу чи ні, коли сповіщення надходить під час запуску програми. Дана функція, завжди відображає сповіщення, коли воно отримане.

```
   _setNotificationHandler() {
        Notifications.setNotificationHandler({
            handleNotification: async () => ({
                shouldShowAlert: true,
                shouldPlaySound: false,
                shouldSetBadge: false,
            }),
        });
    }
```

scheduleNotificationAsync – планує сповіщення, яке буде ініційоване в майбутньому. createCalendarTrigger – це тригер, який призведе до того, що сповіщення буде доставлено один раз, коли компоненти дати та часу відповідатимуть зазначеним значенням. Слановуємо сповіщення, яке спрацює один раз, в певну дату та час.

```
export function createCalendarTrigger(date: Date, time: Time) {
    return {
        timeZone: date.timeZone,
        year: date.year,
        month: date.month,
        day: date.day,
        hour: time.hours,
        minute: time.minutes,
        repeats: false,
    }
}
scheduleNotificationCalendar(content, date: Date, time: Time) {
    Notifications.scheduleNotificationAsync({
        content: content,
        trigger: createCalendarTrigger(date, time),
    });
}
```

Функція, яка називається requestPermissionsAsync питає у користувача дозвіл на сповіщення. Просимо користувача дозволити відображати сповіщення, і відтворювати звук.

```
async requestPermission() {
  const permission = await
Notifications.requestPermissionsAsync();
  let granted = this._isPermissionGranted(permission)
   if (!granted)
      console.error('Permission to send notifications not given by
user')
      return permission;
    }
```

Дана функцій getPermissionsAsync перевіряє поточні налаштування дозволів, пов’язані із сповіщеннями. Перевіряємо, чи дозволено програмі на даний момент відображати сповіщення, та відтворювати звуки.

```
async allowsNotifications() {
  const settings = await Notifications.getPermissionsAsync();
  return this._isPermissionGranted(settings);
}
_isPermissionGranted(permission: NotificationPermissionStatus) {
  return permission.granted || permission.ios ?.status ===
Notifications.IosAuthorizationStatus.PROVISIONAL;
}
```

ВИСНОВКИ

Поширення мобільних технологій дає можливість використовувати нові види доставки повідомлень клієнтам, до яких відносяться технології push-повідомлень. Незважаючи на те, що мобільні push-сповіщення є відносно простою функцією, реалізувати їх досить складно, так як вони повинні реалізовуватися на основі сторонніх служб.

В даній роботі визначено мету, предмет та об’єкт дослідження, а також область, в якій воно проводиться. Проведено аналіз існуючих сервісів push-сповіщення та визначено їх основні переваги та недоліки. Проведений аналіз платформ для побудови системи push-сповіщень основних великих компаній-розробників операційних систем для мобільних пристроїв Google, та Apple. Було проаналізовано та сформовано вимоги до застосунку. На основі цих вимог було обрано технології та засоби реалізації.

Було описано процес розроблення алгоритму виконання розсилки сповіщень. Та детально розглянуто архітектуру даного мобільного застосунку.

Даний застосунок забезпечить користувачів надійною та актуальною інформацією, та зробить інформацію зрозумілою і доступною для всіх.

СПИСОК ЛІТЕРАТУРИ

1. Павлов, А. Д. Системы для поддержки push-уведомлений / А. Д. Павлов, Д. Е. Намиот // International Journal of Open Information Technologies. 2014. – С. 37-44.

2. Інформаційні системи, їх види; апаратне та програмне забезпечення інформаційної системи [Electronic resource]. – Access mode : http://www.kievoit.ippo.kubg.edu.ua/kievoit/2013/95/95.html

3. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript. — Санкт-Петербург: Питер. — 2013. – 496 с.

4. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения. — 2-е изд. — СПб. : Питер. — 2014. — 320 с.

5. Материалы официального сайта языка программирования PHP: [Electronic resource]. – Access mode : http://www.php.net/

6. React Native [Electronic resource] / Facebook Inc – 2020. – Access mode : https://reactnative.dev/

7. Мобільний застосунок [Electronic resource] : [Веб-сайт] // Вікіпедія – вільна енциклопедія. – Access mode : https://uk.wikipedia.org/wiki/Мобільний_застосунок (дата звернення: 04.01.21). – Назва. з екрана.

8. Современный учебник JavaScript [Electronic resource] – Access mode: https://learn.javascript.ru/.

9. React — A JavaScript library for building user interfaces [Electronic resource] — Access mode: https://reactjs.org/.

10. “Блекаут!” Планові відключення електроенергії в Одесі [Electronic resource]. – Access mode : http://app.blackout.today/

11. “1580”. Гаряча лінія міста Львова. [Electronic resource]. – Access mode : https://1580.lviv.ua/

12. CityBot Назар. [Electronic resource]. – Access mode : https://citybot.pro/.

13. Документація по фреймворку Laravel [Electronic resource]. – Access mode : https://laravel.com/

14. Опис технології RESTfull [Electronic resource]. – Access mode : https://uk.wikipedia.org/wiki/REST

15. Документація MySQL [Electronic resource]. – Access mode : https://www.mysql.com/

16. Bonnie Eisenman. Learning React Native: Building Native Mobile Apps with JavaScript, 2nd edition — Published by O'Reilly Media, 2016 — P. 150-400

ДОДАТКИ

Зміст файлу Home.js

```
import React, { Component } from "react";
import {StyleSheet, View, StatusBar, Text, ScrollView, FlatList,
Button, Image} from "react-native";
import Divider from "./component/Divider";
import MaterialCommunityIconsIcon from "react-native-vector-
icons/MaterialCommunityIcons";
import FeatherIcon from "react-native-vector-icons/Feather";
import { FontAwesome } from '@expo/vector-icons';
const baseUrl = "http://192.168.31.227:8888/diplom/public/"

export default class Home extends Component{

    constructor(props) {
        super(props);
        this.state = {
            dataNews:[],
            message: 'Hola',
            nameIcon: "chevron-down",
            selectedId: null,
            setSelectedId: null
        }
    }

    componentDidMount(){
        return fetch(baseUrl + "api/news")
            .then((response) => response.json())
            .then((responseJson) => {
                this.setState({
                    isLoading: false,
                    dataNews: responseJson,
                });
            })
            .catch((error) =>{
                console.error(error);
            })
    }

    render() {
        return (
            <View style={styles.rect}>
                <View style={styles.rect2Column}>
                    <View style={styles.rect2}>
                        <Text style={styles.text11}>Новини</Text>
                    </View>
                    <View style={styles.scrollArea} >
                        <FlatList
                            data={this.state.dataNews}
```

```
                            extraData={this.selectedId}

keyExtractor={(item)=>item.id.toString()}
                            renderItem={this._renderItem}
                        />
                    </View>
                    <View style={styles.rect2ColumnFiller}></View>
                    <View style={styles.rect6}>
                        <View style={styles.rect7}>
                            <MaterialCommunityIconsIcon
                                name="home"
                                style={styles.icon6}
                            ></MaterialCommunityIconsIcon>
                            <FeatherIcon name="info"
style={styles.icon7}></FeatherIcon>
                            <MaterialCommunityIconsIcon
                                name="bell-outline"
                                style={styles.icon8}
                            ></MaterialCommunityIconsIcon>
                            <FeatherIcon name="user"
style={styles.icon9}></FeatherIcon>
                        </View>
                    </View>
                </View>
            </View>
        );
    }


    _renderItem = ({item}) => {
        const {selectedId} = this.state;
        return (
            <ScrollView
                horizontal={false}

contentContainerStyle={styles.scrollArea_contentContainerStyle}>
                <View style={[styles.rect, styles.tweet]}>
                    <View style={styles.imageRow}>
                        <Image
                            source={item.category === 'Свет' ?
require("./assets/son.png") : require("./assets/woda.png")}
                            resizeMode="cover"
                            style={styles.image}></Image>
                        <View style={styles.text3Column}>
                            <Text style={styles.text3}>•{new
Date(item.created_at).toLocaleDateString()}</Text>
                            <Text
style={styles.text}>{item.title}</Text>
                        </View>
                        <FontAwesome
                            //name="chevron-down"
                            onPress={() => {
```

```
                                    if (selectedId === item.id) {
                                        this.setState({selectedId:
null})
                                    } else {
                                        this.setState({selectedId:
item.id})
                                        console.log(selectedId);
                                    }
                                }}
                                name={item.id === selectedId ?
"chevron-up" : "chevron-down"}
                                id={this.state.nameIcon + item.id}
                                style={styles.icon}
                                backgroundColor={"rgba(20,31,40,1)"}
                            ></FontAwesome>
                        </View>
                        <Text style={styles.text4}>
                            {item.id === selectedId ? item.content :
item.content.substring(0, 300) + "..."}
                        </Text>
                    </View>
                    <Divider style={styles.divider}></Divider>
                </ScrollView>
            )
        };}

const styles = StyleSheet.create({
    rect: {
        flex: 1,
        backgroundColor: "#141f2b"
    },
    divider: {
        height: 1,
        marginTop: 20,
    },
    scrollArea: {
        height: 644,
        backgroundColor: "rgba(20,31,40,1)"
    },
    rect2Column: {},
    rect2: {
        height: 104,
        backgroundColor: "#1c2a38"
    },
    text11: {
        color: "rgba(255,255,255,1)",
        fontSize: 24,
        marginTop: 57,
        marginLeft: 24
    },
    rect2ColumnFiller: {
        flex: 1
```

```
},
rect6: {
    height: 71,
    backgroundColor: "#1c2a38"
},
rect7: {
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-around",
    flex: 1
},
icon6: {
    color: "#1da6fa",
    fontSize: 40
},
icon7: {
    color: "rgba(255,255,255,1)",
    fontSize: 40
},
icon8: {
    color: "rgba(255,255,255,1)",
    fontSize: 40
},
icon9: {
    color: "rgba(255,255,255,1)",
    fontSize: 40
},
image: {
    width: 52,
    height: 49,
    borderRadius: 100,
    borderColor: "#1c2a38",
    borderWidth: 0,
    marginTop: 2
},
text3: {
    color: "#798894",
    fontSize: 14,
    lineHeight: 20
},
text: {
    color: "rgba(255,255,255,1)",
    fontSize: 16,
    lineHeight: 20,
    marginTop: 8,
    marginBottom: 10,
},
text3Column: {
    width: 250,
    height: 80,
    marginLeft: 13,
    marginTop: 2,
```

```
        textAlign: 'justify',
    },
    icon: {
        color: "#798894",
        fontSize: 20,
        backgroundColor: "rgba(20,31,40,1)"
    },
    Button: {
        backgroundColor: "rgba(20,31,40,1)"
    },
    imageRow: {
        height: 51,
        flexDirection: "row",
        marginTop: 10,
        marginLeft: 20,
        marginRight: 20
    },
    text4: {
        width: 325,
        height: 'auto',
        color: "rgba(255,255,255,1)",
        fontSize: 10,
        lineHeight: 15,
        marginTop: 50,
        marginLeft: 20,
        textAlign: 'justify',
    },
});
```

Зміст файлу component/Post.js

```
import React, { Component } from "react";
import {StyleSheet, View, Image, Text, Button, ScrollView,
TouchableOpacity} from "react-native";



function Post(props) {

  return (
    <View style={[styles.rect, props.style]}>
      <View style={styles.imageRow}>
        <Image
             source={require("../assets/son.png")}
                  resizeMode="cover"
                  style={styles.image}></Image>
        <View style={styles.text3Column}>
          <Text style={styles.text3}>•{props.Dates}</Text>
          <Text style={styles.text}>{props.Title}</Text>
        </View>
      </View>
```

```
      <Text style={styles.text4}>
        { props.Content }
      </Text>
    </View>
  );
}

const styles = StyleSheet.create({
  rect: {},
  image: {
    width: 52,
    height: 49,
    borderRadius: 100,
    borderColor: "#1c2a38",
    borderWidth: 0,
    marginTop: 2
  },
  text3: {
    color: "#798894",
    fontSize: 14,
    lineHeight: 20
  },
  text: {
    color: "rgba(255,255,255,1)",
    fontSize: 16,
    lineHeight: 20,
    marginTop: 8,
    marginBottom: 10,
  },
  text3Column: {
    width: 250,
    height: 80,
    marginLeft: 13,
    marginTop: 2,
    textAlign: 'justify',
  },
  icon: {
    color: "#798894",
    fontSize: 20,
    backgroundColor: "rgba(20,31,40,1)"
  },
  Button: {
    backgroundColor: "rgba(20,31,40,1)"
  },
  imageRow: {
    height: 51,
    flexDirection: "row",
    marginTop: 10,
    marginLeft: 20,
    marginRight: 20
  },
  text4: {
```

```
    width: 325,
    height: 80,
    color: "rgba(255,255,255,1)",
    fontSize: 12,
    lineHeight: 15,
    marginTop: 25,
    marginLeft: 20,
    textAlign: 'justify',
  },
});
export default Tweet;
```

Зміст файлу component/Divider.js

```
import React, { Component } from "react";
import { StyleSheet, View } from "react-native";

function Divider(props) {
  return <View style={[styles.rect, props.style]}></View>;
}

const styles = StyleSheet.create({
  rect: {
    backgroundColor: "#2a343c"
  }
});

export default Divider;
```

Зміст файлу Login.js

```
import React, { Component } from "react";
import {
  StyleSheet,
  View,
  StatusBar,
  Text,
  TouchableOpacity,
  TextInput
} from "react-native";
import Divider from "./component/Divider";

function Login(props) {
type _t_props = {|
 login: request,
 navigation: Object
|}
type _t_state = {|
 form: Object,
 errors: Object,
```

```
 errorLogin: boolean
|}
const REQUIRED_FIELDS = ['email', 'password'];
const PATTERN_FIELDS = {
 email: ERRORS_CONFIG.email,
 password: {
 pattern: PATTERN_CONFIG.password,
 message: 'Довжина пароля не менше 6 символів!'
 }
};

const INITIAL_FORM = {
 email: '',
 password: '',
};
@inject(({ appStore }) => ({
 login: appStore.login,
}))
@observer
class LoginForm extends Component<_t_props, _t_state> {

 state = {
 form: INITIAL_FORM,
 errors: {},
 errorLogin: false
 }
/
 async componentDidMount() {
 const email = await AsyncStorage.getItem('Email');
 const password = await AsyncStorage.getItem('Password');
 if (email && password) {
 this.setState(() => ({
 form: {
 email: email,
 password,
 }
 }));
 }
 }
 onChangeText = (key: string, text: string) => {
 this.setState(prevState => ({
 form: { ...prevState.form, [key]: text },
68
 errors: removeField(prevState.errors, key),
 errorLogin: false
 }));
 }
 saveToken = async (token: string) => {
  try {
 await AsyncStorage.setItem('Auth', token);
 } catch (error) {
 console.log(error);
```

```
 }
 };
 saveUser = async (email: string, password: string) => {
 try {
 await AsyncStorage.setItem('Email', email);
 await AsyncStorage.setItem('Password', password);

 } catch (error) {
 console.log(error);
 }
 };

 submitHandler = () => {
 if (!checkRequired(this, REQUIRED_FIELDS) && !checkPattern(this,
PATTERN_FIELDS)) {
 const { login, navigation } = this.props;
 const { form } = this.state;
 login(
 {
 email: form.email.trim(),
 password: form.password.trim(),
 },
 (res: Object) => {
  this.clearForm();
 this.saveUser(form.email.trim(), form.password.trim());
 this.saveToken(res.token);
 navigation.navigate( 'Home');
69
 },
 () => this.setState(() => ({ errorLogin: true })),
 );
 }
 }
 render() {
 const { navigation } = this.props;
 const {
 form: { email, password}, errors, errorLogin
 } = this.state;
  return (
      <View style={styles.rect}>
        <View style={styles.textColumn}>
          <Text style={styles.text}>Вхід до системи</Text>
          <View style={styles.rect2}>
            <View style={styles.button3Filler}></View>
            <TouchableOpacity style={styles.button3}>
              <TouchableOpacity
                  style={styles.button}
              >
                <Text style={styles.text2}>Реєстрація</Text>
              </TouchableOpacity>
            </TouchableOpacity>
          </View>
```

```
          <Text style={styles.text3}>
            Електронна адреса
          </Text>
          <Text style={styles.text4}>Пароль</Text>
          <TextInput
              placeholder=""
              style={styles.textInput}
          ></TextInput>
          <Text style={styles.text5}>Забули свій пароль?</Text>
          <TextInput placeholder=""
style={styles.textInput2}></TextInput>
        </View>
        <View style={styles.textColumnFiller}></View>
        <View style={styles.rect4}>
          <Divider style={styles.divider}></Divider>
          <TouchableOpacity
              style={styles.button2}
          >
            <Text style={styles.text6}>Вхід</Text>
          </TouchableOpacity>
        </View>
      </View>
  );
}

const styles = StyleSheet.create({
  rect: {
    flex: 1,
    backgroundColor: "#141f28"
  },
  text: {
    color: "rgba(255,255,255,1)",
    fontSize: 24,
    marginTop: 118,
    marginLeft: 18
  },
  rect2: {
    height: 118,
    backgroundColor: "#1c2a38",
    flexDirection: "row",
    marginTop: -168
  },
  button3Filler: {
    flex: 1,
    flexDirection: "row"
  },
  button3: {
    width: 86,
    height: 150,
    marginTop: 34
  },
  button: {
```

```
  width: 85,
  height: 50,
  justifyContent: "center"
},
text2: {
  width: 166,
  height: 100,
  marginTop: 130,
  color: "#1da1f2",
  fontSize: 18,
  alignSelf: "center"
},
text3: {
  color: "rgba(123,139,151,1)",
  fontSize: 16,
  lineHeight: 20,
  marginTop: 102,
  marginLeft: 19
},
text4: {
  color: "rgba(123,139,151,1)",
  fontSize: 18,
  marginTop: 87,
  marginLeft: 18
},
textInput: {
  width: 339,
  height: 42,
  color: "#1da1f2",
  borderColor: "rgba(123,139,151,1)",
  borderWidth: 0,
  borderBottomWidth: 2,
  fontSize: 18,
  marginTop: 2,
  marginLeft: 18
},
text5: {
  color: "#7b8b97",
  fontSize: 18,
  marginTop: 51,
  marginLeft: 83
},
textInput2: {
  width: 339,
  height: 42,
  color: "#1da1f2",
  borderColor: "#1da1f2",
  borderWidth: 0,
  borderBottomWidth: 2,
  fontSize: 18,
  marginTop: -222,
  marginLeft: 18
```

```
  },
  textColumn: {
    marginLeft: -1,
    marginRight: 1
  },
  textColumnFiller: {
    flex: 1
  },
  rect4: {
    width: 375,
    height: 91,
    alignSelf: "center"
  },
  divider: {
    width: 360,
    height: 1
  },
  button2: {
    width: 109,
    height: 50,
    backgroundColor: "#1da1f2",
    borderRadius: 100,
    justifyContent: "center",
    marginTop: 13,
    marginLeft: 240
  },
  text6: {
    color: "#ffffff",
    fontSize: 24,
    alignSelf: "center"
  }
});

export default Login;
```

Зміст файлу Notification/ActiveNotification.php

```
<?php
namespace App\Notifications;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;
use Illuminate\Notifications\Notification;
use NotificationChannels\ExpoPushNotifications\ExpoChannel;
use NotificationChannels\ExpoPushNotifications\ExpoMessage;

class ActivateNotification extends Notification
{
    use Queueable;

    public function via($notifiable)
```

```
    {
        return [ExpoChannel::class];
        //return ['mail'];
    }

    public function toExpoPush($notifiable)
    {
        return ExpoMessage::create()
            ->badge(1)
            ->enableSound()
            ->title("Account activated")
            ->body("Your account has been activated")
            ->priority('high')
            ->setChannelId("chat-messages");
    }

    public function toArray($notifiable)
    {
        return [
            //
        ];
    }
}
```

Зміст файлу Model/Post.php

```
<?php
namespace App\Model;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Notifications\Notifiable;

class Post extends Model
{
    use Notifiable;

    protected $fillable = [
        'title',
        'slug',
        'category',
        'content',
    ];
}
```

Зміст файлу Admin/Controller/PostController.php

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
```

```
use App\Model\Post;
use App\Notifications\ActivateNotification;
use ExponentPhpSDK\Expo;
use Illuminate\Http\Request;

class PostController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $paginator = Post::paginate(25);

        return view('admin.news.index', compact('paginator'));
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        $arrCategory = array('Вода', 'Свет');
        $news = new Post();

        return view('admin.news.create', compact('news',
'arrCategory'));
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {

        $rules =[
            'title' => 'required|min:5|max:200|unique:posts',
            'slug' => 'max:200',
            'category' => 'required',
            'content' => 'required|string|min:5|max:10000',
            ];

        $message = [
            'title.required' => 'Введите заголовок новости',
```

```
            'content.min' => 'Минимальная длина статьи [:min]
символов',
            ];

        $data = $this->validate($request, $rules, $message);

        if(empty($data['slug'])) {
            $data['slug'] =
\Illuminate\Support\Str::slug($data['title']);
        }

        $news = (new Post())->create($data);

        if($news) {
            $news->notify(new ActivateNotification());
            return redirect()->route('news.index', [$news->id])
                ->with(['success' => 'Успешно сохранено']);
        } else {
            return back()->withErrors(['msg' => 'Ошибка
сохранения'])
                ->withInput();
        }
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        $arrCategory = array('Вода', 'Свет');

        $news = Post::find($id);
        return view('admin.news.edit', compact('news',
'arrCategory'));
    }

    /**
     * Update the specified resource in storage.
```

```
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {

        $news = Post::find($id);

        $rules = [
            'title' => 'required|min:5|max:200',
            'slug' => 'max:200',
            'category' => 'required',
            'content' => 'required|string|min:5|max:10000',
            ];

        $message = [
            'title.required' => 'Введите заголовок статьи',
            'content.min' => 'Минимальная длина статьи [:min]
символов',
        ];

        $data = $this->validate($request, $rules, $message);

        if(empty($data['slug'])) {
            $data['slug'] =
\Illuminate\Support\Str::slug($data['title']);
        }

        if(empty($news)) {
            return back()
                ->withErrors(["msg" => "Запись id =[{$id}] не
найдена"])
                ->withInput();
        }

        //$data = $request->all();

        $result = $news
            ->fill($data)
            ->save();

        if($result) {
            return redirect()
                ->route('news.index', $news->id)
                ->with(['success' => 'Успешно сохранено']);
        } else {
            return back()
                ->withErrors(['msg' => 'Ошибка сохранения'])
                ->withInput();
        }
```

```
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        $news = Post::find($id);
        //dd($news);
        $news->delete();
        return redirect()->route('news.index')->with(['success' =>
'Успешно удалено']);
    }
}
```