

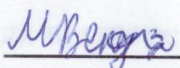
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра

на тему: «Проект Web-додатку для приймача Інтернет-
радіо на базі платформи Arduino»

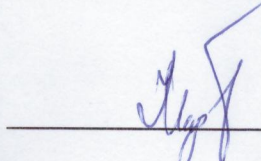
Виконав студент 4-го курсу групи КІ-4
спеціальності 123 «Комп'ютерна інженерія»

 Венедчук Михайло

Керівник, старший викладач кафедри комп'ютерних
наук

 Петросян Р.В.

Рецензент, ВО завідувача кафедри комп'ютерних
наук, к.т.н

 Граф М. С.

Державний університет «Житомирська політехніка»

Факультет інформаційно-комп'ютерних технологій

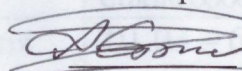
Кафедра комп'ютерної інженерії та кібербезпеки

Спеціальність 123 «Комп'ютерна інженерія»

Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко

1 березня 2022 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Венедчук Михайло Геннадійович

Тема роботи: «Проект Web-додатку для приймача Інтернет-радіо на базі платформи Arduino»

затверджена Наказом університету від 1 березня 2022 р. № 127/с

Термін здачі студентом закінченої роботи 10 червня 2022 р.

Вихідні дані роботи:

Предметом дослідження є розробка пристрою для відтворення Інтернет-радіо на базі платформи Arduino та додатку до нього.

Об'єктом дослідження є використання технологій для розробки IoT пристроїв, та додатків до них.

Обладнання та програмне забезпечення:

1. VSCode
2. Arduino IDE
3. Wi-Fi модуль Wemos D1 mini
4. Аудіо роз'єм 2.5 мм
5. Перемички для макетування 3 шт.

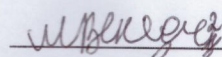
Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Петросян Р.В., ст. викладач кафедри КН	20.04.22	20.04.22
2	Петросян Р.В., ст. викладач кафедри КН	10.05.22	10.05.22
3	Петросян Р.В., викладач кафедри КІ та КБ	25.05.22	25.05.22

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз отриманого завдання та уточнення поставлених завдань	01.03 – 6.03.22	виконано
2	Постановка задачі дослідження	07.03 – 14.03.22	виконано
3	Пошук, огляд та аналіз аналогічних розробок	15.03 – 21.03.22	виконано
4	Аналіз літературних джерел	22.03 – 28.03.22	виконано
5	Проектування структури програми, написання програмного коду вебсервера на базі Arduino	29.03 – 12.04.22	виконано
6	Тестування роботи вебсервера	03.04 – 11.04.22	виконано
7	Аналіз бібліотек React, Redux для побудови додатку	12.04 – 24.04.22	виконано
8	Розробка інтерфейсу за допомогою засобів React Native	25.04 – 01.05.22	виконано
9	Пошук необхідного API, аналіз роботи з ним	02.05 – 08.05.22	виконано
10	Розробка керування станом додатку за допомогою бібліотеки Redux	09.05 – 10.05.22	виконано
11	Аналіз бібліотеки RxJS	11.05 – 13.05.22	виконано
12	Налаштування асинхронності за допомогою бібліотеки RxJS	14.05 – 18.05.22	виконано
13	Налагодження роботи з додатком	19.05 – 25.05.22	виконано
14	Оформлення пояснювальної записки	26.05 – 02.06.22	виконано
15	Підготовка презентації та виступу	03.05 – 08.06.22	виконано

Студент



М.Г. Венедчук

Керівник проекту



Р.В. Петросян

АНОТАЦІЯ

Кваліфікаційна робота складається з веб сервера для Arduino, завданням якого є передача звуку на аудіо пристрій; та клієнтського додатку, який передає дані, про те який саме потік треба приймати, та передавати аудіо пристрою. Пояснювальна записка до роботи містить 85 сторінок та 27 ілюстрацій.

Метою роботи є розробка приймача аудіо потоків.

В результаті роботи набуто навичок роботи з Arduino, створено вбудовану програму в межах якої побудовано вебсервер, на основі пристрою, який здатний передавати інформацію про доступні бездротові точки доступу, підключатися до них, приймати аудіо потік, передаючи його на пристрій для аудіо відтворення. Створено додаток за допомогою бібліотек React, Redux, RxJS, який має змогу підключати Arduino пристрій до доступних бездротових точок доступу, передавати йому посилання на аудіо потік, показувати доступні для програвання інтернет потоки; зберігати та додавати їх самостійно за допомогою рушія зберігання SQLite.

Ключові слова: ARDUINO, ВЕБСЕРВЕР, АУДІО ПОТІК, ВБУДОВАНА ПРОГРАМА, БЕЗДРОВОТА ТОЧКА ДОСТУПУ, REACT, REDUX, RXJS, SQLITE.

					KI.KP.B -123 – 09 - ПЗ			
Змін	Арк.	№ докум.	Підпис	Дата	Проект Web-додатку для приймача Інтернет-радіо на базі платформи Arduino	Лім.	Арк.	Аркушів
Розроб.		Венедчук М.Г.		28.06				
Керівник		Петросян Р.В.		28.06			4	88
Реценз.		Граф М.С.		28.06		ФІКТ, гр. KI-4		
Н. Контр.		Петросян Р.В.		28.06				
Затверд.		Єфіменко А.А.		28.06				

ANNOTATION

The qualification work consists of a web server on Arduino, the task of which is to transfer audio to an audio device, and a client application that transmits data about which stream to receive, and transmits audio to the device. The explanatory note to the work contains 85 pages and 27 illustrations.

The aim of this work is to develop a receiver of audio streams.

As a result, the skills of working with Arduino were acquired, was created firmware with web server, for a device that is able to transmit information about available wireless access points, connect to them, receive an audio stream, transmitting it to a device for audio playback. An application was created with the help of React, Redux, RxJS libraries, which allows to connect Arduino device to available wireless access points, send the audio stream links to it, show available for playback Internet streams; save and add them using the SQLite storage engine.

Keywords: ARDUINO, БЕБСЕРБЕР, AUDIO STREAM, FIRMWARE, WEB SERVER, WIRELESS ACCESS POINT, REACT, REDUX, RXJS, SQLITE.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						5
Зм.	Арк	№ докум	Підпис	Дата		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ТЕХНІЧНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ ПРИЙМАЧА	9
1.1. Аналіз апаратного забезпечення та роботи з ним	9
1.2. Аналіз сервісів, які дозволяють прослуховувати інтернет радіо	11
1.3. Аналіз засобів для проектування додатку	13
Висновок до розділу 1	15
РОЗДІЛ 2. РОЗРОБКА КЛІЄНТСЬКОГО ДОДАТКУ	16
2.1. Аналіз реалізації для діставання аудіо потоку	16
2.2. Проектування сторінки зі списком доступних потоків	17
2.3. Підключення бібліотеки Redux	26
2.4. Підключення бібліотеки RxJS	30
2.5. Проектування сторінки з збереженими станціями	33
2.6. Проектування сторінки для підключення пристрою до мережі	43
2.7. Додавання функцій на головній сторінці	45
Висновок до розділу 2	48
РОЗДІЛ 3. РОЗРОБКИ ВЕБСЕРВЕРА ДЛЯ ПРИЙМАЧА НА ARDUINO	49
3.1. Розробка алгоритму пристрою	49
3.2. Розробка схеми	50
3.3. Створення точки доступу, вебсервера, та приєднання до іншої точки доступу	52
Висновок до розділу 3	58
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
ДОДАТКИ	64

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

DLNA – Digital Living Network Alliance

RxJS – Reactive Extensions Library for JavaScript

JSON – JavaScript Object Notation

API – Application Programming Interface

URL – Uniform Resource Locator

EEPROM – Electrically Erasable Programmable Read-Only Memory

PSK – Pre-shared key

HTTP – HyperText Transfer Protocol

mDNS – Multicast Domain Name System

I²S – Inter-IC Sound

ІКМ – Імпульсно-кодова модуляція

RXD – Received Data

ЦАП - Цифро-аналоговий перетворювач

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						7
Зм.	Арк	№ докум	Підпис	Дата		

ВСТУП

В останні роки кількість користувачів мережі інтернет зростає, охоплюючи все більше сфер нашого життя. Алгоритми глобальних компаній підлаштовуються під власників пристроїв, таким чином, що пропонують рекламу саме під смак користувачів, і контент, який підходить саме для них, змушуючи витратити більше часу. Телефони, значною мірою використовуються, як месенджери, а не за першочерговим призначенням. Перевагою використання радіо є те, що для нього не треба мережа інтернет, недоліком, що можна слухати, обмежену кількість станцій, в залежності від розташування приймача. Для використання інтернет-радіо треба, телефон або комп'ютер, та інтернет, перевагою, є те, що можна знайти будь-яке джерело яке дає аудіо-потік, та географічне розташування не залежить на це. Більшість радіо станцій ведуть мовлення через інтернет теж, для більшого охоплення аудиторії слухачів, та для резерву, щоб продовжувати мовлення, у разі виходу з ладу радіовежі. Так, як інтернет поширюється, набувають популярності IoT пристрої, тому курсовий проект присвячений проектуванню додатку для приймача Інтернет-радіо на базі платформи Arduino.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						8
Зм.	Арк	№ докум	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ТЕХНІЧНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ ПРИЙМАЧА

1.1. Аналіз апаратного забезпечення та роботи з ним

Для розробки приймача інтернет радіо використовується апаратна обчислювальна платформа Arduino. Для роботи з нею треба мати відповідне апаратне забезпечення, в основі якого мікроконтролер, який містить в собі завантажувач, що дозволяє не використовувати зовнішніх програматорів. Архітектура плат дозволяє, використовувати інші пристрої (датчики, світлодіоди, кнопки), або плати розширення, за допомогою цифрових та аналогових входів/виходів[1]. Також за допомогою макетних плат, зникла необхідність паяння дротів до виходів плати[2]. Так, як в цьому проекті треба буде передавати аудіо до іншого пристрою, це теж буде використовуватися. Серед переваг можна відмітити ціну, так як плати не є дорогими, порівнюючи з аналогами; та відкритість апаратного забезпечення, схеми якого розповсюджуються під ліцензією Creative Commons[3]. Досвідчені користувачі можуть самостійно розширювати під себе залізо та ПЗ. Таким чином траплялися випадки, коли після випуску моделі плати, наступні версії вдосконалювалися, після впливу спільноти. Для того, щоб мікроконтролер мав інструкцію роботи, йому необхідно передати Arduino команди, для цього його треба запрограмувати.

Для програмування мікроконтролеру було встановлене середовище розробки «Arduino IDE», в якому є редактор коду(програми пишуться за допомогою мови C або C++), компілятор та модуль передачі прошивки в плату. Також має можливість додавати завантажені самостійно бібліотеки, або користуючись обмеженим вибором у самому середовищі[4]. Для створення програми необхідно написати функцію Setup, яка після прошивання виконається лише раз, при ввімкненні пристрою; та функцію Loop, яка буде виконуватися циклічно, до вимкнення живлення[5].

1.2. Огляд протоколів IoT-пристроїв

Інтернет пристрої можуть надсилати та відправляти дані, зазвичай це інформація від датчиків до сервера. Так, як вони не обмінюються великими об'ємами даних, вони не потребують високої швидкості, також в них доволі обмежена потужність. Тому для їх роботи необхідні полегшені протоколи, які будуть вибагливі до енергозатрат, дозволять обмінюватися інформацією з хмарними сховищами. Для цього є декілька протоколів

MQTT - це мережевий протокол, який працює за принципом видавець-підписник, який здійснює обмін повідомлень між пристроями, з низьким навантаженням на канал зв'язку. Цей протокол використовують у розробці розумного дому, з використанням хмарних сервісів. Видавець публікує повідомлення за темою, а підписник підписується на тему та отримує ці повідомлення. Особливістю є те, що пристроїв (видавців), може бути багато, та треба підписник, якщо в межах локальної мережі, то сервер.

Інший протокол, який ще часто використовується, CoAP, який використовує модель запит-відповідь. Та на відміну від MQTT, це протокол з типом «один до одного», для зв'язку між клієнтом і сервером. Схожий на HTTP, теж має REST архітектуру, та відмінністю є те, що заголовки, методи та коди стану у двійковому коді, для того, щоб помістити більшу кількість пристроїв у мережі.

Також можна використовувати розповсюджений протокол HTTP, його особливостями є - передача команд за допомогою URI, нескладна реалізація, підходить для застосування у локальній мережі, та розгортанні сервера на мікроконтролері

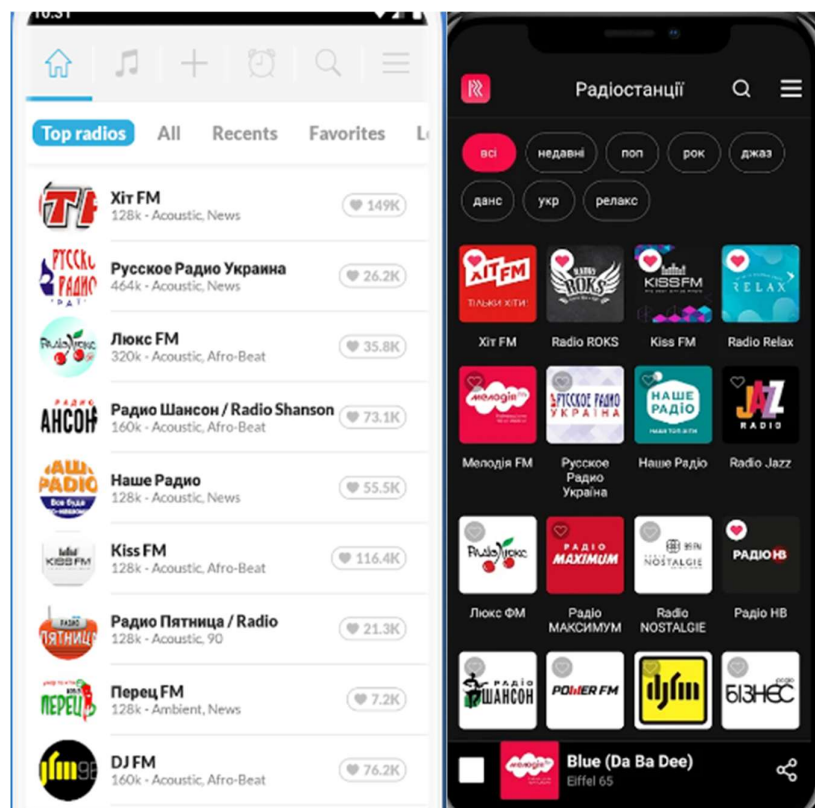
					КІ.КР.Б -123 – 09 - ПЗ	Арк
						10
Зм.	Арк	№ докум	Підпис	Дата		

1.3. Аналіз сервісів, які дозволяють прослуховувати інтернет радіо

Для того, щоб дістати веб-адреси інтернет радіо станцій, спочатку була ідея зробити, як в існуючих додатках, які працюють зі стандартом DLNA, який дозволяє обмінюватися медіа-контентом у локальній мережі, та відтворювати його онлайн. Суть роботи в цих додатка яких полягає в тому, що в них є встановлений веб-переглядач, за допомогою якого, знаходиться необхідний ресурс, та запускається медіа-контент, в той час відбувається фільтрація запитів, і залишається обрати медіа-контент для програвання. Така особлива реалізація здобуття посилання на самий контент, зроблена у зв'язку з рекламою, так як на багатьох веб сторінках для перегляду відеопотоку потрібно її подивитися, інакше бажаного медіа контенту не отримати, посилання теж генерується на деякий час теж з цих причин, тому збережене раніше посилання може не працювати.

Звісно для отримання потоку аудіо такого не має, тому можна просто фільтрувати вхідний трафік, на отримання посилання на аудіо контент, та зберегти його для подальшого використання.

Якщо розглядати додатки саме для прослуховування радіо на смартфоні, то в їх інтерфейс дає можливість прослуховувати деяку кількість радіо станцій, які для актуальності можуть оновлюватися розробниками. В обох подібний інтерфейс. Є можливість переглянути останні прослухані станції, список збережених, локальних(для підбору, за допомогою засобів GPS). Також здійснювати пошук за іменем.



Рисунки 1.1 а, б) – Сервіси для прослуховування інтернет радіо

1.4. Аналіз засобів для проектування додатку

React —JavaScript бібліотека, призначена для створення інтерфейсів користувача, розроблена компанією Facebook. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду — “компонентів”, що також дозволяє їх повторно використовувати[6]. За допомогою такого підходу код стає зрозумілішим. Компоненти зазвичай написані на JSX синтаксисі, який компілюється у виклики методів бібліотеки React, також можна писати і на чистому JavaScript. Синтаксис може нагадувати HTML, так як компоненти пишуться схоже, і атрибути передаються аналогічно, та є синтаксичні особливості, через що його називають розширенням синтаксису для JS.

Redux—бібліотека, рішення архітектури Flux, призначена для управління станом всього додатку[7]. Її використання дозволить керувати станом додатку набагато зручніше, не передавати значення по дереву компонентів. Сам по собі працює синхронно, тому треба використовувати middleware – це проміжне середовище між запитом і тим де цей запит виконався, перед тим, як його опрацювати, відповідь потрапить до цього середовища, їх може бути скільки завгодно, вони виконуватимуться послідовно, прикладом може бути перевірка токена аутентифікації, або ролей.

Для асинхронності між React-компонентами, та станом додатку, використаю бібліотеку RxJS. В ній вона досягається за допомогою використання зв'язки шаблонів Observable, Iterator та функціональних можливостей JavaScript та RxJS для роботи з колекціями[8]. Вона дозволяє організує роботу з подіями і асинхронним кодом, пишучи код декларативно, тобто описуючи результат, який необхідно отримати, замість описання послідовності отримання цього результату[9].

Механізм роботи з бібліотекою не новий, взятий з шаблону «Спостерігач», де один об'єкт, вмiє сповiщати iншi, тi якi слiдкують з ним, якi пiдписанi на його подiї, про змiни його внутрiшнього стану. Його було опублiковано в книзi «Design Patterns: Elements of Reusable Object-Oriented Software» 1994 року.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						13
Зм.	Арк	№ докум	Підпис	Дата		

Основні поняття управління асинхронними подіями в RxJS включають: Observable(або ще «Потік»): головний концепт, RxJS, об'єкт, який виробляє дані.

Observer: Надає колекцію зворотних викликів, які відстежують значення, передане до Observable.

Subscription: Представляє виконуваний файл Observable, в основному використовується для скасування виконання.

Operators: Це чиста функція, яка дозволяє обробляти операції збору та функціонального стилю програмування, такі як map, filter, concat, flatMap тощо.

Subject: Еквівалентний EventEmitter, його єдиний метод полягає в тому, щоб передати значення або подію кільком спостерігачам.

RxJS оперує об'єктами Observable, які існують у кількох різновидах (Subject, Scheduler) та реалізують принцип push-систем передачі даних від "постачальника" до "споживача". Крім push-систем виділяють ще й pull-системи передачі. pull-системах "споживач" вирішує, коли отримати дані від "постачальника". Будь-яка функція JavaScript – яскравий приклад такої системи. Код виклику функції – "споживач", сама функція – "постачальник". У випадку RxJS Observable саме відправник вирішує, коли відправити дані одержувачу. До цього типу push-систем належать і об'єкти Promise, які надають дані зареєстрованому callback – функціям та ініціюють їх виклик. Порівняємо існуючі pull/push-системи у мові програмування JavaScript, наведено у рис 2.2.1

	Single	Multiple
Pull	Function	Iterator
Push	Promise	Observable

Рисунок 1.2 Порівняння pull/push-систем

Висновок до розділу 1

У розділі було розглянуто роботу з платформою Arduino, принципи написання вбудованих програм у середовищі розробки Arduino IDE, призначеної для написання програм для апаратного забезпечення, та завантаження їх на плату. Проаналізовано рішення додатків, для прослуховування інтернет радіостанцій. Розглянути бібліотеки для архітектури додатку, які дозволяють будувати інтерфейс, керувати станом додатку та асинхронними подіями. Також використання шаблонів, які реалізовано у цих бібліотеках.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						15
Зм.	Арк	№ докум	Підпис	Дата		

РОЗДІЛ 2. РОЗРОБКА КЛІЄНТСЬКОГО ДОДАТКУ

2.1. Аналіз реалізації для діставання аудіо потоку

Для можливої реалізації цього було досліджено можливості React Native: WebView можна використати для роботи веб-переглядача, в рамках додатку, тільки, на відміну від вище вказаного прикладу необхідно встановити фільтрування на аудіо контент. Щоб можна було відстежувати запити, слід використати властивість injectedJavaScript, куди додати код для відстеження. Інший варіант витягування вебданих (англ. web scraping), який полягає у створенні структурованих даних з веб сторінки[10]. У даному випадку можна було б перевіряти каталоги інтернет радіостанцій, на наявність посилань з аудіо поточним контентом, за допомогою регулярних виразів.

Та є більш доречний варіант, який не буде настільки вибагливим до ресурсів, скористатися існуючим прикладним програмним інтерфейсом(API), для запиту посилань. Розробник може надавати можливості уточнювати запитовані дані за допомогою параметрів до URL, та дозволяти фільтрувати їх.

Знайшовши, таке джерело даних, було проаналізовано майбутній інтерфейс для додатку, в даному випадку можна завантажити всі станції за раз, та не має потреба зберігати всі дані, якщо розробник надає можливість запитувати дані більш точніше за допомогою параметрів до URL. За допомогою цього можна запитувати посилання за мовою транслявання контенту або за країнами.

2.2. Проектування сторінки зі списком доступних потоків

У зв'язку з розвитком бібліотеки React, з версії 16.8 функціональні компоненти стали еквівалентними класовим, шляхом введення хуків. До того ж, доступ до деяких аспектів був виключно у класах. Після нововведень компоненти, стали займати менше місця, та на розуміння коду стало витрачатися менше часу. Хуки розширили можливості функціональних компонентів, вони дозволяють використання стану, побічних ефектів[11], створення користувачьких хуків, для винесення логіки компонента у функції, для подальшого повторно використання.

Тож для зберігання посилань аудіо потоків використаю хук `useState`, щоб його використати, треба розуміти оператор мови JS – деструктуризуюче присвоєння[12]. За допомогою цього оператора можна розпакувати значення з масиву, об'єкта, чи функції в різні змінні (наприклад `[a, b] = [10, 20]`, де `a` буде дорівнювати 10, `b` – 20). Також додам стан для змінної, яка буде відповідати за те, від якого номеру станції, варто їх відобразити.

Лістинг 2.1 Використання хуку стану

```
const [stations, setStations] = useState([])
const [offset, setOffset] = useState(0);
```

Після оголошення стану за допомогою цього хуку, отримую першим значенням поточний стан, другим функцію, що дозволяє його оновити. У конструкторі функції `useState`, передам початкове значення – пустий масив, так, як він буде заповнюватися під час виконання.

Для заповнення масиву скористаюсь ще одним хуком `useEffect` для виконання побічних ефектів, запиту даних після початку роботи програми[13]. Для перетворення відповіді формату JSON з `string` до об'єкта, використаю метод `json()`;

Лістинг 2.2 Запит даних при ініціалізації додатку

```
useEffect(() => {
  async function fetchStations() {
    const response = await fetch(`https://de1.api.radio-
    browser.info/json/stations/bylanguage/ukrainian?limit=10&offset=${offset}`);
    const stationsList = await response.json();
    setStations(stationsList) });
```

Тут виникають проблеми, ця функція буде спрацьовувати після кожної зміни стану, навіть тієї, до якої цей код не має відношення. Таким чином це спричинить проблеми продуктивності, так як стан, який стосується безпосереднього цієї функції може не змінюватися. Тому цей хук має додаткові параметри, те від чого має залежати, чи треба, що б ця функція спрацьовувала, для їх передачі треба додати до другим параметром після анонімною функції, додати масив для змінних, при зміні яких, функція буде виконуватися, в цьому випадку це змінна `offset`, таким чином після її зміни буде відправлятися ще один запит.

Лістинг 2.3 Додавання залежності до хуку `useEffect`

```
...
  setStations(stationsList)
}, [offset]);
```

Також у результаті виконання запиту, до змінної `stationsList`, прийде об'єкт, з багатьма значеннями, які будуть зайві, так як вони використовуватися не будуть та їх не потрібно зберігати до стану, залишивши тільки ім'я станції та посилання на потік. Було досліджено влаштовані методи об'єкту масив у мові програмі Javascript, для опрацювання отриманої відповіді формату JSON від API, та вирішено, що метод `Array.prototype.map()` підходить для реалізації. В нього необхідно передати функцію, яка повертатиме значення, яке буде трансформувати кожен елемент масиву.

Лістинг 2.4 Трансформація відповіді за допомогою методу `map`

```
setStations(stationsList.map(i => {
  const { name, url } = i;
  return { name, url }
})));
```

Або скоротивши до:

Лістинг 2.5 Трансформація відповіді за допомогою методу `map`, з скороченим записом анонімною функції

```
stationsList.map(({name, url}) => ({name, url }));
```

Для показування індикатора перевірятиму довжину масиву, якщо нуль, скористаюся компонентом `ActivityIndicator[14]`, для інформування користувача про завантаження даних; якщо не нуль, то показуватиму контент за

допомогою компонента, відповідального за відображення списку – FlatList. Його властивість data, якого приймає масив об'єктів, у даному випадку посилання на аудіо потоки, та імена станцій. Також необхідно, щоб кожен елемент масиву мав властивість key, для коректного рендерингу, та при зміні списку React ефективніше знаходитиме конкретні елементи, в даному випадку буде id, який прийде з серверу[15]. Для контейнеру списку є властивість для вказування стилів, containerViewStyle, за допомогою якої напишу стилі для рівномірного розподілу елементів. Для вказування яким чином треба огортати кожен елемент колекції, треба скористатися властивістю renderItem, яка приймає функцію зворотного виклику[16], для відображення використаю Button, обробником подій при натисканні на який, буде відображення посилання, для перевірки, чи посилання прийшло правильно. Для реалізації цього скористаюся класом Alert бібліотеки React-Native, що являє клас відповідальний за модальне вікно, за допомогою його функції alert, зможемо побачити те, що передаємо[17], у даному випадку передам посилання на аудіо.

Лістинг 2.6 Відображення кнопок з посиланнями

```
stations.length == 0 ? <ActivityIndicator size="large" style={{
  justifyContent: "center", flexDirection: "row", flex: 1 }} /> : (
  <View style={{ flex: 1, padding: 25 }}>
    <FlatList
      data={stations}
      containerViewStyle={{ flexGrow: 1, justifyContent:
'space-around' }}
      renderItem={({ item }) => (
        <Button onPress={() => Alert.alert(item.url)} key={item.id}
title={item.name}></Button>)} /> </View >
```

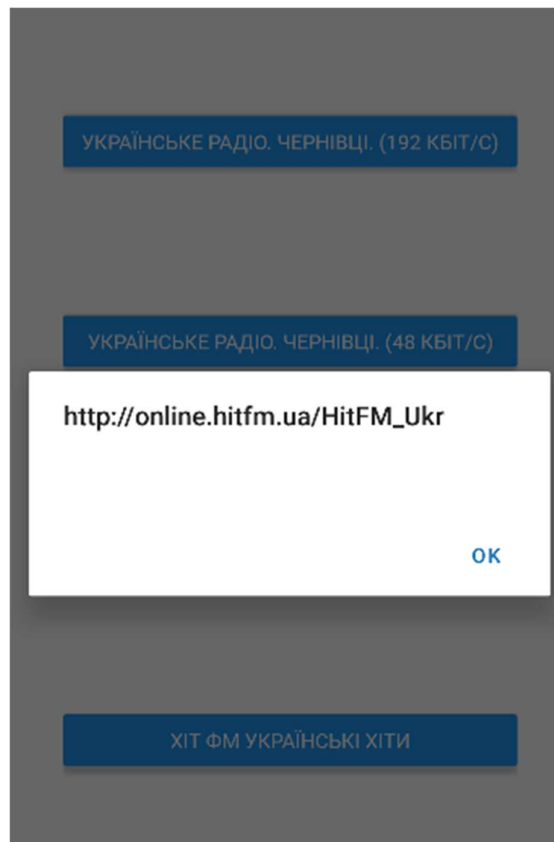


Рисунок 2.1 Відображення списку станцій, та сповіщення після кліку з посиланням

Для того, щоб мати можливість перегляду інших станцій, додаю кнопки для контролю змінної offset. Додаю ще дві кнопки знизу, для навігації, та змінюватиму стан змінної offset.

Лістинг 2.7 Кнопки навігації

```
<View style={{ flex: 1 }}>
  <Button onPress={() => { setOffset(offset - 10) }} title="<-Previous" ></Button>
</View>
<View style={{ flex: 1 }}>
  <Button onPress={() => { setOffset(offset + 10)}} title="Next->"></Button>
</View>
```

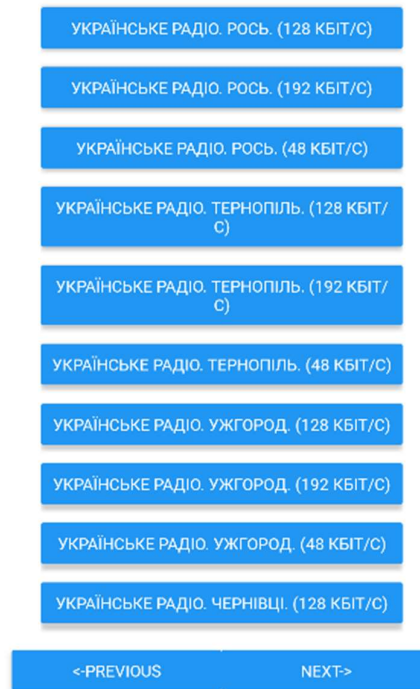


Рисунок 2.2 Відображення списку станцій, та кнопок навігації

Треба передбачити, можливі помилки, тому показуватиму першу кнопку, тільки, якщо є можливість повернутися назад, якщо offset буде більше за нуль. Також варто урахувати, що список може закінчитися, тому додам умову та запитаю, чи бажає користувач почати спочатку, якщо ні показуватиметься цей же список. Заберу можливість натискати кнопку переходу на попередню сторінку на першій сторінці, тобто, можна буде переходити тільки на наступну, для цього скористаюся умовним вимальовуванням[18], перевірятиму змінну offset.

Лістинг 2.8 Показ сповіщення про закінчення списку

```
const showConfirmDialog = () => {
  return Alert.alert(
    "Thats the final page",
    `Would you like to get to the first?`,
    [{ text: "Yes",
      onPress: () => {
        setOffset(0);
      } },
      { text: "No", }
    ]
  );
};

useEffect(() => {
  async function fetchStations() {
    const stationsList = await response.json();
```

```

        if (offset > 0 && stationsList.length == 0)
            setStations(({name, url}) => {return {name, url }})
        else
            showConfirmDialog()
    } }, [offset]);
    ...

    {(offset > 0) && <View style={{ flex: 1 }}>
        <Button onPress={() => { setOffset(offset - 10) }}
            title="<-Previous"
            disabled={disabledChangingList}
        ></Button>
    </View>
    <View style={{ flex: 1 }}>
        <Button onPress={() => { (offset % 10 == stations.length - 10) ?
setOffset(offset + 10) : showConfirmDialog() }}
            title="Next->"
            disabled={disabledChangingList}
        ></Button>
    </View>

```

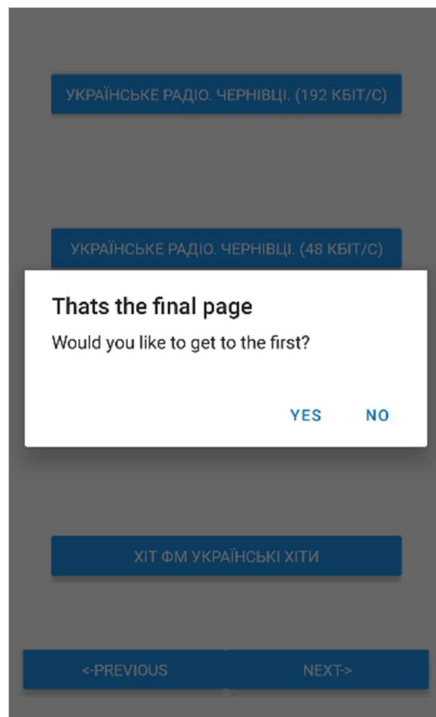


Рисунок 2.3 Відображення сповіщення про показ станцій спочатку

Додам затримку, щоб запит встиг дійти, і показатися результат, та в цей час вимикатиму кнопки, щоб користувач чітко бачив, що результат обробляється, і не міг натискати до того, як результат прийшов. Додам для цього ще одну змінну стану.

Лістинг 2.9 Показ сповіщення про закінчення списку

```

const [disabledChangingList, setDisabledChangingList] = useState(false);
...
useEffect(() => {
    setDisabledChangingList(true);

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		22



Рисунки 2.4 а, б) Відображення анімації оновлення, та вимкнених кнопок навігації, при очікуванні даних

Перед передачею пристрою Arduino, посилання треба перевірити, тому необхідно додати функцію відкриття посилання у браузері. Для цього зміню обробник кліку, щоб можна було бачити панель «Open with», та відкрити посилання. Для цього використаю клас Linking[18], бібліотеки React Native, який звертається до Linking API, пристрою. Він має метод openURL, який приймає

посилання. Тому заміню обробник функції натискання на кнопку з `Alert.alert(item.url)` на `Linking.openURL(item.url)`.

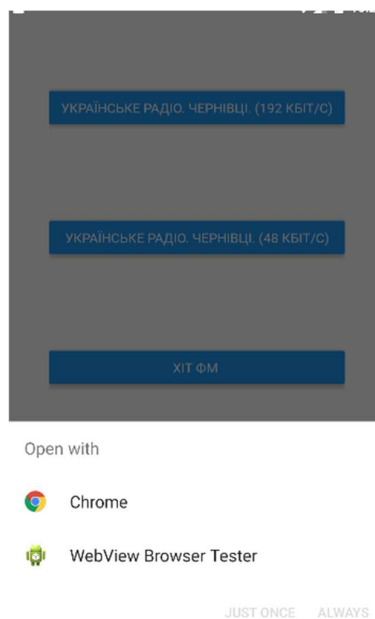


Рисунок 2.5 Панель «Open with», після натискання на кнопку

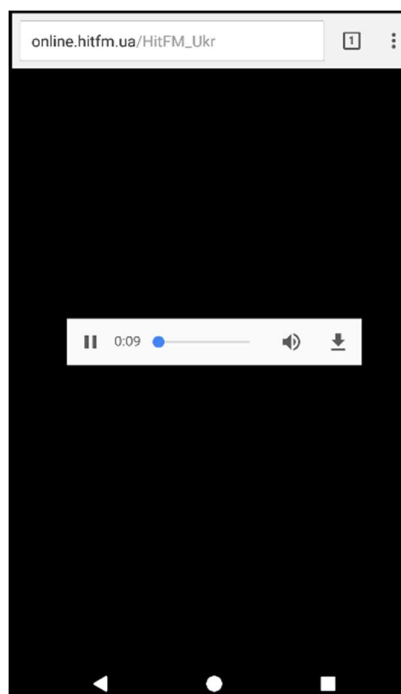


Рисунок 2.6 Відображення програвача у веб браузері, після переходу

Винесу кнопки до окремого компонента, таким чином це буде дочірній елемент, який можна винести в окремий файл, куди теж перенесу метод

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		24

showConfirmDialog. Та передаватиму змінну стану offset, та функцію setOffset як властивість до компонента. Таким чином в React-архітектурі досягається композиція[19] додатку, використовуючи тип відносин Has-a[20], тобто панель з кнопками має панель навігації.

Лістинг 2.10 Використання компонента OffsetButtons після виносу в окремий файл

```
import OffsetButtons from './OffsetButtons';
...
<OffsetButtons offset={offset} setOffset={setOffset} />
...
```

Лістинг 2.11 Компонент OffsetButtons окремому файлі

```
export default OffsetButtons = (offset, setOffset) => (
  <View style={{ bottom: 10, flexDirection: 'row', justifyContent: "space-
around", alignItems: "flex-end" }}>
    {(offset > 0) &&
      <View style={{ flex: 1 }}>
        <Button onPress={() => setOffset(offset - 10) }
          title="<-Previous"
        ></Button>
      </View>
    <View style={{ flex: 1 }}>
      <Button onPress={() => { setOffset(offset + 10) }}
        title="Next->"
      ></Button>
    </View >
  </View > )
```

2.3. Підключення бібліотеки Redux

Для того, щоб уникнути використання у функціональних компонентах хуку `useState` (або ще є можливість керування об'єктом `this.state` у класових компонентах, в проєкті використовують виключно функціональні, для ергономічності коду), та передавання стану до дочірніх компонентів, використаю рішення архітектури Flux, бібліотеку Redux, для керування стану всього додатку. Це дозволить використовувати стан додатку набагато зручніше, не передавати значення по дереву компонентів, а зберігати стан в одному місці – у сховищі. Бібліотека використовує переваги Context з React API, за допомогою якого створюється глобальні змінні, які можуть використовуватися в будь-якому місці в додатку[22]. Для початку реалізації сховища створю файл «`reducer.js`» з функцією, яка повертатиме об'єкт стану[22], створю початковий стан, який буде параметром за замовчуванням до функції. Також скористаюся методом `combineReducers`, для того, щоб можна було створювати окремі функції, для одного стану.

Лістинг 2.12 Сховище керування станом

```
import { combineReducers } from "redux"
import { SET_STATIONS } from './actions';

const initialState = {
  stations: [],
};
function stationsReducer(state = initialState.stations, action) {
  switch (action.type) {
    case SET_STATIONS: {
      return [...state, action.stations]
    }
  }
  return state;
}

export default combineReducers({
  stations: stationsReducer,
});
```

Для того, щоб відстежити, в який саме спосіб треба змінювати стан в функції Reducer, використовується оператор `switch` для властивості `type`. Для генерування нового стану, використаю попередній стан за допомогою оператора розширення[24].

Тепер можна створити сховище за допомогою функції createStore, куди й передам функцію reducer.

Лістинг 2.13 Підключення сховища до додатку

```
import {createStore } from 'redux'
import rootReducer from './app/reducer';
...
const store = createStore(rootReducer)
return (
  <Provider store={store} >
    <RadioStationsApp />
  </Provider>);
```

Для доступу до сховища, використаємо хук суть використання якого, така ж, що й useState, використовуючи при цьому модель Redux.

Ця функція приймає функцію reducer, куди передається поточний стан, та об'єкт Action(об'єкт, що має дві властивості: type, що може бути, будь яким рядком, і необов'язкову властивість payload – будь якого типу), та початкове значення.

Вона повертає масив, подібно функції useState(де стан оновлюється на-пряму), перше значення якого – стан, друга – функція, замість функції, яка на-пряму оновлює стан, отримується функція dispatch, в яку передається Action, що прямує у функцію Reducer, де в свою чергу вирішується, як саме обчислити наступний стан[24].

Замість використання функції useReducer, існують альтернативні хуки: useSelector(для витягнення специфічних даних зі стану), та useDispatch(для функції в яку передати об'єкт Action). Перенесу об'єкти, які їх зроблять (Action creators[25]) в окремий файл:

Лістинг 2.14 Action creators

```
export const SET_STATIONS = 'SET_STATIONS';
export const setStations = (stations) => ({
  type: SET_STATIONS,
  payload: { stations },
});
const GET_STATIONS = 'GET_STATIONS';

export function getStations() {
  return {
    type: GET_STATIONS
  }}
}
```

Лістинг 2.15 Робота з сховищем

```
import { useDispatch, useSelector } from 'react-redux';
import { getStations, setStations } from './actions';

const { stations } = useSelector((state) => state.stations);

useEffect(() => {
  async function fetchStations() {
    const response = await fetch(`https://de1.api.radio-
browser.info/json/stations/bylanguage/ukrainian?limit=10&offset=${offset}`);
    const stationsList = await response.json();
    dispatch(setStations(stationsList));
  }, [offset]);
```

Також перенесу змінну offset до reducer.

Лістинг 2.16 Додавання змінної offset до функції reducer

```
const initialState = {
  offset: 0,
  stations: [],
}
export default combineReducers({
  offset: offsetReducer,
  stations: stationsReducer,
});
function offsetReducer(state = initialState.offset, action) {
  switch (action.type) {
    case INCREASE_OFFSET:
      return state + 10;
    case DECREASE_OFFSET:
      return state - 10;
    case ' SET_DEFAULT_OFFSET':
      return initialState.offset; }
  return state;
}...
```

Компонент OffsetButtons зміниться після перенесення:

Лістинг 2.17 Робота з станом сховища у файлі OffsetButtons

```
...
const offset = useSelector(state => state.offset);
const dispatch = useDispatch();
const showConfirmDialog = () => {
  return Alert.alert(
    "Thats the final page",
    `Would you like to get to the first?`,
    [
      {
        text: "Yes",
        onPress: () => {
          dispatch(setShowConfirmationDialog(false));
          dispatch(setDefaultOffset());
        },
      },
      {
        text: "No",
        onPress: () => {
          dispatch(setShowConfirmationDialog(false));
        }
      }
    ]
  );
}
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						28
Зм.	Арк	№ докум	Підпис	Дата		

```

        },
    ]
    );
};

return (
    <View style={{ bottom: 10, flexDirection: 'row', justifyContent: "space-
around", alignItems: "flex-end" }}> {(offset > 0) &&
        <View style={{ flex: 1 }}>
        <Button onPress={() => dispatch(decreaseOffset)} title="<-Previous"></Button>
        </View>
        <View style={{ flex: 1 }}>
        <Button onPress={() => { dispatch(increaseOffset) }}
            title="Next->"
            ></Button>
        </View >
        <OffsetButtons offset={offset}/>
    </View > );

```

Тож тепер немає потреби передавати значення змінної до файлу навігаційних кнопок, тому використання буде:

Лістинг 2.18 Вимальовування компонента OffsetButtons

```
<OffsetButtons />
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		29

2.4. Підключення бібліотеки RxJS

Сама бібліотека Redux працює синхронно, тому для додавання асинхронності треба застосування middleware, одним з яких є RxJS.

За допомогою Епіс функцій[26], можна підписатися на потоки Action та State, таким чином в цих функціях будуть отримуватися ці потоки, та можна контролювати стан. В них може бути багато observable конструкцій, та в кінцевій має повернутися Action. На кожен відданий Action, буде застосовано функцію store.dispatch(). Тож створюю middleware до сховища, використавши функцію після створення сховища, та передачі туди middleware, викличу функцію epicMiddleware.run, куди передам функцію з створеного файлу epic.js, де буде сама Епіс функція:

Лістинг 2.19 Підключення epicMiddleware до додатку

```
import { applyMiddleware, createStore } from 'redux'
import { createEpicMiddleware } from 'redux-observable';
import rootEpic from './epic';

const App = () => {
  const epicMiddleware = createEpicMiddleware();
  const store = createStore(rootReducer,
    applyMiddleware(epicMiddleware)
  );
  epicMiddleware.run(rootEpic);
  return (
    <Provider store={store} >
      <RadioStationsApp />
    </Provider>
  );
};
```

Епіс функція:

Лістинг 2.20 Епіс функція, яка виконує асинхронний запит

```
const getStationsEpic = (action$, state$) => action$.pipe(
  ofType('GET_STATIONS'),
  mergeMap(() =>
    ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/bylanguage/${state$.value.stationLanguage}?limit=${state$.value.limit}&offset=${state$.value.offset}`)
  ).pipe(
    map(response => setStations(
      response.map(({ name, url_resolved }) => ({ name, url_resolved}))
    ))
  );
);

export default getStationsEpic;
```

Тоді у функції useEffect, видалю запит, замість якого викличу функцію dispatch в яку передам Action creator, для потрапляння у функцію Епіс.

Лістинг 2.21 Виклик функції `dispatch`, після якого `middleware` перехопить потік у еріс функції, де надішлеться запит

```
useEffect(() => {
    dispatch(getStations());
}, [offset]);
```

Плюсом даного підходу є можливість зручного трансформування даних, різними операторами бібліотеки, тому скориставшись ними, зроблю запит всіх доступних станцій для конкретної мови транслювання контенту, а користуючись навігацією здійснюватиму перехід по наявних даних, замість того, щоб надсилати запит, кожного разу. Для реалізації цього скористаюся оператором `bufferCount`, який бере значення з потоку, та розбиває їх, в залежності від значення, яке передається до нього[27]. В даному випадку передаватиму туди значення стану додатку `limit`, яке відповідає, за кількість станцій на сторінці.

Після трансформування застосую функцію `mergeAll()`, для поєднання до одного потоку, після цього `toArray()`, для конвертування потоку значень до масиву, і після цього призначу в стан значення відповідно до змінної `offset`. Необхідно при зміні значення `offset`, призначати до стану список, який вже було завантажено, при першому зверненні. Тому створю змінну поза межами функції для зберігання списку. Також додам можливість перегляду всього списку одразу, в самій функції буде перевірка значення `limit`, якщо воно буде дорівнювати нулю, то призначу в стан значення всіх об'єктів станцій.

Лістинг 2.22 Еріс функція для отримання даних про станції

```
let stations = [];
const getStationsEpic = (action$, state$) => action$.pipe(
  ofType('GET_STATIONS'),
  switchMap((a) => {
    if (stations.length == 0 && !!a.payload.limit)
      return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/bylanguage/${state$.value.stationLanguage}`)
      .pipe(
        map(response =>
          from(response.map(({ name, url_resolved }) => ({ name, url_resolved })))
            .pipe( bufferCount(a.payload.limit) )),
        mergeAll(),
        toArray(),
        map((result) => {
          stations = result;
```

```

        return setStations(result[a.payload.offset / 10]);
    })
    )
    else if (a.payload.limit == 0)
        return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/bylan-
guage/${state$.value.stationLanguage}`)
        .pipe(
            map(response => (setStations(response)))
        )

    if (stations.length > (a.payload.offset / 10))
        return of(setStations(stations[a.payload.offset / 10]));

    return of(setShowConfirmationDialog(true));
}
),
);

```

При вимальовуванні компонента кнопок додаю умову, при якій вони будуть показуватися тільки, якщо значення `limit` позитивне.

Лістинг 2.23 Вимальовування компонента `OffsetButtons`, якщо присутнє значення `limit`

```
{!!limit && <OffsetButtons />}
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		32

2.5. Проектування сторінки з збереженими станціями

Перед створенням самої сторінки розроблю можливість додавання станцій, з існуючої сторінки. Для цього створю модальне вікно, яке буде відкриватися після натискання на кнопку, в якому буде можливість обрати між додаванням до збережених, відкриттям у браузері, програванням на пристрої. Для цього створю окремих компонент, який використаю на сторінці з станціями. Вікно буде відкриватися після натискання на кнопку, в компонент треба буде передавати дані про натиснуте значення, щоб відкривати те посилання, яка зберігалася в тій кнопці. Тому зміню обробник події натискання на кнопку, та зберігатиму дані вибраної станції у компоненті, після чого передаватиму їх у дочірній компонент, також відбудеться виклик функції відповідальної за відображення модульного вікна, її теж треба передавати до дочірнього компонента, щоб опрацювати вибір, або закриття вікна.

Лістинг 2.24 Вимальовування компонента модульного вікна

```
const [modalStation, setModalStation] = useState();
const [isModalVisible, setIsModalVisible] = useState(false);
const handleModal = () => setIsModalVisible(() => !isModalVisible);
...
renderItem={({ item }) => (
  <Button onPress={(e) => { setModalStation(item); handleModal(); }} key={item.id}
    title={item.name}></Button>
)
...
{isModalVisible && <ChooseStationModal modalStation={modalStation}
  isModalVisible={isModalVisible} handleModal={() => handleModal()}>}
```

Для створення модульного вікна скористаюся бібліотекою «react-native-modal», компонентом Modal, в який передам в значення властивості isVisible значення, передане батьківським компонентом isModalVisible. В стилі зазначу значення прозорості 1, за допомогою якого фон не матиме кольору, та буде затемнений. В середині якого буде компонент View, який займатиме все місце, а його вміст буде розташовуватися в кінці за допомогою властивості justifyContent, та значення 'flex-end'. Це зроблено, для того, щоб можна було натиснути над елементами, таким чином закривши вікно. Для цього огорну це компонентом TouchableWithoutFeedback[28], та передам в onPressOut, виклик функції handleModal переданої зверху.

Додам в вікно кнопки, для подальших дій. Обробник кліку кнопки, яка відповідає за відкриття у браузері зроблю таким же, як був до цього, додавши функцію для закриття вікна.

Лістинг 2.25 Кнопка з відкриттям посилання у браузері

```
<Button title="Open with" onPress={() => {handleModal(); return  
Linking.openURL(modalStation.url_resolved);}} />
```

Далі додам кнопку для додавання станції до збережених, в ній теж буде функція для закриття вікна, та передача інформації про станцію до Epic функцію, в якій буде відбуватися зберігання.

Для зберігання використаю полегшену реляційну базу даних SQLite, через бібліотеку «expo-sqlite». Для створення бази даних треба вказати її ім'я в конструкторі функції SQLite.openDatabase(), далі за допомогою цього об'єкта викликається функція transaction, куди передається анонімна функція, в яку передається об'єкт транзакції, та в нього є функція executeSql. Її використання полягає у написанні запитів напрямку рядком. Тому створю таблицю favoriteStations, з полями для імен, та посилань, за допомогою мови визначення даних[29], таблиця створюватиметься у тому разі, якщо її ще не створено, таким чином, якщо згенерувати додаток, то дані будуть збережені після виходу з додатку.

Лістинг 2.26 Створення бази даних та таблиці в ній для зберігання станцій

```
const db = SQLite.openDatabase('db.db');  
const addToFavoriteEpic = (action$, state$) => action$.pipe(  
  ofType('ADD_FAVORITE'),  
  switchMap((f) => {  
    db.transaction(txn => {  
      txn.executeSql(  
        `CREATE TABLE IF NOT EXISTS favoriteStations (id INTEGER PRIMARY KEY  
        AUTOINCREMENT, name VARCHAR(20), url VARCHAR(50))`, [],  
        () => console.log("table created successfully"),  
        error => console.log("error on creating table " + error.message)),
```

Для вставки даних в таблицю в цій же функції скористаюся мовою маніпулювання даних[30].

Лістинг 2.27 Вставка даних до таблиці

```
txn.executeSql(`INSERT INTO favoriteStations (name, url) VALUES (?, ?)`, [f.name,
f.url_resolved],
  () => console.log(`${f.name} added successfully to favoriteStations`),
  error => console.log("error on adding favoriteStations " + error.message));
  return of(addFavoriteStation(f.name, f.url_resolved));
}), );
```

Додам в функцію reducer масив, для зберігання цих записів, щоб повернути в Еріс функції об'єкт Action creator, для оновлення їх в стані.

Лістинг 2.28 Додавання стану для збережених станцій

```
const initialState = {
  ...
  favoriteStations: [],
}
export default combineReducers({
  ...
  favoriteStations: favoriteStationsReducer }

function favoriteStationsReducer(state = initialState.favoriteStations, action) {
  switch (action.type) {
    case 'ADD_FAV':
      if (!state.favoriteStations)
        return { favoriteStations: [...state, action.payload] }
      return { ...state, favoriteStations: [...state.favoriteStations, action.payload] }
  }
}
```

Та сам Action creator:

Лістинг 2.29 Action creator для додавання станції до стану збережених

```
export const addFavoriteStation = (name, url_resolved) => ({
  type: 'ADD_FAV',
  payload: { name, url_resolved }
});
```

Щоб дати можливість переглядати збережені станції, створю окрему сторінку, для цього створю навігацію між сторінками в додатку, після цього додам окрему сторінку.

Також додам кнопку для програвання на пристрої.

Лістинг 2.30 Кнопка для програвання на пристрої

```
<Button title={`Play ${modalStation.name} on the ESP`}
onPress={() => { handleModal(); return dispatch(setUrl(modalStation.url_resolved)) }} />
```

Далі передам посилання пристрою, так, як відповідь не потрібна нічого не повертатиметься.

Лістинг 2.31 Передача посилання пристрою

```
const setUrlEpic = (action$, state$) => action$.pipe(
  ofType('SET_URL'),
  switchMap((url) => {
    return ajax.get(`http://ztu.local/url?url=${url.url}`).pipe(
      map(res => {
        if (!res.response) {
          throw new Error();
        }
        return of([]);
      }),
      catchError(() => (empty()))
    )
  })
);
```

Також додам у модальному вікні назву станції, щоб користувач був поінформований, яку саме станцію він обрав. Додам стилі до назви станції, щоб вона візуально виділялася серед тексту. Тому це буде шрифт курсив, з нижнім підкресленням такого ж кольору, як і кнопка.

Лістинг 2.32 Компонент модульного вікна

```
export default ChooseStationModal = ({ modalStation, isModalVisible, handleModal }) => {
  const dispatch = useDispatch();

  return (
    <Modal isVisible={isModalVisible} style={{ opacity: 1, margin: 0 }}>
      <TouchableWithoutFeedback
        onPressOut={() => handleModal()}
        activeOpacity={1}
      >
        <View style={{ with: '100%', height: '100%', justifyContent: 'flex-end' }}>
          { /* <Button title={ButtonTitle(modalStation.name)} */ }
          <Text style={{ backgroundColor: 'white', textAlign: 'center' }}>Station name: <Text
style={{
  fontWeight: 'bold',
  textDecorationColor: "rgb(33, 150, 243)",
  textDecorationLine: 'underline', fontStyle: 'italic'
}}>{modalStation.name}</Text></Text>
          <Button title={`Play ${modalStation.name} on the ESP`}
            onPress={() => { handleModal(); return dispatch(setUrl(modalStation.url_resolved)) }} />
          <Button title="Add to favorite" onPress={() => { handleModal(); return
dispatch(addToFavorites(modalStation)); }} />
          <Button title="Open with" onPress={() => { handleModal(); return
Linking.openURL(modalStation.url_resolved); }} />
        </View>
      </TouchableWithoutFeedback>
    </Modal >
  );
};
```

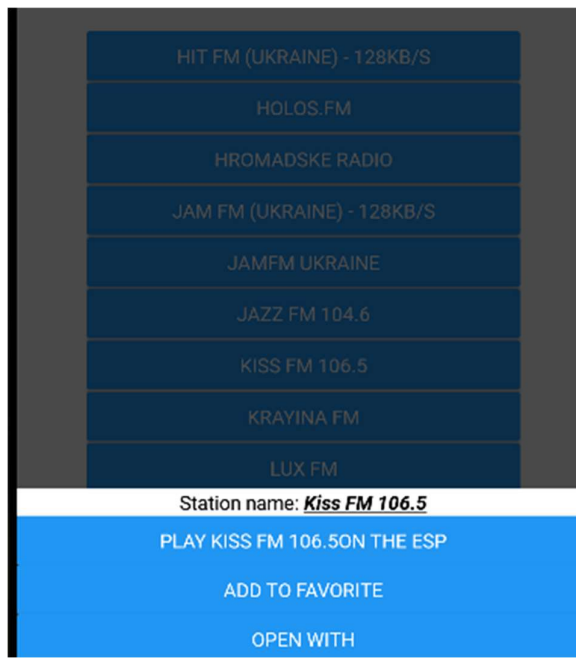


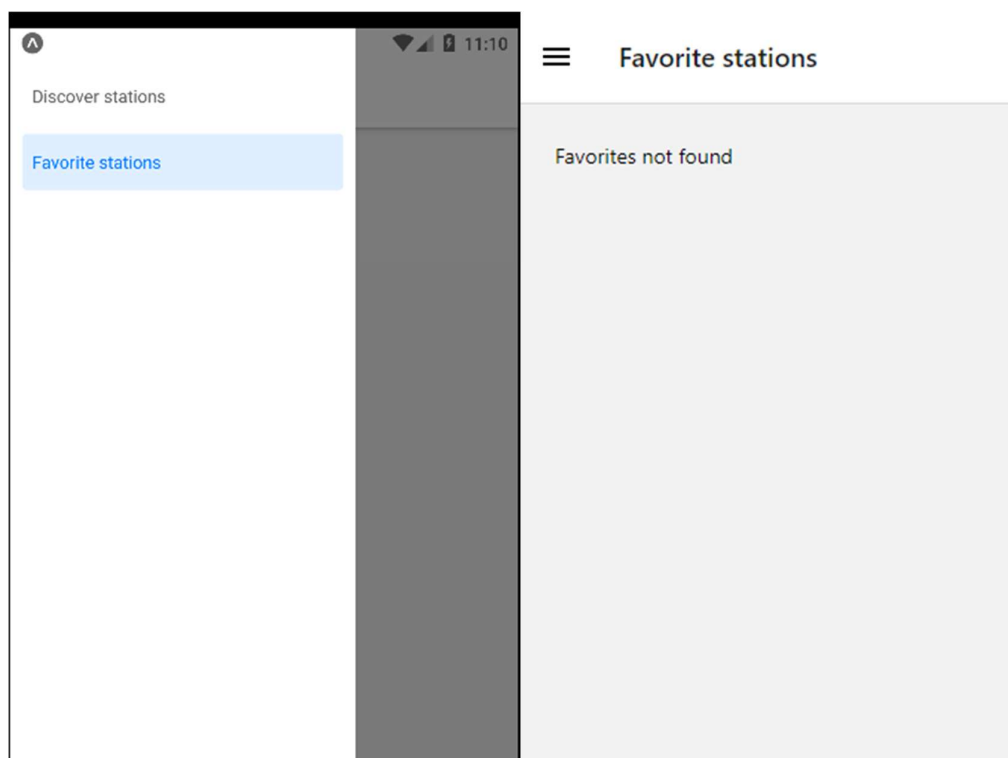
Рисунок 2.7 Модальне вікно після кліку на кнопку

Для створення навігації між сторінками створю бічну панель, за допомогою якої можна буде переходити на інші сторінки. Щоб це реалізувати треба скористатися бібліотекою React Navigation, яка необхідна для маршрутизації та навігації додатку, за допомогою якої можна створити компонент DrawerNavigator [32]. Користаючись ним нову, створю сторінку, додам компонент, у кореневому файлі:

Лістинг 2.33 Використання компоненту DrawerNavigator

```
import { createDrawerNavigator, } from '@react-navigation/drawer';
const Drawer = createDrawerNavigator();
...
<Provider store={store} >
  <Drawer.Navigator>
    <Drawer.Screen name="Discover stations" component={StationsScreen} />
    <Drawer.Screen name="Favorite stations" component={FavoriteScreen} />
  </Drawer.Navigator>
</Provider>
```

Далі створю компонент з назвою FavoriteScreen, на основі існуючої сторінки. Перевірятиму наявність записів, які діставатимуться зі сховища, якщо їх немає показуватиму відповідне повідомлення.



Рисунки 2.8 а) Навігаційне вікно; б) Вікно збережених станцій, за умови відсутності збережених

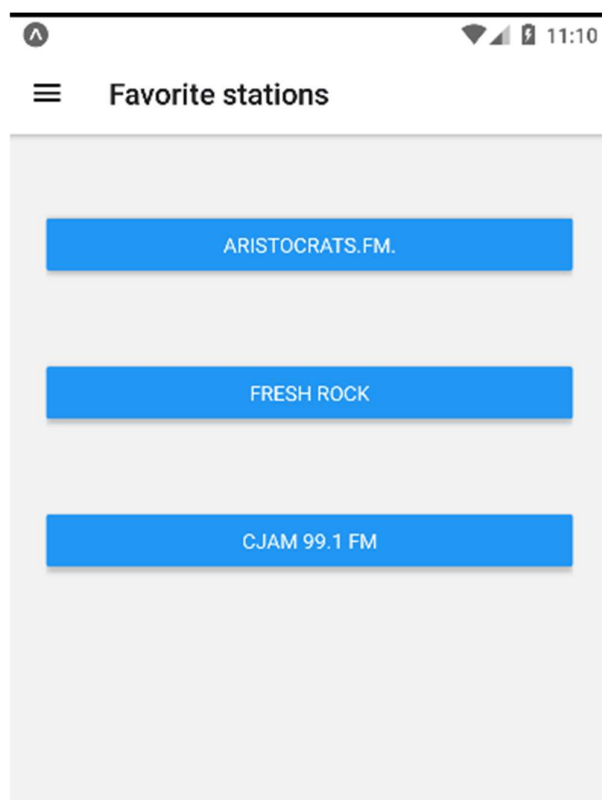


Рисунок 2.9 Вікно збережених станцій, в стані присутності станцій

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		38

Додам обробник події кліку, такий, як на головній сторінці, показувати те ж модальне вікно. Перероблю, що б компонент вікна був універсальний, та підходив як на додавання, так і видалення збереженого об'єкта. Для цього буду передавати компоненту властивість, за допомогою якої, буду розрізняти, та рендерити потрібне.

Лістинг 2.34 Рефакторинг компоненту, для повторного застосування

```
export default ChooseStationModal = ({ modalStation, isModalVisible,
  handleModal, add }) => {
  ... {add ?
    <Button title="Add to favorite" onPress={() => { handleModal(); return
      dispatch(addToFavorites(modalStation)); }} />
    : <Button title="Delete from favorites" onPress={() => { handleModal();
      return dispatch(deleteFromFavorites(modalStation.name)); }} />}
```

В зв'язку з цим додаю властивість до головного компоненту

Лістинг 2.35 Додавання змінної у головному вікні

```
{isModalVisible && <ChooseStationModal modalStation={modalStation} isModalVisible={isModalVisible} handleModal={() => handleModal()} add={true} />}
```

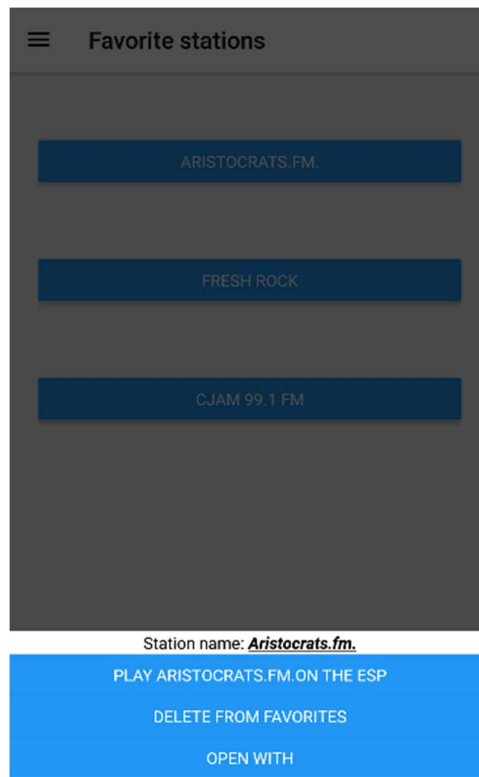


Рисунок 2.10 Застосування того ж компонента модального вікна на сторінці збережених станцій

Додам Epic функцію, де буду робити видалення у базі даних SQLite та додам обробник у сховищі, для видалення кнопки зі сторінки.

Лістинг 2.36 Видалення запису в базі даних, та повернення об'єкта, за допомогою якого відбудеться видалення в сховищі

```
const deleteFavoriteEpic = (action$, state$) => action$.pipe(
  ofType('DELETE_FAVORITE'),
  switchMap((f) => {
    db.transaction(txn => {
      txn.executeSql(
        txn.executeSql(
          `DELETE FROM favoriteStations WHERE name = ?`, [f.name],
        ) => {
          console.log(`${f.name} deleted successfully from favoriteStations`)
          return of(deleteFavorite(f.name));
        },
      ),
      error => console.log("error on deleting favoriteStations " + error.message));
    });
    return of(deleteFavorite(f.name));
  }),
);
```

Видалення зі сховища відбуватиметься за допомогою функції filter[32], в яку передам анонімну функцію, яка проходить по масиву, та перевіряє, чи кожен елемент на задану умову, в даному випадку на рівність з об'єктом, який прийшов.

Лістинг 2.37 Видалення зі сховища

```
function favoriteStationsReducer(state = initialState.favoriteStations, action) {
  ...
  case 'DELETE_FAV':
    return {
      ...state, favoriteStations: state.favoriteStations.filter(item => item.name !==
action.payload.name),
    }
}
```

Додам можливість додавання станції, за допомогою самостійного вписування посилання, для цього створю нове модульне вікно, в якому будуть Input компоненти[33], та Button, після натискання на який відбудеться зберігання імені та посилання радіо станції. Також перевірятиму правильність написаного посилання, тільки після цього відбудеться розблокування кнопки, та користувач матиме можливість натиснути на неї, для цього створю ще одну змінну стану, та передаватиму її до властивості disabled компонента Button. Після натискання відбудеться очищення полів введення тексту, для того, щоб

при введенні параметрів наступної станції, не доводилося їх очищувати клавіатурою. В сховищі використовуватиметься існуючий обробник, так як додавання відбудеться аналогічно з головною сторінкою.

Лістинг 2.38 Компонент модульного вікна, для введення параметрів станції

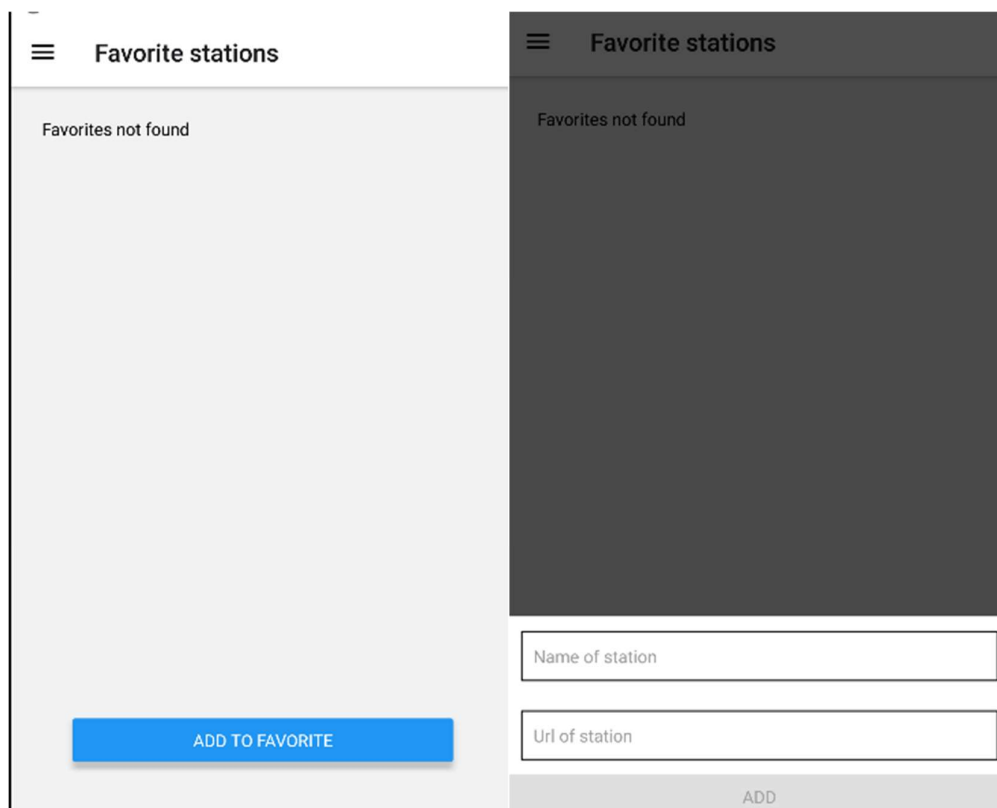
```
export default AddFavorite = ({ isModalVisible, handleModal }) => {
  const dispatch = useDispatch();

  const [name, onChangeName] = useState(null);
  const [url, onChangeUrl] = useState(null);
  const [isDisabled, setIsDisabled] = useState(false);

  const handleChange = (url) => {
    onChangeUrl(url);

    setIsDisabled(url.slice(0, 5) === "http:");
  }
  return (
    <Modal isVisible={isModalVisible} style={{ opacity: 1, margin: 0, width: '100%', bottom:
0, position: 'absolute' }}
      onSwipeComplete={() => handleModal()}
      onBackdropPress={() => handleModal()}
      onRequestClose={() => handleModal()}
    >
      <>
        <View style={{ backgroundColor: 'white', bottom: '0%' }}>
          <TextInput
            style={styles.input}
            onChangeText={onChangeName}
            value={name}
            placeholder="Name of station"
          />
          <TextInput
            style={styles.input}
            onChangeText={(e) => handleChange(e)}
            value={url}
            placeholder="Url of station"
          />
          <Button title={'Add'}
            disabled={!isDisabled}
            onPress={() => { handleModal(); onChangeName(null); onChangeUrl(null);
return dispatch(addFavoriteStation(name, url)); }} />
        </View>
      </>
    </Modal>
  );
}
const styles = StyleSheet.create({
  input: { height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,}, });
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		41



Рисунки 2.11 а) Вікно збережених станцій; б) Модульне вікно для додавання станції

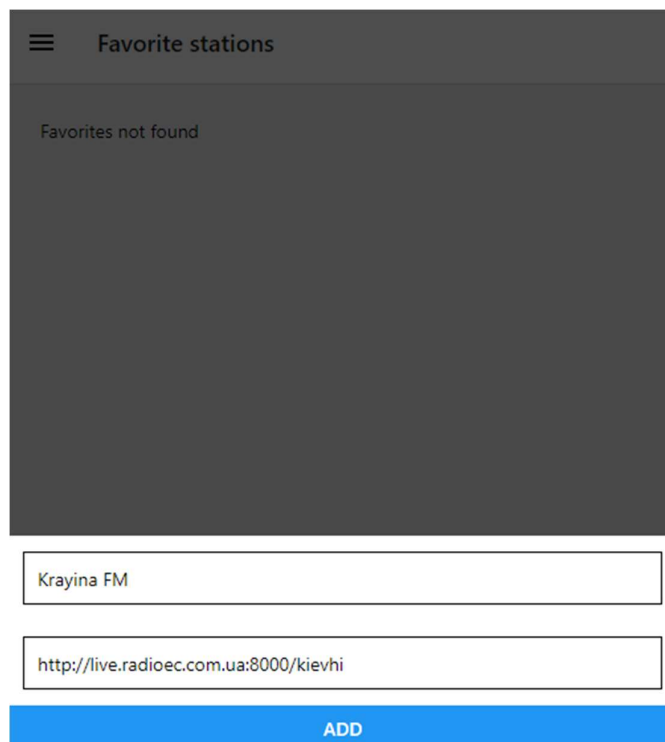


Рисунок 2.12 Модульне вікно з розблокованою кнопкою після введення даних інтернет радіостанції

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		42

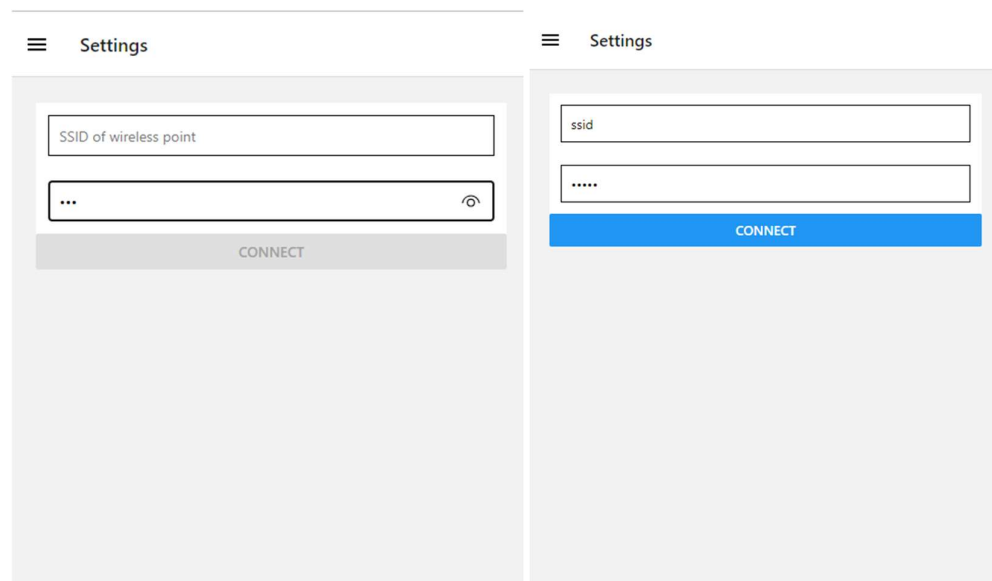
2.6. Проектування сторінки для підключення пристрою до мережі

Щоб мати змогу підключити пристрій до різних мереж, створюю ще одну сторінку, з полями введення параметрів точок доступу. Зроблю це аналогічно з модульним вікном. Розблоковувати після введення назви точки доступу, так як пароль може бути відсутній. Також зроблю, щоб при введенні паролю, його не було видно, для цього скористуюся властивістю компонента `TextInput`, `secureTextEntry`.

Після натискання кнопки, введені дані треба надіслати до пристрою, тому створю для цього `Epic` функцію, а в компоненті викликати функцію `dispatch` для передавання значень до неї. Передача даних пристрою відбуватиметься за допомогою рядка запиту в `URL`.

Лістинг 2.39 `Epic` функція для підключення пристрою до мережі

```
const connectToAPEpic = (action$, state$) => action$.pipe(
  ofType('CONNECT_AP'),
  switchMap((params) => {
    return
    ajax.get(`http://ztu.local/connect?ssid=${params.ssid}&password=${params.password}`)
  })
);
```



The image contains two side-by-side screenshots of a mobile application interface. Both screens have a header with a hamburger menu icon and the title 'Settings'. The left screenshot (a) shows a form with two input fields: the first is labeled 'SSID of wireless point' and contains the text '...'; the second is a password field with a toggle icon on the right. Below the fields is a gray button labeled 'CONNECT'. The right screenshot (b) shows the same form, but the password field is now filled with '*****' and the 'CONNECT' button is blue.

Рисунки 2.13 а) Модульне вікно з заблокованою кнопкою
б) Модульне вікно з розблокованою кнопкою після введення параметрів мережі

Для перевірки надсилання запиту скористаюся «Інструментами розробки» у браузері та перегляну вкладку Network

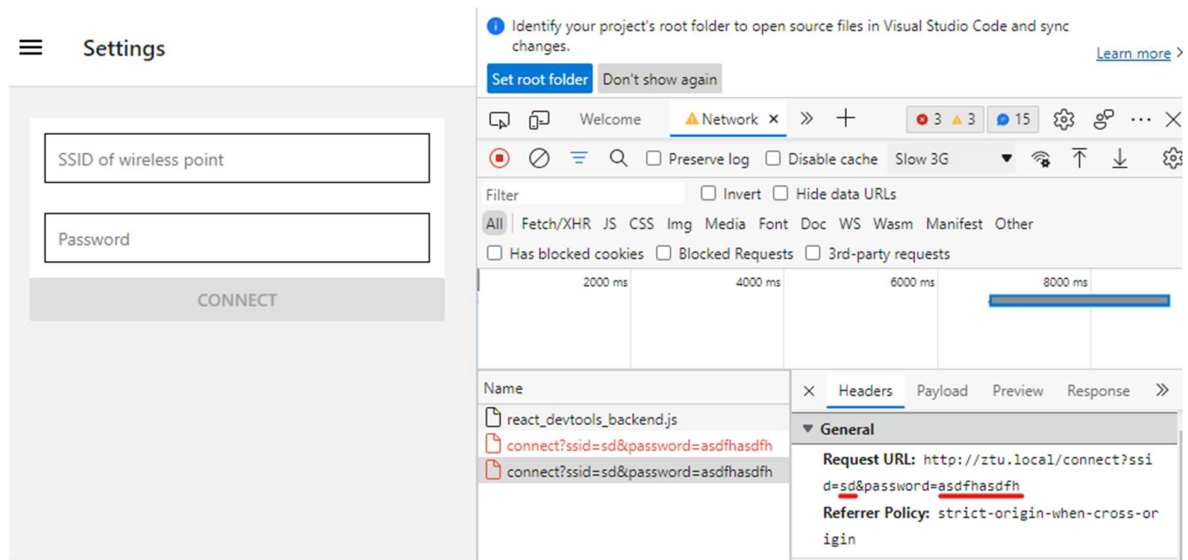


Рисунок 2.14 Перевірка надсилання запиту

2.7. Додавання функцій на головній сторінці

На головній сторінці на даний момент відображається обмежена кількість станцій, які дістаються за посиланням вказаним в Еріс функції. Для того, щоб додати можливість перегляду одразу всіх станцій, додам компонент, який відповідатиме за зміну режиму перегляду.

Створю перемикач, для цього використаю компонент Switch, для якого створю змінну стану, яка буде відповідати за його відображення. При зміні значення буде мінятися змінна у сховищі, яка відповідає, за кількість станцій на сторінці.

Лістинг 2.40 Компонент перемикача зі станом

```
const [isEnabled, setIsEnabled] = useState(!limit);
const toggleSwitch = () => {
  setIsEnabled(!isEnabled);
  dispatch(isEnabled ? setDefaultLimit() : setResetLimit())
};
...
<Switch onChange={toggleSwitch}
          value={isEnabled}/>
```

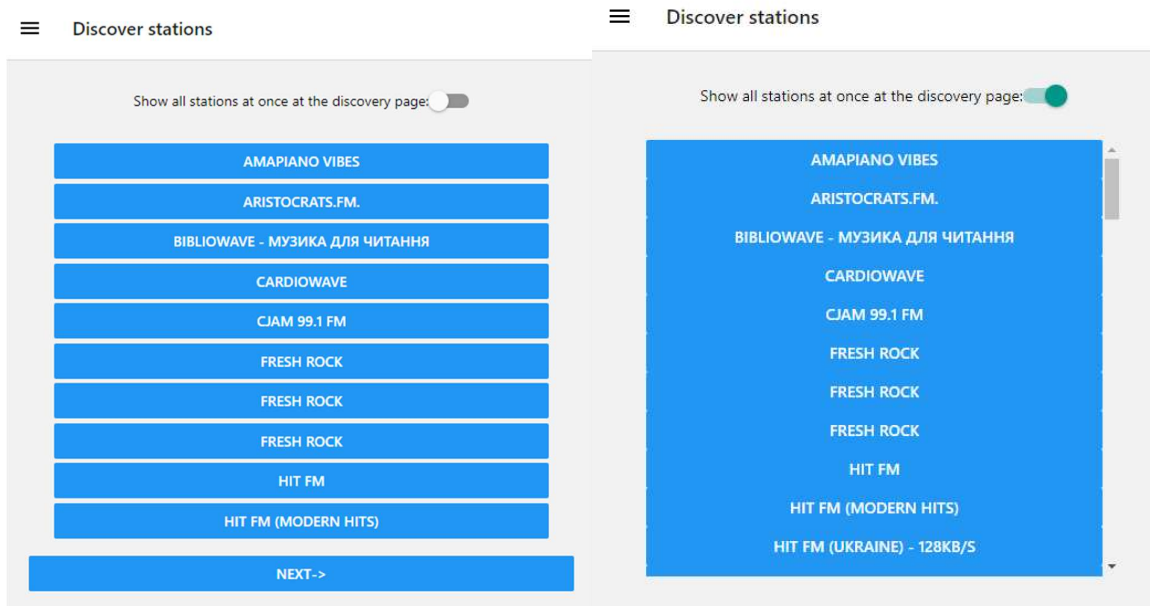
Лістинг 2.41 Додавання до сховища змінною, відповідальної за кількості значень на сторінці

```
const initialState = {
  ..
  limit: 10, }
...
export default combineReducers({
  ...
  limit: limitReducer, });

function limitReducer(state = initialState.limit, action) {
  switch (action.type) {
    case 'SET_DEFAULT_LIMIT':
      return 10
    case 'RESET_LIMIT':
      return 0;
  }
  return state;
}
```

Лістинг 2.42 Еріс функція, для зміни перемикача

```
const setDefaultLimit = (action$, state$) => action$.pipe(
  ofType('RESET_LIMIT'),
  switchMap((action) => {
    return ajax.getJSON(`https://de1.api.radio-
browser.info/json/stations/bylanguage/${state$.value.stationLanguage}`).pipe(
  switchMap(response => of(setStations(response.map(({ name, url_resolved }) => ({
name, url_resolved }))).filter((i) => i.url_resolved.slice(0, 5) == 'http:'))))));
```



Рисунки 2.15 а) Головна сторінка з вимкненим перемикачем;
б) З ввімкненим

Для того, щоб додати можливість вибору мови, створю випадний список, для вибору мову. Відповідно до мови буде змінюватися посилання у Еріс функції, та станції будуть показуватися за конкретною мовою.

Для створення використаю компоненти:

- Modal – для керування появою та зникненням списку;
- TouchableOpacity – для відстеження кліку повз;
- FlatList – для рендерингу списку доступних для вибору мов.

Щоб відрендерити список під кнопкою, скористаюся хуком useRef, та створю посилання на сам компонент, за допомогою посилання зможу взяти його ширину та координати, для того, щоб помістити список, під ним.

Після кліку на обрану мову, прибереться випадаючий список та стан в сховищі зміниться, в Еріс функції надішлеться запит для отримання станцій іншою мовою.

Лістинг 2.43 Робота з координатами для рендерингу списку

```
const openDropdown = () => {
  DropdownButton.current.measure((_fx, _fy, _w, h, _px, py) => {
    setDropdownTop(py + h);
    setDropdownLeft(_px);
    setDropdownWidth(_w);
  });
};
```

```

    setVisible(true);
};

```

Лістинг 2.44 Епіс функція для зміни мови, з перевіркою перемикача

```

const setLanguageEpic = (action$, state$) => action$.pipe(
  ofType('SET_LANGUAGE'),
  switchMap((a) => {
    if (a.payload.limit > 0)
      return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/by-
language/${state$.value.stationLanguage}`).pipe(
        map(response =>
          from(response.map(({ name, url_resolved }) => ({ name, url_resolved
}))).filter((i) => i.url_resolved.slice(0, 5) == 'http:')
        ).pipe(
          bufferCount(state$.value.limit),
        )
      ),
    mergeAll(),
    toArray(),
    map((result) => {
      stations = result;
      state$.value.offset = 0;
      return setStations(result[state$.value.offset]);
    })
  )
  else if (a.payload.limit == 0)
    return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/by-
language/${state$.value.stationLanguage}`).pipe(
      map(response => (setStations(response))))

  return empty();

});

```

Висновок до розділу 2

У розділі спроектовано додаток, який має бічне меню, для навігації, в якому є декілька сторінок, на головній сторінці користувач може переглянути доступні інтернет радіо станції з зручною навігацією, вказувати бажану мову, якою мовить станція. При натисканні на кнопку з назвою станції, з'явиться модальне вікно, де буде можливість зберегти станцію, відкрити у браузері для перевірки роботи, передавати посилання пристрою Arduino для подальшого програвання. На сторінці збережених радіо станцій, можна побачити збережені раніше станції, їх можна програвати на пристрої, або у браузері, також є можливість видаляти та додавати їх вручну. На сторінці налаштувань можна вказувати пристрою, до якої мережі йому підключитися та вказати дані для підключення.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						48
Зм.	Арк	№ докум	Підпис	Дата		

РОЗДІЛ 3. РОЗРОБКИ ВЕБСЕРВЕРА ДЛЯ ПРИЙМАЧА НА ARDUINO

3.1. Розробка алгоритму пристрою

Пристрій матиме функціонал, який складатиметься з таких функцій:

- приймання параметрів для підключення до мережі;
- зберігання даних цієї мережі та спроба підключення до неї ж після повторного вмикання;
- програвання аудіо інтернет потоку на аудіо пристрій.

Таким чином алгоритм роботи пристрою матиме вигляд:

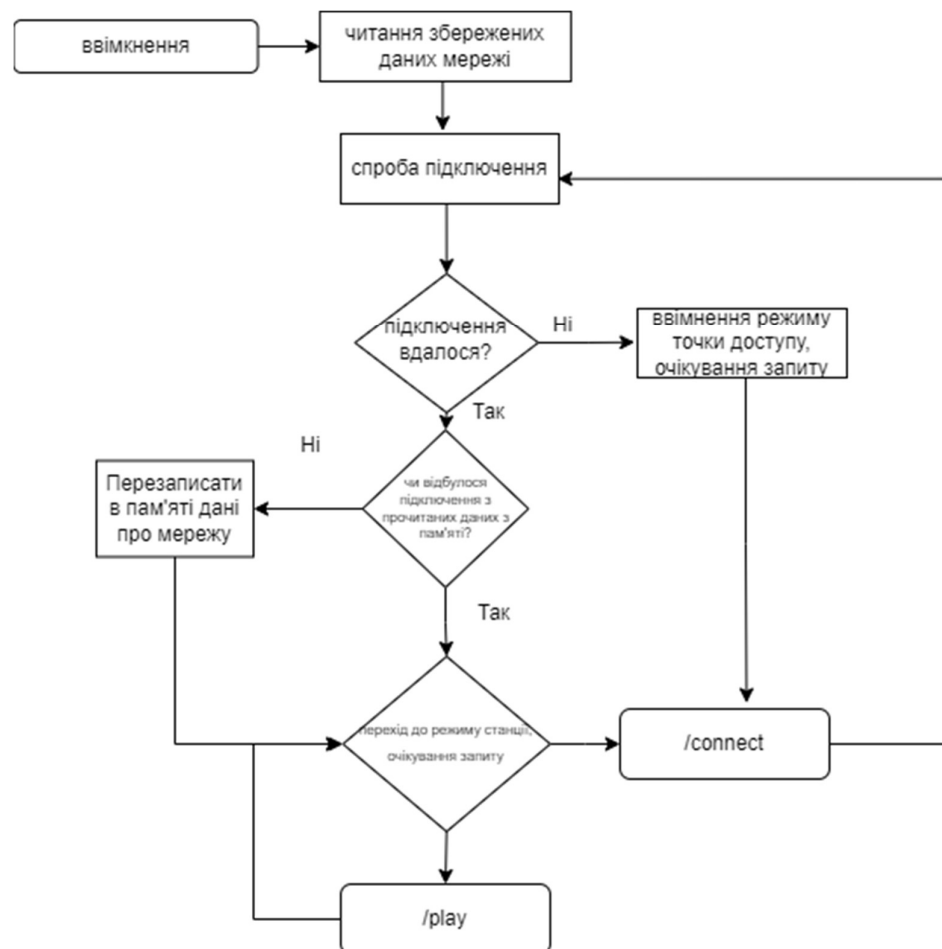


Рисунок 3.1 – Алгоритм роботи пристрою

3.2. Розробка схеми

Структурна схема пристрою для прийому та програвання інтернет радіо зображена на рисунку 3.2.

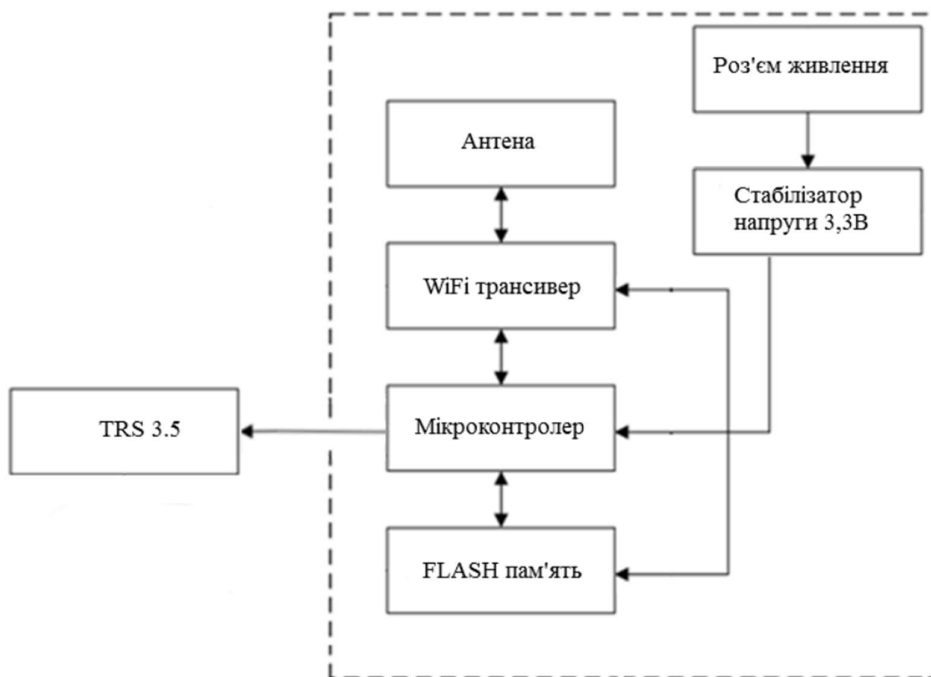


Рисунок 3.2 – Структурна схема пристрою приймача інтернет радіо

На схемі можна бачити, що плата містить і мікроконтролер, і WiFi трансивер, тому не має потреби підключати їх окремо. За допомогою такого модулю, вестиму розробку аудіо приймача. Крім плати на схемі, зображено TRS-роз'єм, на вихід з якого передаватиметься звук.

Для розробки пристрою було обрано модуль «Wemos D1 mini». Це повноцінний контролер Енергонезалежної пам'яті(EEPROM) мікроконтролер не має, тому при зверненні до неї, відбуватиметься запис в флеш-пам'ять, застосовуватимуть це для зберігання даних мережі, щоб пристрій пам'ятав останню мережу, та міг підключитися до неї після повторно підключення.

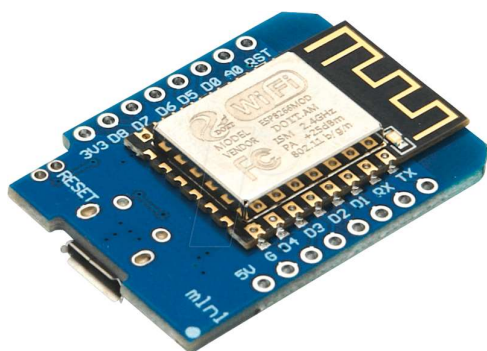


Рисунок 3.3 – Плата «Wemos D1 Mini»

Для передачі аудіо, під'єднаю аудіо роз'єм 2.5 мм до порту RX(RXD), так як додаткової панелі ЦАП додатково не використовується, та цей порт відповідатиме за вихід аудіо.



Рисунок 3.4 – Принципова схема пристрою

3.3. Створення точки доступу, веб сервера, та приєднання до іншої точки доступу

Для підключення модуля до мережі, спочатку необхідно просканувати доступні для модуля мережі в зоні його досягання, щоб дати користувачу можливість вибору мережі. Зустрічаються варіанти проектів, де дані мережі вписуються в коді явно, методом жорсткого кодування, цей варіант буде працювати, за умови якщо пристрій буде завжди в межах одного місцезнаходження, та ім'я мережі не буде змінюватися. В іншому разі треба буде вписувати у прошивку пристрою кожного разу. Застосую інший варіант, дозволю користувачу передавати параметри для підключення, також додаю можливість дивитися які мережі доступні пристрою, скануванням мереж.

Для реалізації цього було проаналізовано приклад з документації спільноти ESP8266[34] з використанням бібліотеки ESP8266WiFi.h. Щоб передати ці дані користувачу, створю вебсервер в режимі точки доступу, для цього скористаюся цією ж бібліотекою. Для реалізації слід використати метод softAP, який має наступну реалізацію:

Лістинг 3.1 Реалізація методу softAP, класу ESP8266WiFiAPClass

```
// In ESP8266WiFiAPClass
public: bool softAP(const char *ssid, const char *psk = NULL, int channel = 1,
int ssid_hidden = 0, int max_connection = 4)
```

В який треба обов'язково передати вказати назву точки доступу(SSID), та за бажанням: ключ(PSK); WLAN – канал; чи є точка прихованою; максимальну кількість підключень.

Щоб передати дані про доступні мережі клієнту створю вебсервер[36], для цього використаю бібліотеку ESP8266WebServer, її використання необхідно для обробки HTTP – запитів. Тому створю об'єкт, в який передам номер порту, який є закріплений за протоколом http:

Лістинг 3.2 Створення об'єкту класу ESP8266WebServer

```
ESP8266WebServer server(80);
```

Після цього треба додати у метод loop рядок `server.handleClient()`, для прослуховування HTTP запитів від клієнтів. Якщо `handleClient` виявить нові запити, він автоматично виконає потрібні функції, вказані під час налаштування у функцію `setup`.

Для реалізації REST підходу, в цьому об'єкті є метод `on(path, function)`, за допомогою якого відбувається реєстрація обробника функції, за заданим шляхом. Для передачі клієнту даних про доступні точки використаю формат JSON, для цього трансформую рядок відповідним чином.

Лістинг 3.3 Передача відповіді про доступні для пристрою точки доступу

```
#define HOSTNAME "webserver"

ESP8266WebServer server(80);

void setup() {
  WiFi.softAP("ZTU","password");
  WiFi.setHostname(HOSTNAME);

  server.on("/apList", handleShowApList);
  server.begin();
}

void handleShowApList() {
  int ssidNum = WiFi.scanNetworks();
  if( ssidNum > 0 ){
    String resultJson = "{ \"Points\": [";

    for (int i = 0; i < ssidNum; i++) {
      resultJson += "\"" + WiFi.SSID(i) + "\"";
      if (i != ssidNum - 1)
        resultJson += ",";
    };
    resultJson += "] }";
    server.send(200, "application/json", resultJson);
  }
  else
    server.send(200, "application/json", "{\"No available points\" } ");
}
```

Створю також обробник для того, щоб підключатися до конкретної точки доступу, для цього пристрій прийматиме дані для підключення за допомогою рядка запиту в URL. В екземплярі класу `ESP8266WebServer`, для цього є метод `arg`, в який передається назва параметру. Цей метод поверне значення яке передано, до вказаного параметру.

Для ініціалізації підключення до точки доступу використаю метод `WiFi.begin()`, в який передам вище згадані значення.

Якщо до мережі не вдасться підключитися, точка доступу залишиться ввімкненою, якщо вдасться, то після підключення до мережі, пристрій з режиму точки доступу перейде до режиму станції, тому варто вимкнути точку доступу, для цього є метод екземпляру класу ESP8266WiFi:

```
public: bool softAPdisconnect(bool wifioff = false)
```

Зареєструю цей обробник до сервера:

Лістинг 3.4 Підключення пристрою до точки доступу, перехід з режиму точки доступу до режиму станції

```
void setup() {
...
server.on("/connect", handlerConnect);
...
void handlerConnect() {
    WiFi.softAPdisconnect(true);
    String ssid = server.arg("ssid");
    String password = server.arg("password");

    WiFi.begin(ssid, password);

    if (WiFi.status() != WL_CONNECTED) {
        WiFi.softAP("ZTU", "password");
    }else{
        isConnected = true;
    }
}
```

Для того, щоб не вводити кожного разу IP адресу при звертанні до пристрою, зміню доменні ім'я, після невдалих спроб налаштування mDNS, знайдено рішення за допомогою NetBIOS. Після підключення постає проблема, що кожного разу після перезавантаження пристрою, треба повторювати ті ж дії, з підключенням до мережі. Для збереження використаю даних підключеної мережі бібліотеку EEPROM[37], яка в даному випадку зберігатиме дані для підключення у флешпам'яті. Для запису до неї створю окремий метод, який перевірятиме чи є підключення, та застосовуватиму його в функції setup.

Після коректного підключення до мережі, з функції, яка приймає параметри, записуватиму дані мережі. В функції setup читатиму дані з цієї пам'яті, та пробуватиму підключитися за допомогою цих даних, якщо підключення не вдалося запускатиметься режим точки доступу, та чекатиме на отримання даних для підключення. Читання та написання даних в цю пам'ять робитиму по байтово в циклі.

Лістинг 3.5 Читання та зберігання даних з пам'яті EEPROM

```

void setup() {
    EEPROM.begin(512);
    ...
    String esid;
    for (int i = 0; i < 32; ++i)
    {
        esid += char(EEPROM.read(i));
    }

    String epass = "";
    for (int i = 32; i < 96; ++i)
    {
        epass += char(EEPROM.read(i));
    }
    WiFi.begin(esid.c_str(), epass.c_str());
    checkWhetherConnected(esid.c_str(), epass.c_str(), true)
    server.on("/connect", handleConnect);

    server.begin();
}
void handleConnect() {
    String ssid = server.arg("ssid");
    String password = server.arg("password");

    WiFi.begin(ssid, password);

    checkWhetherConnected(ssid, password);
}
void checkWhetherConnected(String ssid, String password, bool fromMemory){
    if (testWifi())
    {
        // server.on("/apList", handleShowApList);
        WiFi.softAPdisconnect(true);

        Serial.println("inside");
        if(fromMemory)
            return;

        Serial.println("clearing eeprom");
        for (int i = 0; i < 96; ++i) { //clearing eeprom
            EEPROM.write(i, 0);
        }

        for (int i = 0; i < ssid.length(); ++i)
        {
            EEPROM.write(i, ssid[i]);
            Serial.print("Wrote: ");
            Serial.println(ssid[i]);
        }
        Serial.println("writing eeprom pass:");
        for (int i = 0; i < password.length(); ++i)
        {
            EEPROM.write(32 + i, password[i]);
            Serial.print("Wrote: ");
            Serial.println(password[i]);
        }
        EEPROM.commit();
    }
}

```

Зм.	Арк	№ докум	Підпис	Дата

КІ.КР.Б -123 – 09 - ПЗ

Арк

55

```

else
{
    Serial.println("outside");
    launchWeb();
}
}
bool testWifi(void)
{
    int c = 0;
    Serial.println("Waiting for Wifi to connect");
    while ( c < 20 ) {
        Serial.println(WiFi.status());
        Serial.println(c);
        if (WiFi.status() == WL_CONNECTED)
        {
            return true;
        }
        delay(1000);
        Serial.print("*");
        c++;
    }
    return false;
}

```

Створюю обробник запиту для приймання посилання на потік. Для його програвання використовуватиму бібліотеку ESP8266Audio, яка дасть можливості декодування потоку, та його програвання. Робиться в ній це за допомогою цифрового стандарту для передачі звуку I²S, який використовується для передачі ІКМ аудіо даних між інтегральними схемами.

Тож для програвання необхідно створити, об'єкт класу AudioSourceHTTPStream, який буде приймати потік по посиланню. Потім AudioSourceBuffer, для виділення додаткового буферу в оперативній пам'яті для попереднього об'єкту, так як треба буде забезпечити безперервне відтворення потоку. AudioOutputI2SNoDAC – для використання I²S інтерфейсу для програвання, та передачі звуку на канал передачі даних, на порт RXD, який приймає дані. AudioGeneratorMP3 – для декодування потоку.

Далі в останньому об'єкті викликати функцію begin(), передавши туди об'єкт буферу та об'єкт, який відповідає за програвання, та передачу.

Лістинг 3.6 Приймання посилання та його програвання на вихід RX

```

void setup() {
    ...
    server.on("/url", handleURL);
    ...
}

```

```

void handleURL() {
    audioFlag = false;
    char charBuf[50];
    String urlString = server.arg("url");
    Serial.printf(stringUrl.c_str());
    urlString.toCharArray(charBuf, 50);

    const char *URL = charBuf;

    file = new AudioFileSourceHTTPStream(URL);
    buff = new AudioFileSourceBuffer(file, 2048);
    out = new AudioOutputI2SNoDAC();
    mp3 = new AudioGeneratorMP3();
    mp3->begin(buff, out);

    audioFlag = true;
    playingnow = urlString.c_str();

    server.send(200, "text/html", nameAP);
}

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		57

Висновок до розділу 3

У розділі створено пристрій на платформі Arduino, який працює у режимі веб сервера, який може працювати у режимі точки доступу, до того моменту, поки його не підключити. Після підключення пристрій, запам'ятає мережу, шляхом запису параметрів до флеш пам'яті, та при наступному вмиканні зможе автоматично приєднатися до мережі, при умові, що вона буде знаходитися у зоні доступу. Після підключення до мережі, пристрою треба передати посилання для програвання інтернет радіо станції, після цього він зможе транслювати потік на підключений пристрій для аудіо відтворення.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						58
Зм.	Арк	№ докум	Підпис	Дата		

ВИСНОВКИ

Метою роботи є створення пристрою, приймача інтернет радіо. У ході роботи пристрій, який працює у режимі веб сервера та розроблено додаток до керування ним. Пристрій виступатиме у ролі точки доступу до передавання даних про мережу. Після підключення пристрій запам'ятає дані мережі та буде намагатися підключитися до неї, після наступного включення.

Додаток до пристрою включає: сторінку для перегляду доступних станцій для програвання, з можливістю відкривання посилання у веб переглядачі, таким чином перевіряючи, чи дійсне посилання. Реалізовано зберігання окремо, зберігання виконано засобами SQLite, збережені станції можна побачити у вікні збережених, їх звідти можна видаляти та додавати вручну. Також створено сторінку для підключення пристрою до мережі.

При розробці додатку використовувалися бібліотеки:

- для керуванням всього стану додатку Redux;
- для асинхронного керування станом RxJS.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		59

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Boxall J. Arduino Workshop: A Hands-On Introduction with 65 projects / John Boxall. – San Francisco: No Starch Press, 2013. – Chapter 8 Expanding Your Arduino; P. 161–186.
2. Boxall J. Arduino Workshop: A Hands-On Introduction with 65 projects / John Boxall. – San Francisco: No Starch Press, 2013. – Article 3.3 The Solderless Breadboard; P. 41–42.
3. Ярнольд С. Arduino для начинающих. / Стюарт Ярнольд. – Москва: Эксмо, 2017. – Раздел 1.1 Что такое Arduino?; ст 10-12.
4. Purdum J. Beginning C for Arduino, Second Edition: Learn C Programming for the Arduino / Jack Purdum. – Cincinnati: Apress, 2015. – Chapter 12 Arduino Libraries; P 277-298.
5. Boxall J. Arduino Workshop: A Hands-On Introduction with 65 projects / John Boxall. – San Francisco: No Starch Press, 2013. – Article 3.3 Article 2.3 Creating Your First Sketch in the IDE; P. 27–31.
6. Dabit N. React Native in Action 1st ed. / Nader Dabit. – NY: Manning Publications, 2019. – Article 1.4.6, Thinking in components; P. 9-10.
7. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Chapter 26, Handling Application State; P. 443-458.
8. P. Daniels P., Atencio L. RxJS in Action / Paul P. Daniels, Luis Atencio. – Shelter Island: Manning Publications, 2017. – Article 2.1.1, Functional programming; P. 30–37.
9. P. Daniels P., Atencio L. RxJS in Action / Paul P. Daniels, Luis Atencio. – Shelter Island: Manning Publications, 2017. – Article 1.5, Reactive and other programming paradigms; P. 26–27.
10. Web scraping [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Web_scraping
11. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Section 1.3, Hooks; P. 21-22.

					<i>КІ.КР.Б -123 – 09 - ПЗ</i>	<i>Арк</i>
<i>Зм.</i>	<i>Арк</i>	<i>№ докум</i>	<i>Підпис</i>	<i>Дата</i>		<i>60</i>

12. Simpson K., You Don't Know JS: ES6 and Beyond / Kyle Simpson. – Sebastopol: O'Reilly Media; 2015. Article 2.4, Destructuring; P. 19-32.
13. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Section 1.3, Performing initialization and cleanup actions; P. 443-458.
14. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Article 18.3, Indicating progress; P. 322-325.
15. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Chapter 17, Rendering Item Lists; P. 306-320.
16. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Chapter 5, Event Handling - The React Way; P. 91-103.
17. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Chapter 21, Displaying Modal Screens; P. 364-382.
18. Dabit N. React Native in Action 1st ed. / Nader Dabit. – Shelter Island: Manning Publications, 2019. – Article 2.1.1, Correctly manipulating component state; P. 28-32.
19. Masiello E. Mastering React Native. / Eric Masiello. – Birmingham: Packt Publishing; 2017. Article 7.4.2, Opening the browser with linking; P. 280-283.
20. Dabit N. React Native in Action 1st ed. Shelter Island / Nader Dabit. – Shelter Island: Manning Publications, 2019. – Article 1.7.3, Component composition; P. 17-19.
21. Has-a [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Has-a>
22. Dabit N. React Native in Action 1st ed. Shelter Island / Nader Dabit. – Shelter Island: Manning Publications, 2019. – Chapter 8, Using the Redux data architecture library; P. 179-192.
23. Masiello E. Mastering React Native. / Eric Masiello. – Birmingham: Packt Publishing; 2017. Article 5.2.6, Implementing Redux; P. 192-198.

24. Simpson K., You Don't Know JS: ES6 and Beyond / Kyle Simpson. – Sebastopol: O'Reilly Media; 2015. Article 2.2, Spread / Rest; P. 13-15.
25. Boduch A., Derks R. React and React Native / Adam Boduch, Roy Derks. – Birmingham: Packt Publishing, 2020. Section 4.4, Using reducer Hooks to scale state management; P. 81-90.
26. Masiello E. Mastering React Native. / Eric Masiello. – Birmingham: Packt Publishing; 2017. Section 5.2.3.2, Action creators; P. 196-197.
27. P. Daniels P., Atencio L. RxJS in Action / Paul P. Daniels, Luis Atencio. – Shelter Island: Manning Publications, 2017. – Article 4.5 Buffering in RxJS; P. 111–116.
28. P. Daniels P., Atencio L. RxJS in Action / Paul P. Daniels, Luis Atencio. – Shelter Island: Manning Publications, 2017. – Article 4.5 Buffering in RxJS; P. 111–116.
29. Masiello E. Mastering React Native. / Eric Masiello. – Birmingham: Packt Publishing; 2017. – Article 4.1.2, Touchable; P. 133-136.
30. Taylor A. SQL For Dummies, 9th Edition. / Allen G. Taylor – Hoboken: John Wiley & Sons, Inc; 2019 – Article 3.2, Data Definition Language; P.56-67.
31. Taylor A. SQL For Dummies, 9th Edition. / Allen G. Taylor – Hoboken: John Wiley & Sons, Inc; 2019 – Article 3.2, Data Manipulation Language; P.68-76.
32. Dabit N. React Native in Action 1st ed. / Nader Dabit. – NY: Manning Publications, 2019. – Article 6.4, Using DrawerNavigator to create drawer-based navigation; P. 160-161.
33. Morgan J. Simplifying JavaScript: Writing Modern JavaScript with ES5, ES6, and Beyond 1st Edition / Joe Morgan. – The Pragmatic Programmers, LLC, 2018. – Article 23, Pull Out Subsets of Data with filter() and find(); P. 141-146.
34. Masiello E. Mastering React Native. / Eric Masiello. – Birmingham: Packt Publishing; 2017. – Article 4.1.9, TextInput; P. 158-163.

35. Scan [Электронный ресурс] // ESP8266 Arduino Core's documentation! – Режим доступа до ресурсу: <https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html>
36. Блум Дж. Изучаем Arduino: инструменты и методы технического волшебства. / Блум Джереми. – СПб: БХВ-Петербург; 2020. – Раздел 17.3.4 Программа сервера для платы Arduino; С. 476-488.
37. Purdum J. Beginning C for Arduino, Second Edition: Learn C Programming for the Arduino / Jack Purdum. – Cincinnati: Apress, 2015. – Article 10.3 EEPROM Memory; P 233-241.

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		63

ДОДАТКИ

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		64

Файл App.js, який є головним, та виступає в ролі контейнера для інших файлів

```
import 'react-native-gesture-handler';
import React from 'react';
import StationsScreen from './app/screens/StationsScreen';
import SettingsScreen from './app/screens/SettingsScreen';
import { Provider } from 'react-redux';
import { applyMiddleware, createStore } from 'redux';
import rootReducer from './app/reducer';
import rootEpic from './app/epic';
import { createEpicMiddleware } from 'redux-observable';
import { NavigationContainer } from '@react-navigation/native';
import {
  createDrawerNavigator,
} from '@react-navigation/drawer';
import FavoriteScreen from './app/screens/FavoriteScreen';

const Drawer = createDrawerNavigator();

const App = () => {
  const epicMiddleware = createEpicMiddleware();
  const store = createStore(rootReducer,
    applyMiddleware(epicMiddleware)
  );
  epicMiddleware.run(rootEpic);
  return (
    <Provider store={store} >
      <NavigationContainer>
        <Drawer.Navigator>
          <Drawer.Screen name="Discover stations" component={Sta-
tionsScreen} />
          <Drawer.Screen name="Favorite stations" component={Favor-
iteScreen} />
          <Drawer.Screen name="Settings" component={SettingsScreen} />
        </Drawer.Navigator>
      </NavigationContainer>
    </Provider>
  );
};

export default App;
```

Файл FavoriteScreen.js, який використовується для відображення вікна зі збереженими станціями

```
import React, { useState } from 'react';
import { FlatList, Text, View, Button } from 'react-native';
import { useSelector } from 'react-redux';
import AddFavorite from '../components/AddFavorite';
import ChooseStationModal from '../components/ChooseStationModal';

export default FavoriteScreen= () => {
  const { favoriteStations } = useSelector((state) => state.favoriteStations);

  const [modalStation, setModalStation] = useState();

  const [isModalVisible, setIsModalVisible] = useState(false);
  const handleModal = () => setIsModalVisible(() => !isModalVisible);

  const [isModalCreateVisible, setIsModalCreateVisible] = useState(false);
  const handleModalCreate = () => setIsModalCreateVisible(() => !isModalCreateVisible);

  return (
    <View style={{ flex: 1, padding: 25 }}>
      <>
        {
          (!favoriteStations?.length) ? <Text>Favorites not found</Text> :
          (
            <>
              <FlatList
                data={favoriteStations}
                contentContainerStyle={{ flexGrow: 1, justifyContent:
'space-around' }}
                renderItem={({ item }) => (
                  <Button onPress={(e) => { handleModal(); return
setModalStation(item); }} key={item.id} title={item.name}></Button>
                )}
              />
              {isModalVisible && <ChooseStationModal
modalStation={modalStation} isModalVisible={isModalVisible} handleModal={() =>
handleModal()} />}
            </>
          )
        }
        <View style={{ flex: 1, padding: 25, justifyContent: 'flex-end' }}>
          <Button onPress={() => { setIsModalCreateVisible(true) }} ti-
tle='Add to favorite'></Button>
        </View>
        < AddFavorite isModalVisible={isModalCreateVisible} handleModal={() =>
handleModalCreate()} />
      </>
    </View >
  );
};
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						66
Зм.	Арк	№ докум	Підпис	Дата		

Файл SettingsScreen.js, який за допомогою якого можна виконати підключення пристрою до безпроводної точки доступу

```
import React, { useEffect, useState } from 'react';
import { TextInput, View, StyleSheet, Button, } from 'react-native';
import { useDispatch, } from 'react-redux';
import { connectAp, } from '../actions';

export default SettingsScreen = () => {
  const dispatch = useDispatch();

  const [ssid, onChangeSsid] = useState('');
  const [password, onChangePassword] = useState('');

  const [isEnabled, setIsEnabled] = useState(false);

  useEffect(() => {
    if (ssid)
      setIsEnabled(true);
    else
      setIsEnabled(false);
  }, [ssid]);

  return (
    <View style={{ flex: 1, padding: 25 }}>
      {
        <>
          <View style={{ backgroundColor: 'white', bottom: '0%' }}>
            <TextInput
              style={styles.input}
              onChangeText={onChangeSsid}
              value={ssid}
              placeholder="SSID of wireless point"
            />
            <View>
              <TextInput
                style={styles.input}
                onChangeText={onChangePassword}
                value={password}
                placeholder="Password"
                secureTextEntry={true}
              />
            </View>
            <Button title={'Connect'}
              disabled={!isEnabled}
              onPress={() => { onChangeSsid(''); onChangePassword('');
dispatch(connectAp(ssid, password)); }} />
          </View>
        </>
      }
    </View>
  )
}
```

```

        </View >
    );
};

const styles = StyleSheet.create({
    input: {
        height: 40,
        margin: 12,
        borderWidth: 1,
        padding: 10,
    },
});

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						68
Зм.	Арк	№ докум	Підпис	Дата		

Файл StationsScreen.js, який рендерить вікно, з доступними для програвання інтернет радіостанціями

```
import React, { useState, useEffect } from 'react';
import { ActivityIndicator, FlatList, Text, View, Button, Switch, SafeAreaView } from
'react-native';
import { useDispatch, useSelector } from 'react-redux';
import OffsetButtons from '../components/OffsetButtons';
import { getStations, setDefaultLimit, setResetLimit } from '../actions';
import ChooseStationModal from '../components/ChooseStationModal';

export default StationsScreen = () => {
  const stations = useSelector((state) => state.stations);
  const offset = useSelector(state => state.offset);
  const limit = useSelector(state => state.limit);
  const stationLanguage = useSelector(state => state.stationLanguage);
  const [modalStation, setModalStation] = useState();
  const [isModalVisible, setIsModalVisible] = useState(false);
  const handleModal = () => setIsModalVisible(() => !isModalVisible);

  const [isEnabled, setIsEnabled] = useState(!limit);
  const toggleSwitch = () => {
    setIsEnabled(!isEnabled);
    dispatch(isEnabled ? setDefaultLimit() : setResetLimit())
  };

  const dispatch = useDispatch();
  useEffect(() => {
    dispatch(getStations(limit, offset));
  }, [offset]);

  return (
    <View style={{ flex: 1, padding: 25 }}>
      {
        (!stations?.length) ? <ActivityIndicator size="large" style={{ justi-
fyContent: "center", flexDirection: "row", flex: 1 }} /> : (
          <>
            <SafeAreaView style={{ justifyContent: 'center', position:
'relative', flexDirection: 'row', margin: 5 }}>
              <Text>
                Show all stations at once at the discovery page:
              </Text>
              <Switch
                onChange={toggleSwitch}
                value={isEnabled}
              />
            </SafeAreaView>
            <Text style={{ right: '5%', top: '5%', position: 'absolute'
}}>
```

```

        { /* {(limit || limit == 0) ? <>{offset} - {offset +
10}</> : <>0 - {stations && stations?.length}</>} */}
        </Text>
        <View style={{ padding: 25, flex: 1 }}>
            <>
                <FlatList
                    data={stations}
                    contentContainerStyle={{ flexGrow: 1, justifyCon-
tent: 'space-around' }}
                    renderItem={({ item }) => (
                        <Button onPress={(e) => { setModalStation(item); handleModal(); }}
key={item.id} title={item.name}></Button>
                    )}
                />
                {isModalVisible && <ChooseStationModal
modalStation={modalStation} isModalVisible={isModalVisible} handleModal={() =>
handleModal()} add={true} />}
            </>
        </View >
    </>
    )
    }
    {!!limit && <OffsetButtons />}
</View >
);
};

```

Файл компонента AddFavorite.js, за допомогою якого можна додавати станції з головного вікна до збережених

```
import { View, Button, TextInput, StyleSheet } from 'react-native';
import { addFavoriteStation, } from '../actions';
import Modal from "react-native-modal";
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';

export default AddFavorite = ({ isModalVisible, handleModal }) => {
  const dispatch = useDispatch();

  const [name, onChangeName] = useState(null);
  const [url, onChangeUrl] = useState(null);
  const [isDisabled, setIsDisabled] = useState(false);

  const handleChange = (url) => {
    onChangeUrl(url);

    setIsDisabled(url.slice(0, 5) === "http:");
  }

  return (
    <Modal isVisible={isModalVisible} style={{ opacity: 1, margin: 0, width:
'100%', bottom: 0, position: 'absolute' }}
      onSwipeComplete={() => handleModal()}
      onBackdropPress={() => handleModal()}
      onRequestClose={() => handleModal()}
    >
      <>
        <View style={{ backgroundColor: 'white', bottom: '0%' }}>
          <TextInput
            style={styles.input}
            onChangeText={onChangeName}
            value={name}
            placeholder="Name of station"
          />
          <TextInput
            style={styles.input}
            onChangeText={(e) => handleChange(e)}
            value={url}
            placeholder="Url of station"
          />
          <Button title={'Add'}
            disabled={!isDisabled}
            onPress={() => { handleModal(); onChangeName(null); on-
ChangeUrl(null); return dispatch(addFavoriteStation(name, url)); }} />
        </View>
      </>
    </Modal>
  );
};
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		71

```

    );
}
const styles = StyleSheet.create({
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  },
});
});

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		72

Файл компонента модального вікна ChooseStationModal.js, який використовується для програвання станції на пристрої, додавання до/видалення з збережених, програвання у браузері

```
import { Text, View, Button, TouchableWithoutFeedback, Linking } from 'react-native';
import { addToFavorite, deletefromFavorites, setUrl } from '../actions';
import Modal from "react-native-modal";
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';

export default ChooseStationModal = ({ modalStation, isModalVisible, handleModal, add
}) => {
  const dispatch = useDispatch();
  return (
    <Modal isVisible={isModalVisible} style={{ opacity: 1, margin: 0 }}>
      <TouchableWithoutFeedback
        onPressOut={() => handleModal()}
        activeOpacity={1}
      >
        <View style={{ with: '100%', height: '100%', justifyContent: 'flex-end' }}>
          { /* <Button title={ButtonTitle(modalStation.name)} */ }
          <Text style={{ backgroundColor: 'white', textAlign: 'center' }}>Station
name: <Text style={{
            fontWeight: 'bold',
            textDecorationColor: "rgb(33, 150, 243)",
            textDecorationLine: 'underline', fontStyle: 'italic'
          }}>{modalStation.name}</Text></Text>
          <Button title={`Play ${modalStation.name} on the ESP`}
            onPress={() => { handleModal(); return dis-
patch(setUrl(modalStation.url_resolved)) }} />
            {add ?
              <Button title="Add to favorite" onPress={() => { handleModal(); return
dispatch(addToFavorite(modalStation)); }} />
                : <Button title="Delete from favorites" onPress={() => { handleModal();
return dispatch(deletefromFavorites(modalStation.name)); }} />
              <Button title="Open with" onPress={() => { handleModal(); return Link-
ing.openURL(modalStation.url_resolved); }} />
            </View>
          </TouchableWithoutFeedback>
        </Modal >
      );
    };
  }
```

Файл компонента Dropdown.js, для вибору мови відображених станцій

```

import React, { useRef, useState } from 'react';
import { FlatList, StyleSheet, Text, TouchableOpacity, Modal, View, } from 'react-native';

const Dropdown = ({ label, data, onSelect }) => {
  const DropdownButton = useRef();
  const [visible, setVisible] = useState(false);
  const [selected, setSelected] = useState(undefined);
  const [dropdownTop, setDropdownTop] = useState(0);
  const [dropdownLeft, setDropdownLeft] = useState(0);
  const [dropdownWidth, setDropdownWidth] = useState(0);

  const toggleDropdown = () => {
    visible ? setVisible(false) : openDropdown();
  };

  const openDropdown = () => {
    DropdownButton.current.measure((_fx, _fy, _w, h, _px, py) => {
      setDropdownTop(py + h);
      setDropdownLeft(_px);
      setDropdownWidth(_w);
    });
    setVisible(true);
  };

  const onItemPress = (item) => {
    setSelected(item);
    onSelect(item);
    setVisible(false);
  };

  const renderItem = ({ item }) => (
    <TouchableOpacity style={styles.item} onPress={() => onItemPress(item)}>
      <Text>{item.label}</Text>
    </TouchableOpacity>
  );

  const renderDropdown = () => {
    return (
      <Modal visible={visible} transparent animationType="fade">
        <TouchableOpacity
          style={styles.overlay}
          onPress={() => setVisible(false)}
        >
          <View style={[styles.dropdown, { top: dropdownTop, left: dropdownLeft,
width: dropdownWidth }]}>
            <FlatList
              data={data}
              renderItem={renderItem}
              keyExtractor={(item, index) => index.toString()}
            >

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		74

```

        />
      </View>
    </TouchableOpacity>
  </Modal>
);
};
return (
  <TouchableOpacity
    ref={DropdownButton}
    style={styles.button}
    onPress={toggleDropdown}
  >
    {renderDropdown()}
    <Text style={styles.buttonText}>
      {(selected && `Language: ${selected.label}`) || label}
    </Text>
  </TouchableOpacity>
);
};
const styles = StyleSheet.create({
  button: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: 'white',
    height: 40,
    width: '100%',
  },
  buttonText: {
    flex: 1,
    textAlign: 'center',
  },
  icon: {
    marginRight: 10,
  },
  dropdown: {
    position: 'absolute',
    backgroundColor: 'white',
    width: '100%',
    shadowColor: '#000000',
    shadowRadius: 4,
    shadowOffset: { height: 4, width: 0 },
    shadowOpacity: 0.5,
  },
  overlay: {
    width: '100%',
    height: '100%',
  },
  item: {
    paddingHorizontal: 10,
    paddingVertical: 10,
    borderBottomWidth: 1,
  },
}); export default Dropdown;

```

Файл компонента OffsetButtons.js, який відповідає за відображення кнопок навігації

```
import React from 'react';
import { Alert, View, Button, } from 'react-native';
import { useDispatch, useSelector } from 'react-redux';
import { setDefaultOffset, setShowConfirmationDialog } from '../actions';

const increaseOffset = { type: 'INCREASE_OFFSET' };
const decreaseOffset = { type: 'DECREASE_OFFSET' };

export default OffsetButtons = () => {
  const offset = useSelector(state => state.offset);
  const showConfirmationDialog = useSelector(state => state.showConfirmationDialog);
  const dispatch = useDispatch();
  const showConfirmDialog = () => {
    return Alert.alert(
      "Thats the final page",
      `Would you like to get to the first?`,
      [
        {
          text: "Yes",
          onPress: () => {
            dispatch(setShowConfirmationDialog(false));
            dispatch(setDefaultOffset());
          },
        },
        {
          text: "No",
          onPress: () => {
            dispatch(setShowConfirmationDialog(false));
          }
        }
      ],
    );
  };
  return (
    <View style={{ bottom: 10, flexDirection: 'row', justifyContent: "space-around", alignItems: "flex-end" }}>
      {showConfirmationDialog.showConfirmationDialog && showConfirmDialog()}
      {(offset > 0) &&
        <View style={{ flex: 1 }}>
          <Button onPress={() => dispatch(decreaseOffset)}
            title="<-Previous"
          ></Button>
        </View>
        <View style={{ flex: 1 }}>
          <Button onPress={() => {
            dispatch(increaseOffset)
          }}
            title="Next->"
          ></Button>
        </View >
      </View >
    );
  }
}
```

Файл actions.js, де знаходяться об'єкти, які використовуються для занесення даних у сховище

```
export const setUrl = (url) => ({
  type: 'SET_URL',
  url,
});

export const setDefaultLimit = () => ({
  type: 'SET_DEFAULT_LIMIT'
});

export const setResetLimit = () => ({
  type: 'RESET_LIMIT'
});

export const addToFavorite = ({ name, url_resolved }) => ({
  type: 'ADD_FAVORITE',
  name, url_resolved,
});

export const deletefromFavorites = (name) => ({
  type: 'DELETE_FAVORITE',
  name,
});

export const deleteFavorite = (name) => ({
  type: 'DELETE_FAV',
  payload: { name },
});

export const setFavorites = (stations) => ({
  type: 'SET_FAVORITES',
  payload: { stations },
});

export const setDefaultOffset = () => ({
  type: 'SET_DEFAULT_OFFSET',
});

export const connectAp = (ssid, password) => ({
  type: 'CONNECT_AP',
  payload: { ssid, password },
});

export const addFavoriteStation = (name, url_resolved) => ({
  type: 'ADD_FAV',
  payload: { name, url_resolved }
});

export const setShowConfirmationDialog = (showConfirmationDialog) => ({
  type: 'SHOW_CONFIRMATION_DIALOG',
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		77

```

    payload: showConfirmationDialog,
  });

const GET_STATIONS = 'GET_STATIONS';

export function getStations(limit, offset) {
  return {
    type: GET_STATIONS,
    payload: { limit, offset },
  }
}

export const INCREASE_OFFSET = 'INCREASE_OFFSET';
export const DECREASE_OFFSET = 'DECREASE_OFFSET';

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		78

Файл `epic.js`, який перехоплює Action об'єкти, та дозволяє асинхронно змінювати стан додатку

```
import { ofType, combineEpics } from "redux-observable";
import { bufferCount, empty, from, map, mergeAll, of, toArray } from "rxjs";
import { catchError, ignoreElements, mapTo, switchMap, timeout } from "rxjs/operators";
import { ajax } from 'rxjs/ajax';
import { addFavoriteStation, deleteFavorite, getStations, setFavorites, setShowConfirmationDialog } from "./actions";

import * as SQLite from 'expo-sqlite';

const db = SQLite.openDatabase('db.db');

function setStations(stations) {
  return {
    type: 'SET_STATIONS', payload: { stations }
  };
}

let stations = [];
const getStationsEpic = (action$, state$) => action$.pipe(
  ofType('GET_STATIONS'),
  switchMap((a) => {
    if (stations.length == 0 && !!a.payload.limit)
      return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/by-language/${state$.value.stationLanguage}`).pipe(
        map(response =>
          from(response.map(({ name, url_resolved }) => ({ name, url_resolved }))).filter((i) => i.url_resolved.slice(0, 5) == 'http:'))
          .pipe(
            bufferCount(a.payload.limit),
          )
        ),
        mergeAll(),
        toArray(),
        map((result) => {
          stations = result;
          return setStations(result[a.payload.offset / 10]);
        })
      )
    else if (a.payload.limit == 0)
      return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/by-language/${state$.value.stationLanguage}`).pipe(
        map(response => (setStations(response))))
    if (stations.length > (a.payload.offset / 10))
      return of(setStations(stations[a.payload.offset / 10]));

    return of(setShowConfirmationDialog(true));
  })
);
```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		79

```

    }
  ),
);

const setLanguageEpic = (action$, state$) => action$.pipe(
  ofType('SET_LANGUAGE'),
  switchMap((a) => {
    if (a.payload.limit > 0)
      return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/by-language/${state$.value.stationLanguage}`)
        .pipe(
          map(response =>
            from(response.map(({ name, url_resolved }) => ({ name, url_resolved })))
              .filter((i) => i.url_resolved.slice(0, 5) === 'http:')
                .pipe(
                  bufferCount(state$.value.limit),
                )
          ),
          mergeAll(),
          toArray(),
          map((result) => {
            stations = result;
            state$.value.offset = 0;
            return setStations(result[state$.value.offset]);
          })
        )
    else if (a.payload.limit === 0)
      return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/by-language/${state$.value.stationLanguage}`)
        .pipe(
          map(response => (setStations(response)))
        )

    return empty();
  })
);

const setDefaultLimit = (action$, state$) => action$.pipe(
  ofType('RESET_LIMIT'),
  switchMap((action) => {
    return ajax.getJSON(`https://de1.api.radio-browser.info/json/stations/bylanguage/${state$.value.stationLanguage}`)
      .pipe(
        switchMap(response => of(setStations(response.map(({ name, url_resolved }) => ({ name, url_resolved })))
          .filter((i) => i.url_resolved.slice(0, 5) === 'http:'))))
      ));
  ));

const setUrlEpic = (action$, state$) => action$.pipe(
  ofType('SET_URL'),
  switchMap((url) => {
    return ajax.get(`http://ztu.local/url?url=${url.url}`)
      .pipe(
        map(res => {
          if (!res.response) {
            throw new Error();
          }
        })
      )
  })
);

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		80

```

        return of([]);
      }),
      catchError(() => (empty()))
    )
  }));

const connectToAPEpic = (action$, state$) => action$.pipe(
  ofType('CONNECT_AP'),
  switchMap((params) => {
    return ajax.get(`http://ztu.local/connect?ssid=${params.payload.ssid}&password=${params.payload.password}`)
  })
);

const addToFavoriteEpic = (action$, state$) => action$.pipe(
  ofType('ADD_FAVORITE'),
  switchMap((f) => {
    db.transaction(txn => {
      txn.executeSql(
        `CREATE TABLE IF NOT EXISTS favoriteStations (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), url VARCHAR(50))`, [],
        () => console.log("table created successfully"),
        error => console.log("error on creating table " + error.message)),
      txn.executeSql(
        `INSERT INTO favoriteStations (name, url) VALUES (?, ?)`,
        [f.name, f.url_resolved],
        () => console.log(`${f.name} added successfully to favoriteStations`),
        error => console.log("error on adding favoriteStations " + error.message));
    });

    return of(addFavoriteStation(f.name, f.url_resolved));
  }),
);

const deleteFavoriteEpic = (action$, state$) => action$.pipe(
  ofType('DELETE_FAVORITE'),
  switchMap((f) => {
    db.transaction(txn => {
      txn.executeSql(
        `DELETE FROM favoriteStations WHERE name = ?`, [f.name],
        () => {
          console.log(`${f.name} deleted successfully from favoriteStations`);
          return of(deleteFavorite(f.name));
        },
        error => console.log("error on deleting favoriteStations " + error.message));
    });
    return of(deleteFavorite(f.name));
  }),
);

```

```

const getFavoriteEpic = (action$, state$) => action$.pipe(
  ofType('GET_FAVORITE'),
  switchMap((f) => {
    var results = [];
    db.transaction(txn => {
      txn.executeSql(
        `SELECT * FROM favoriteStations ORDER BY id DESC`,
        [],
        (sqlTxn, res) => {
          console.log(sqlTxn);
          console.log("favoriteStations retrieved successfully");
          let len = res.rows.length;

          if (len > 0) {
            for (let i = 0; i < len; i++) {
              let item = res.rows.item(i);
              results.push({ name: item.name, url: item.url_resolved
            });
          }
        }
      ),
      error => { console.log("error on getting categories " + error.mes-
sage); return; },
    );
  });
  return setFavorites(results);
}),
);
export default combineEpics(
  getStationsEpic,
  setLanguageEpic,
  setUrlEpic,
  connectToAPEpic,
  addToFavoriteEpic,
  getFavoriteEpic,
  deleteFavoriteEpic,
  setDefaultLimit,
);

```

Файл reducer.js, який використовується для управління станом

```

import { combineReducers } from "redux"
import { INCREASE_OFFSET, DECREASE_OFFSET, SET_STATIONS } from './actions';

const initialState = {
  offset: 0,
  stationAllAtOnce: false,
  stationLanguage: 'ukrainian',
  disabledOffsetButtons: false,
  limit: 10,
  stations: [],
  favoriteStations: [],
  showConfirmationDialog: false,
}

export default combineReducers({
  offset: offsetReducer,
  stations: stationsReducer,
  limit: limitReducer,
  availabilityOffsetButtons: availabilityOffsetButtonsReducer,
  stationLanguage: stationLanguageReducer,
  showConfirmationDialog: showConfirmationDialogReducer,
  favoriteStations: favoriteStationsReducer,
});

function favoriteStationsReducer(state = initialState.favoriteStations, action) {
  switch (action.type) {
    case 'ADD_FAV':
      if (!state.favoriteStations)
        return { favoriteStations: [...state, action.payload] }
      return { ...state, favoriteStations: [...state.favoriteStations, action.payload] }
    case 'SET_FAVORITES':
      return { ...state, favoriteStations: action.payload.stations }
    case 'DELETE_FAV':
      return {
        ...state, favoriteStations: state.favoriteStations.filter(item => item.name
        !== action.payload.name),
      }
      default: return state
    }
  }
}

function offsetReducer(state = initialState.offset, action) {
  switch (action.type) {
    case INCREASE_OFFSET:
      return state + 10;
    case DECREASE_OFFSET:
      return state - 10;
    case 'SET_DEFAULT_OFFSET':
      return initialState.offset;
  }
  return state;
}

```

```

function stationsReducer(state = initialState.stations, action) {
  switch (action.type) {
    case SET_STATIONS: {
      if (action.payload.stations)
        return action.payload.stations
      }
    case 'RESET_STATIONS': {
      return { ...state, stations: null }
      }
  }
  return state;
}

function stationLanguageReducer(state = initialState.stationLanguage, action) {
  switch (action.type) {
    case 'SET_LANGUAGE':
      return action.payload.language;
  }
  return state;
}

function showConfirmationDialogReducer(state = initialState.showConfirmationDialog,
action) {
  switch (action.type) {
    case 'SHOW_CONFIRMATION_DIALOG':
      return { showConfirmationDialog: action.payload }
  }
  return state;
}

function limitReducer(state = initialState.limit, action) {
  switch (action.type) {
    case 'SET_DEFAULT_LIMIT':
      return 10
    case 'RESET_LIMIT':
      return 0;
  }
  return state;
}

function availabilityOffsetButtonsReducer(state = initialState.disabledOffsetButtons,
action) {
  switch (action.type) {
    case 'SET_ENABLE_OFFSET_BUTTONS':
      return { ...state, disabledOffsetButtons: true }
    case 'SET_DISABLE_OFFSET_BUTTONS':
      return { ...state, disabledOffsetButtons: false }
  }
  return state;
}

```

Файл вбудованої програми, для програмування мікроконтролера

```

#include <ESP8266NetBIOS.h>
#include <Arduino.h>
#include <ESP8266WebServer.h>
#include <ESP8266WiFi.h>
#include <EEPROM.h>
#include "AudioFileSourceHTTPStream.h"
#include "AudioFileSourceBuffer.h"
#include "AudioGeneratorMP3.h"
#include "AudioOutputI2SNoDAC.h"

#define HOSTNAME "webserver"

ESP8266WebServer server(80);

AudioGeneratorMP3 *mp3;
AudioFileSourceHTTPStream *file;
AudioFileSourceBuffer *buff;
AudioOutputI2SNoDAC *out;
// char *URL;
bool audioFlag = false;
String nameAP;
String playingnow;

bool testWifi(void);
void launchWeb(void);
void checkWhetherConnected(String ssid, String password, bool fromMemory = false);

void setup() {
  Serial.begin(115200);
  EEPROM.begin(512);

  String esid;
  for (int i = 0; i < 32; ++i)
  {
    esid += char(EEPROM.read(i));
  }

  String epass = "";
  for (int i = 32; i < 96; ++i)
  {
    epass += char(EEPROM.read(i));
  }
  Serial.println(esid);
  Serial.println(epass);

  WiFi.begin(esid.c_str(), epass.c_str());
  nameAP = esid.c_str();
  checkWhetherConnected(esid.c_str(), epass.c_str(), true);
  if (!NBNS.begin("ztu")) { // ztu.local
    Serial.println("Error setting up MDNS responder!");
  }
  server.onNotFound(handleShowApList);
  server.on("/apList", handleShowApList);
  server.on("/connect", handleConnect);
  server.on("/url", handleURL);

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
Зм.	Арк	№ докум	Підпис	Дата		85

```

server.on("/apname", [](){server.send(200, "text/html", nameAP)});
server.on("/playingnow", [](){server.send(200, "text/html", playingnow)});

server.begin();
}
void checkWhetherConnected(String ssid, String password, bool fromMemory){
  if (testWifi())
  {
    // server.on("/apList", handleShowApList);
    WiFi.softAPdisconnect(true);

    Serial.println("inside");
    if(fromMemory)
      return;

    Serial.println("clearing eeprom");
    for (int i = 0; i < 96; ++i) { //clearing eeprom
      EEPROM.write(i, 0);
    }

    for (int i = 0; i < ssid.length(); ++i)
    {
      EEPROM.write(i, ssid[i]);
      Serial.print("Wrote: ");
      Serial.println(ssid[i]);
    }
    Serial.println("writing eeprom pass:");
    for (int i = 0; i < password.length(); ++i)
    {
      EEPROM.write(32 + i, password[i]);
      Serial.print("Wrote: ");
      Serial.println(password[i]);
    }
    EEPROM.commit();
  }
  else
  {
    Serial.println("outside");
    launchWeb();
  }
}
bool testWifi(void)
{
  int c = 0;
  Serial.println("Waiting for Wifi to connect");
  while ( c < 20 ) {
    Serial.println(WiFi.status());
    Serial.println(c);
    if (WiFi.status() == WL_CONNECTED)
    {
      return true;
    }
    delay(1000);
    Serial.print("*");
    c++;
  }
  return false;
}
void launchWeb()

```

```

{
    Serial.println("");
    if (WiFi.status() != WL_CONNECTED)
        WiFi.softAP("ZTU", "password");
    else
        Serial.println("WiFi connected");

    // server.on("/connect", handleConnect);
}

void handleShowApList() {;
    int ssidNum = WiFi.scanNetworks();
    if( ssidNum > 0 ){
        String resultJson = "{ \"Points\": [";

        for (int i = 0; i < ssidNum; i++) {
            resultJson += "\"" + WiFi.SSID(i) + "\"";
            if (i != ssidNum - 1)
                resultJson += ",";
        };
        resultJson += "] }";

        server.send(200, "application/json", resultJson);
    } else

        server.send(200, "application/json", "{\"No available points\" } ");
}

void handleConnect() {
    String ssid = server.arg("ssid");
    String password = server.arg("password");

    WiFi.begin(ssid, password);

    checkWhetherConnected(ssid, password);
}

void handleURL() {
    audioFlag = false;
    char charBuf[50];
    String urlString = server.arg("url");
    Serial.printf(stringUrl.c_str());
    urlString.toCharArray(charBuf, 50);

    const char *URL = charBuf;

    file = new AudioFileSourceHTTPStream(URL);
    buff = new AudioFileSourceBuffer(file, 2048);
    out = new AudioOutputI2SNoDAC();
    mp3 = new AudioGeneratorMP3();
    mp3->begin(buff, out);

    audioFlag = true;
    playingnow = urlString.c_str();

    server.send(200, "text/html", nameAP);
}

void loop() {
    server.handleClient();
}

```

```

static int lastms = 0;
if(audioFlag){
if (mp3->isRunning()) {
    if (millis()-lastms > 1000) {
        lastms = millis();
        Serial.printf("Running for %d ms...\n", lastms);
        Serial.flush();
    }
    if (!mp3->loop()) mp3->stop();
} else {
    delete file;
    delete mp3;
    audioFlag = false;
} }}

```

					КІ.КР.Б -123 – 09 - ПЗ	Арк
						88
Зм.	Арк	№ докум	Підпис	Дата		