BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC ĐÔNG Á

----📖-----

Lập trình di động 2

Đề tài:

ĐÓNG MỞ CỔNG

GVHD: **ThS. Tạ Quốc Ý**
SVTH: **Lê Vũ Trọng Đức**
Lương Quang Hùng
Đoàn Ngọc Sơn
Dương Bá Thắng
Trần Đăng Quân
Lớp: **ST21A1A**

Đà Nẵng, tháng 04/2024

MỤC LỤC

Chương 1. TỔNG QUAN VỀ REACT NATIVE 3
1.1. Giới thiệu 3
1.2. Ưu điểm của React Native 3
1.3. Nhược điểm của React Native 4
Chương 2. CÀI ĐẶT REACT NATIVE 5
Chương 3. Xây dựng 6

Chương 1. TỔNG QUAN VỀ REACT NATIVE

1.1. Giới thiệu

React Native là một framework do Facebook phát triển, cho phép tạo các ứng dụng di động bằng các công nghệ Web quen thuộc mà hiệu suất vẫn rất tốt. Ngôn ngữ lập trình chính được sử dụng trong React Native là Javascript. Do đó, bạn sẽ không cần phải học nhiều ngôn ngữ lập trình, cũng không cần phải tìm hiểu hệ sinh thái từng nền tảng. Đặc biệt, nếu bạn đang là một nhà phát triển ứng dụng web, muốn nhanh chóng xây dựng một ứng dụng mobile thì React native là giải pháp tốt nhất mà bạn nghĩ tới lúc này. Như cái tên của nó, React Native được xây dựng dựa trên thư viện web nổi tiếng ReactJS.

1.2. Ưu điểm của React Native

- **Thời gian phát triển ứng dụng nhanh**: Điều mà các nhà phát triển "chót yêu" React native đó là họ có thể tái sử dụng và tái chế các component trước đó mà họ đã xây dựng cho web. Họ chỉ cần copy và paste, có khi code chạy luôn mà không cần phải sửa gì. Nói vậy thôi, nhưng về cơ bản, code của React Native giống với ReactJS, nên bạn sẽ tận dụng được tối đa code đã làm.
- **Chi phí phát triển ứng dụng rẻ**: Cái này quá rõ ràng rồi. Thay vì bạn sẽ phải bỏ công sức viết code riêng cho từng nền tảng. Giờ đây, bạn chỉ cần viết code một lần duy nhất, bằng một ngôn ngữ duy nhất - Javascript. Giảm chi phí training cho các thành viên trong dự án.
- **Một code base cho nhiều nền tảng**: Với đặc điểm, chỉ viết một code base, bạn có thể release cho cả Android và iOS. Bạn vừa tiết kiệm được công sức phát triển, vừa nhanh phát hành sản phẩm ra thị trường.

- **Dễ dàng kết hợp với native code**: Một số tính năng đặc thù cần phải tương tác nhiều với phần cứng của thiết bị như: tính năng camera, tính năng Bluetooth… bạn sẽ cần phải sử dụng code native (Java/Kotlin hoặc Swift). React Native cho phép bạn code native ngay chính trong project và có thể tương tác với code Javascript bên ngoài.
- **Tính năng Hot Reloading/Live reloading**: Là tính năng cho phép cập nhật ứng dụng ngay sau khi chỉnh sửa mã nguồn mà không cần phải build lại toàn bộ. Nó giữ cho ứng dụng hoạt động bình thường và được cập nhật thêm những thay đổi theo thời gian thực. Điều này giúp rút ngắn thời gian chờ đợi rất nhiều mỗi khi chỉnh sửa mã nguồn.
- **Hệ sinh thái React lớn**: Vì React Native được xây dựng dựa trên thư viện ReactJS nên nó kế thừa khá nhiều thư viện, cũng như cộng đồng lập trình viên. Bạn dễ dàng tìm thấy hàng trăm, hàng ngàn thư viện, công cụ hỗ trợ cho việc xây dựng ứng dụng của bạn. Đặc biệt, với sự "chống lưng" của ông lớn công nghệ Facebook, bạn hoàn toàn yên tâm rằng React Native sẽ còn được phát triển dài dài.

1.3. Nhược điểm của React Native

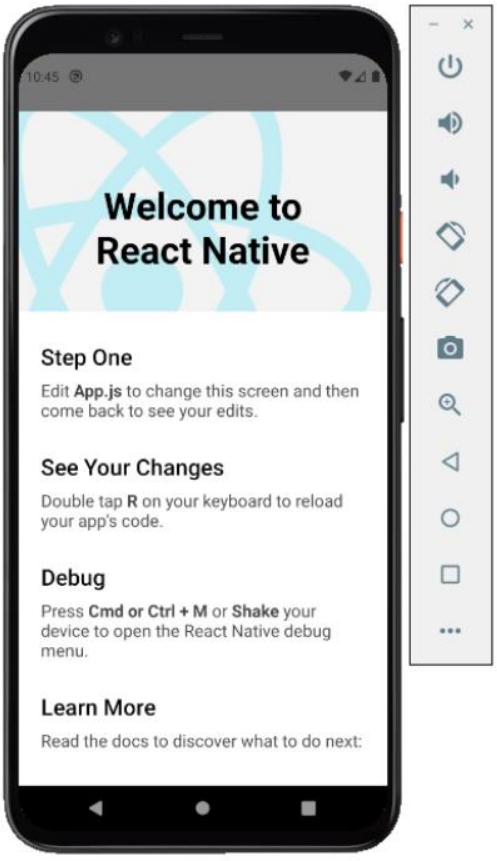
- **Hiệu năng vẫn kém hơn so với Native một xíu**: Mặc dù React Native đã được tối ưu để có thể can thiệp tới code native của hệ thống. Với React Native, nó cho phép bạn render các View bằng native API, nó gọi tới chính SDK tương ứng từng nền tảng để build ứng dụng. Tuy nhiên, dù gì nó vẫn phải chạy qua một tầng trung gian (là JS Runtime) nên hiệu năng chắc chắn sẽ không thể bằng các ứng dụng native được.

- **Thiết kế không hiệu quả**: Nếu ứng dụng của bạn có thiết kế phức tạp, coi tương tác nâng cao với người dùng là một trong những lợi thế kinh doanh thì bạn nên chọn giải pháp phát triển ứng dụng native.
- **Cập nhật version quá nhanh**: Nói chung cái gì nhanh quá cũng không tốt. Việc cập nhật version quá nhanh nói lên rằng React Native vẫn còn nhiều bugs, cần phải cải thiện nhiều. Chưa kể mỗi khi React Native nâng cấp, ứng dụng của bạn cũng cần xem xét có nên nâng cấp theo không, mà việc nâng cấp khi ứng dụng đang chạy ổn định và đã phát hành ra thị trường luôn là một công việc mệt mỏi, vì bạn sẽ phải test lại rất nhiều.

Chương 2. CÀI ĐẶT REACT NATIVE

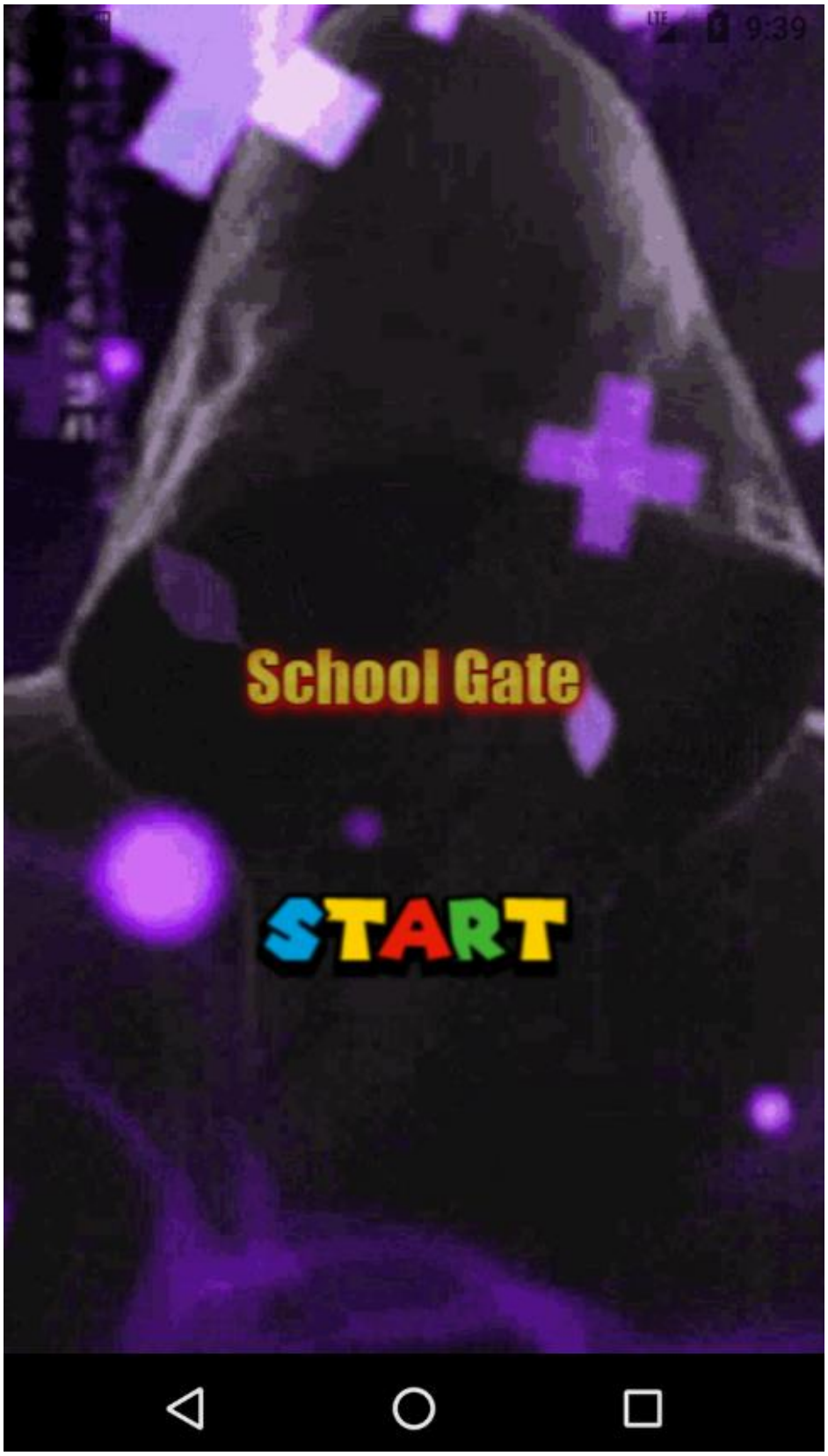
Mở Terminal sau đó thực hiện các bước:

- B1: npm install -g create-react-native-app
- B2: create-react-native-app <name>
- B3: npm start hoặc yarn android ,yarn ios,yarn web
- B4: npm install -g react-native-cli
- B5: npx react-native run-android



Chương 3. Xây dựng đồ án

3.1. Màn hình Start



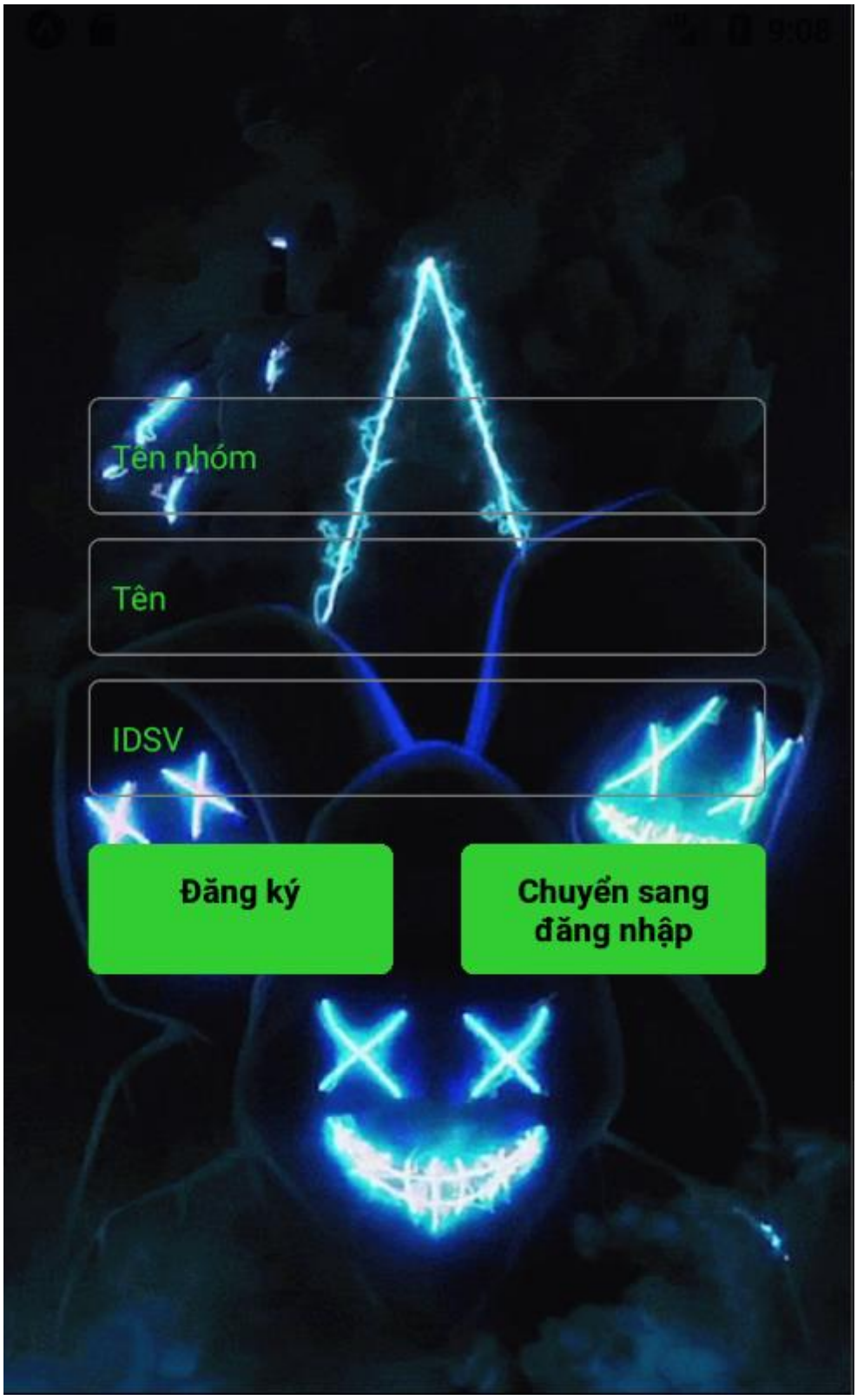
```
import React, { useEffect, useRef, useState } from 'react';
import { ImageBackground, StyleSheet, Animated, Image, TouchableOpacity } from
'react-native';
import SignInSignUp from './SignInSignUp';
import { IndexStyles } from './styleApp/styles';
const bgImage = require('../assets/mang-hinh.gif');
const textImage = require('../assets/index.png');
const startImage = require('../assets/start.png');
const styles = IndexStyles;
const Introduce = () => {
    const [screen, setScreen] = useState('Index');
    if (screen === 'SignInSignUp') {return <SignInSignUp />;}
    return <Index onNavigate={() => setScreen('SignInSignUp')} />;
};

const Index = ({ onNavigate }) => {
    const opacity = useRef(new Animated.Value(0)).current;
    useEffect(() => {
        Animated.loop(
            Animated.sequence([
                Animated.timing(opacity, {
                    toValue: 1,
                    duration: 2500,
                    useNativeDriver: true,
                }),
                Animated.timing(opacity, {
                    toValue: 0,
                    duration: 2500,
                    useNativeDriver: true,
                }),
            ]),
        ).start();
    }, []);
    return (
        <ImageBackground source={bgImage} style={styles.container}>
            <Animated.Image
                source={textImage}
                style={{
                    ...styles.overlayImage,
                    opacity: opacity,
                }}
            />
            <TouchableOpacity onPress={onNavigate}>
                <Image source={startImage} style={styles.buttonImage}/>
            </TouchableOpacity>
        </ImageBackground>
    );
};
export default Introduce;
```

Tạo ra một màn hình giới thiệu (introduction screen). Màn hình giới thiệu là màn hình đầu tiên mà người dùng thấy khi mở ứng dụng lần đầu tiên. Mục đích của màn hình giới thiệu là chào mừng người dùng và cung cấp một số thông tin cơ bản về ứng dụng.

Cụ thể, đoạn mã này có các chức năng sau:

- **Hiển thị Hình ảnh nền và Văn bản chồng lớp (Overlay Text)**:
    - Sử dụng **ImageBackground** để hiển thị một hình ảnh làm nền cho màn hình giới thiệu. Hình ảnh nền này có thể là một hình ảnh liên quan đến nội dung của ứng dụng hoặc là một hình ảnh đơn giản để tạo không gian phông nền cho các phần khác của màn hình.
    - Sử dụng **Animated.Image** để hiển thị một hình ảnh chồng lớp (overlay image) hoặc văn bản chồng lớp lên hình ảnh nền. Trong trường hợp này, hình ảnh hoặc văn bản này được làm mờ (opacity) theo một chuỗi hoạt ảnh để tạo hiệu ứng chuyển động hoặc thu hút sự chú ý của người dùng.
- **Nút Bắt đầu (Start Button)**:
    - Sử dụng một **TouchableOpacity** bao quanh một hình ảnh để tạo ra một nút bắt đầu hoặc một điểm truy cập vào ứng dụng. Khi người dùng nhấn vào nút này, họ có thể được dẫn đến một màn hình khác trong ứng dụng hoặc thực hiện một hành động cụ thể.
- **Chuyển hướng và Xử lý Sự kiện (Navigation and Event Handling)**:
    - Sử dụng hook **useState** để quản lý trạng thái của màn hình (**screen**). Khi người dùng nhấn vào nút bắt đầu, trạng thái của màn hình được thay đổi để chuyển đến màn hình **SignInSignUp**.
    - Sử dụng prop **onNavigate** để truyền hàm xử lý sự kiện chuyển hướng từ màn hình giới thiệu tới màn hình **SignInSignUp**.

3.2. Màn hình đăng nhập, đăng ký


Tên nhóm
Tên
IDSV
Đăng ký
Chuyển sang đăng nhập

```
import { ImageBackground, TouchableOpacity, Text, View, TextInput } from
"react-native";
import React, { useState } from "react";
import background from "../assets/background.gif";
import axios from 'axios';
import { SignInSignUpStyle } from "./styleApp/styles";
import Home from "./homeOne";

const styles = SignInSignUpStyle;

const SignInSignUp = () => {
    const [isLogin, setIsLogin] = useState(true);
    const [group, setGroup] = useState('');
    const [username, setUsername] = useState('');
    const [idsv, setIdsv] = useState('');
    const [isLoggedIn, setIsLoggedIn] = useState(false);
//API Post
    const handlePress = async () => {
        console.log('Dữ liệu gửi đến server:', { group, username, idsv });
        const url = isLogin ? 'http://192.168.170.25:3000/login' :
'http://192.168.170.25:3000/signup';
        const data = isLogin ? { username, idsv } : { group, username, idsv };
        try {
            const response = await axios.post(url, data);
            console.log(response.data);
            if (response.data === 'Đăng nhập thành công!') {
                setIsLoggedIn(true);
            }
        } catch (error) {
            console.error(error);
        }
    }

    if (isLoggedIn) {
        return <Home />;
    }
    return (
        <ImageBackground source={background} style={styles.container}>
            <View style={styles.buttonContainer}>
                {!isLogin && (
                    <TextInput
                        style={styles.input}
                        placeholder="Tên nhóm"
                        placeholderTextColor="#32CD32"
                        value={group}
                        onChangeText={setGroup}
                    />
                )}
                <TextInput
```

```
                    style={styles.input}
                    placeholder="Tên"
                    placeholderTextColor="#32CD32"
                    value={username}
                    onChangeText={setUsername}
                />
                <TextInput
                    style={styles.input}
                    placeholder="IDSV"
                    placeholderTextColor="#32CD32"
                    value={idsv}
                    onChangeText={setIdsv}
                />
                <View style={styles.buttonRow}>
                    <TouchableOpacity
                        onPress={handlePress}
                        style={styles.button}>
                        <Text style={styles.buttonText}>
                            {isLogin ? "Đăng nhập" : "Đăng ký"}
                        </Text>
                    </TouchableOpacity>
                    <TouchableOpacity
                        onPress={() => setIsLogin(!isLogin)}
                        style={styles.button}>
                        <Text style={styles.buttonText}>
                            {isLogin ? "Chuyển sang đăng ký" : "Chuyển sang
đăng nhập"}
                        </Text>
                    </TouchableOpacity>
                </View>
            </View>
        </ImageBackground>
    );
}

export default SignInSignUp;
```

Tạo một màn hình đăng nhập và đăng ký trong ứng dụng di động sử dụng React Native. Bằng cách sử dụng các thành phần như **TextInput**, **TouchableOpacity**, và **ImageBackground**, cung cấp một giao diện người dùng cho người dùng để nhập thông tin đăng nhập và đăng ký.

Cụ thể, nó thực hiện các chức năng sau:

- **Hiển thị Giao diện Người dùng** :

- Sử dụng **ImageBackground** để hiển thị một hình ảnh nền cho màn hình đăng nhập và đăng ký.
- Sử dụng các **TextInput** để cho phép người dùng nhập thông tin như tên nhóm (nếu ở chế độ đăng ký), tên người dùng và ID sinh viên.
- Sử dụng các **TouchableOpacity** để tạo nút đăng nhập và nút chuyển đổi giữa chế độ đăng nhập và đăng ký.
- Sử dụng **Text** để hiển thị các nhãn như "Đăng nhập" hoặc "Đăng ký".

- **Quản lý Trạng thái (State Management)**:
  - Sử dụng hook **useState** để quản lý các trạng thái như **isLogin** (xác định xem người dùng đang trong quá trình đăng nhập hay đăng ký), **group**, **username**, **idsv** (các trường thông tin nhập vào), và **isLoggedIn** (xác định xem người dùng đã đăng nhập thành công hay chưa).
- **Xử lý Sự kiện và Gửi Yêu cầu API (Event Handling and API Request)**:
  - Khi người dùng nhấn vào nút đăng nhập hoặc đăng ký, sự kiện được xử lý bởi hàm **handlePress**.
  - Hàm **handlePress** tạo yêu cầu HTTP (POST) đến máy chủ với các thông tin đăng nhập hoặc đăng ký được cung cấp bởi người dùng thông qua **axios**.
  - Sau khi nhận được phản hồi từ máy chủ, nếu đăng nhập thành công, người dùng sẽ được chuyển đến màn hình **Home**.
- **Xuất khẩu (Export)**:
  - Thành phần **SignInSignUp** được xuất ra như là một thành phần mặc định của tệp, có thể được sử dụng và nhúng vào các thành phần khác trong ứng dụng.

3.3. Kết nối MongoDB

```
//nodeJS
const mongoose = require('mongoose');
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
app.use(bodyParser.json());
app.use(cors());
mongoose.connect('mongodb://localhost:27017/AppArduino', {useNewUrlParser:
true, useUnifiedTopology: true});
mongoose.connection.on('error', err => {
  console.error('error connection to MongoDB:', err);
});
mongoose.connection.once('open', () => {
  console.log('connection to MongoDB good!');
});
const userSchema = new mongoose.Schema({
  group: String,
  username: String,
  idsv: String,
});
const User = mongoose.model('User', userSchema);
app.post('/signup', async (req, res) => {
    console.log('Dữ liệu nhận được từ client:', req.body);
  const { group, username, idsv } = req.body;
  const user = new User({ group, username, idsv });
  await user.save();
  res.send('Đăng ký thành công!');
});
app.post('/login', async (req, res) => {
  const { username, idsv } = req.body;
  const user = await User.findOne({ username, idsv });
  if (user) {
    res.send('Đăng nhập thành công!');
  } else {
    res.send('Tên đăng nhập hoặc IDSV không đúng!');
  }
});
app.listen(3000, () => console.log('http://localhost:3000'));
```

Đoạn mã này là một ứng dụng Node.js sử dụng Express và MongoDB để tạo một API đăng ký và đăng nhập người dùng.

- **Kết nối Cơ sở dữ liệu (Database Connection)**:
    - Sử dụng **mongoose** để kết nối tới cơ sở dữ liệu MongoDB chạy trên **localhost** ở cổng **27017**.
    - Mongoose là một thư viện Node.js giúp tương tác với cơ sở dữ liệu MongoDB một cách dễ dàng và linh hoạt.
- **Khởi tạo ứng dụng Express (Express Application Setup)**:
    - Tạo một ứng dụng Express bằng cách gọi hàm **express()**.
    - Sử dụng **body-parser** để phân tích các yêu cầu HTTP gửi đến máy chủ từ client dưới dạng JSON.
    - Sử dụng **cors** để cho phép truy cập tới các tài nguyên từ các miền khác nhau trên internet.
- **Định nghĩa Schema và Model (Schema and Model Definition)**:
    - Định nghĩa một Schema cho đối tượng người dùng bao gồm các trường như **group**, **username**, và **idsv**.
    - Sử dụng **mongoose.model()** để tạo một Model từ Schema đã định nghĩa. Model này sẽ được sử dụng để thao tác với các bản ghi trong cơ sở dữ liệu.
- **API Đăng ký và Đăng nhập (Signup and Login API)**:
    - Tạo hai endpoints **POST** tương ứng cho đăng ký (**/signup**) và đăng nhập (**/login**).
    - Khi nhận yêu cầu đăng ký, ứng dụng sẽ tạo một bản ghi mới trong cơ sở dữ liệu với thông tin người dùng được gửi từ client.
    - Khi nhận yêu cầu đăng nhập, ứng dụng sẽ kiểm tra xem thông tin người dùng có khớp với bất kỳ bản ghi nào trong cơ sở dữ liệu hay không. Nếu tìm thấy, nó sẽ trả về thông báo "Đăng nhập thành công!", ngược lại sẽ trả về thông báo "Tên đăng nhập hoặc IDSV không đúng!".
- **Lắng nghe các yêu cầu HTTP (Listen for HTTP Requests)**:
    - Ứng dụng sẽ lắng nghe các yêu cầu tới cổng **3000** trên localhost.

- Khi ứng dụng đã được khởi động, thông điệp "http://localhost:3000" sẽ được in ra console.

3.4. Kết nối Esp32

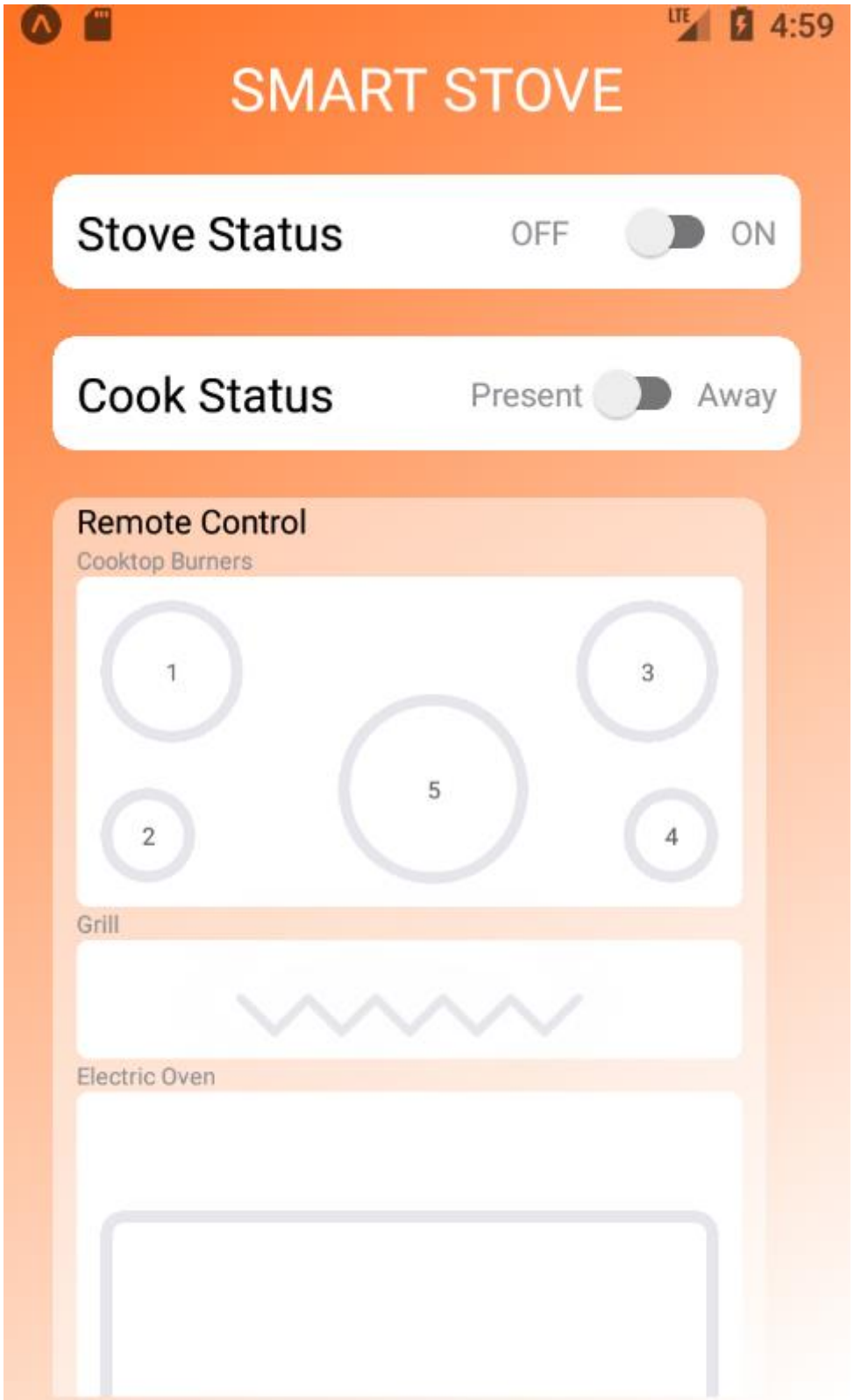
```
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>

//  Wi-Fi
const char* ssid = "Tên wifi";
const char* password = "mật khẩu wifi";

// Tạo một đối tượng WebServer (port 80)
WebServer server(80);

// GPIO pin mà LED được kết nối
const int ledPin = 2;

void handleToggle() {
  if (server.hasArg("plain")) {
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, server.arg("plain"));
    bool state = doc["state"];
    digitalWrite(ledPin, state ? HIGH : LOW);
    server.send(200, "application/json", "{\"status\":\"success\"}");
  } else {
    server.send(400, "application/json", "{\"status\":\"error\"}");
  }
}
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  // Kết nối với Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected.");
  Serial.println(WiFi.localIP());
  // Định nghĩa các route cho web server
  server.on("/toggle", HTTP_POST, handleToggle);
  // Bắt đầu server
  server.begin();
}
void loop() {
```

```
  server.handleClient();
}
```

Đây là một số điểm chính của mã:

- **Kết nối Wi-Fi**:
    - Sử dụng thư viện **WiFi.h** để kết nối với mạng Wi-Fi thông qua tên mạng và mật khẩu đã cung cấp.
    - Trong hàm **setup()**, Arduino cố gắng kết nối với mạng Wi-Fi và in ra địa chỉ IP cục bộ khi kết nối thành công.
- **Tạo Máy chủ Web**:
    - Sử dụng thư viện **WebServer.h** để tạo một đối tượng **WebServer** lắng nghe trên cổng 80.
    - Trong hàm **setup()**, định nghĩa một route **/toggle** nhận yêu cầu POST, và khi nhận được yêu cầu, nó gọi hàm **handleToggle()**.
- **Xử lý Yêu cầu**:
    - Trong hàm **handleToggle()**, Arduino kiểm tra xem yêu cầu có chứa dữ liệu dạng JSON không. Nếu có, nó phân tích cú pháp JSON để lấy trạng thái của thiết bị.
    - Sau đó, nó sử dụng trạng thái này để bật hoặc tắt đèn được kết nối với pin GPIO.
    - Cuối cùng, nó gửi một phản hồi JSON với trạng thái "success" hoặc "error".
- **Chạy vòng lặp**:
    - Trong hàm **loop()**, Arduino liên tục xử lý các yêu cầu từ client bằng cách gọi **server.handleClient()**. Điều này giữ cho máy chủ luôn sẵn sàng để nhận và xử lý yêu cầu từ client.

3.5. Màn hình Figma



```
import React, { useState } from 'react';
import { View, Switch, Text, StyleSheet } from 'react-native';

export const ButtonOne = () => {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState);

  return (
    <View style={styles.container}>
      <Text style={styles.text}>OFF</Text>
      <Switch
        trackColor={{ false: "#767577", true: "#FF7121" }}
        thumbColor={isEnabled ? "#f4f3f4" : "#f4f3f4"}
        ios_backgroundColor="#696868"
        onValueChange={toggleSwitch}
        value={isEnabled}/>
      <Text style={styles.textON}>ON</Text>
    </View>
  );}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center",
    flexDirection: 'row',
  },
  text: {
    marginRight: 20,
    color: '#8E8E93',},
  textON:{
    color: '#8E8E93',},
  switchSytle: {
    marginRight: 20,
  },});
export const ButtonTwo = () =>{
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState);
  return (
    <View style={styles.container}>
      <Text style={styles.textON}>Present</Text>
      <Switch
        trackColor={{ false: "#767577", true: "#FFE607" }}
        thumbColor={isEnabled ? "#f4f3f4" : "#f4f3f4"}
        ios_backgroundColor="#696868"
        onValueChange={toggleSwitch}
        value={isEnabled}/>
      <Text style={styles.textON}>Away</Text>
    </View>
  );}
```

```
import { Text, StyleSheet, View } from 'react-native';
import { BoxContentOne, BoxContentTwo, BoxContentThree } from
'./itemContent1';
import { ImageOne } from '../public/images/images';

function ItemContentScreen() {
    return (
        <View style={styles.container}>
            <Text>Remote Control</Text>
            <Text style={styles.textTitle}>Cooktop Burners</Text>
            <BoxContentOne/>
            <Text style={styles.textTitle}>Grill</Text>
            <BoxContentTwo images={<ImageOne />} />
            <Text style={styles.textTitle}>Electric Oven</Text>
            <BoxContentThree/>

        </View>
    );
}

const styles = StyleSheet.create({
    container: {
        marginTop: 20,
        paddingHorizontal: 10,
        flexDirection: 'column',
        alignItems: 'flex-start',
        justifyContent: 'flex-start',
        backgroundColor: '#FFFFFF80',
        borderRadius: 10,
        width: 300,
        height: 450,
    },
    textTitle:{
        fontSize: 10,
        color:'#8E8E93',
    }
});

export default ItemContentScreen;
```

Giải Các Phần Code Cơ Bản:

Phần này CSS

```
const styles = StyleSheet.create({
    container: {
        marginTop: 20,
        paddingHorizontal: 10,
        flexDirection: 'column',
        alignItems: 'flex-start',
        justifyContent: 'flex-start',
        backgroundColor: '#FFFFFF80',
        borderRadius: 10,
        width: 300,
        height: 450,
    },
    textTitle:{
        fontSize: 10,
        color:'#8E8E93',
    }
});
```

Phần Này Là Phần Thư Viện React Native

```
import React, { useState } from 'react';
import { View, Switch, Text, StyleSheet } from 'react-native';
```

Phần Này Là Phần View(Màng hình giao diện.)

```
export const ButtonOne = () => {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState);

  return (
    <View style={styles.container}>
      <Text style={styles.text}>OFF</Text>
      <Switch
        trackColor={{ false: "#767577", true: "#FF7121" }}
        thumbColor={isEnabled ? "#f4f3f4" : "#f4f3f4"}
        ios_backgroundColor="#696868"
        onValueChange={toggleSwitch}
        value={isEnabled}/>
      <Text style={styles.textON}>ON</Text>
    </View>
  );}
```

Phần này lấy Hình ảnh tại dự án được thêm vào và gọi bằng đường dẫn.

```
import { ImageOne } from '../public/images/images';
```