

## **Cross Platform App Development Lab Experiment No. 1**

**Aim:** Designing user interface and wireframe for multiple layouts using Figma.

### **Objectives:**

1. Understanding the concept of wireframing and its role in the design process.
2. To get an overview of Figma, its features, and its relevance in UI/UX design.

### **Theory:**

- Wireframing is the initial step in creating user- friendly interfaces. Figma is a popular tool for this purpose due to its collaboration features and adaptability to different devices.
- About my mobile applications which is for customers and scrap dealer to easily report and track scrap materials.
- The scrap management app helps people who want to sell scrap and those who want to buy it to talk and make deals easily. It has features like creating accounts, showing what scrap is available.

### **Requirements:**

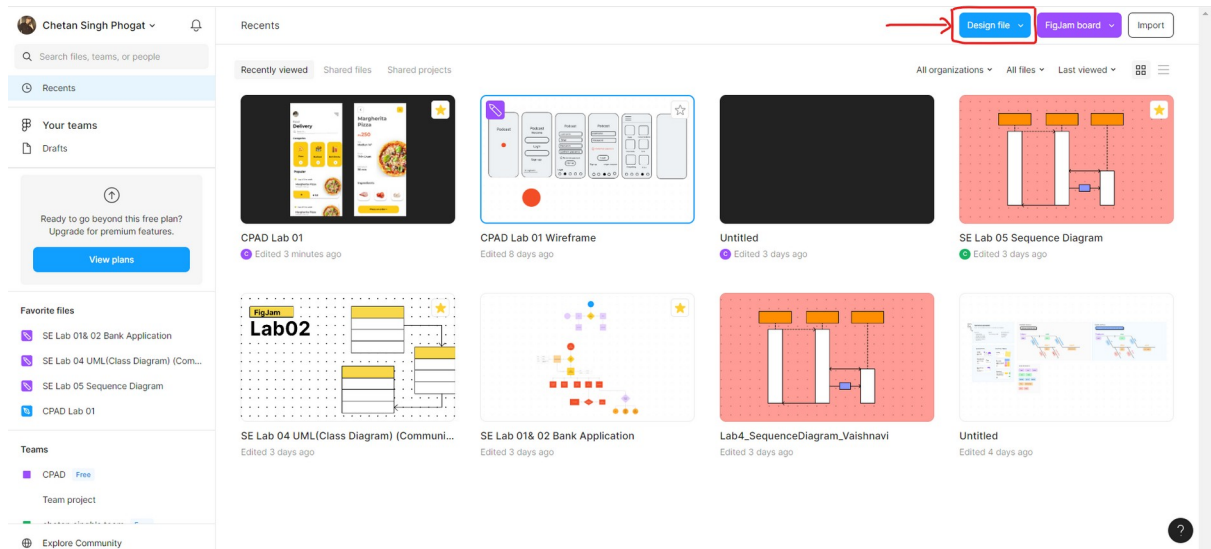
1. Computer with internet access.
2. A Figma account.
3. Word processing software for creating the lab report.

### **Tools:**

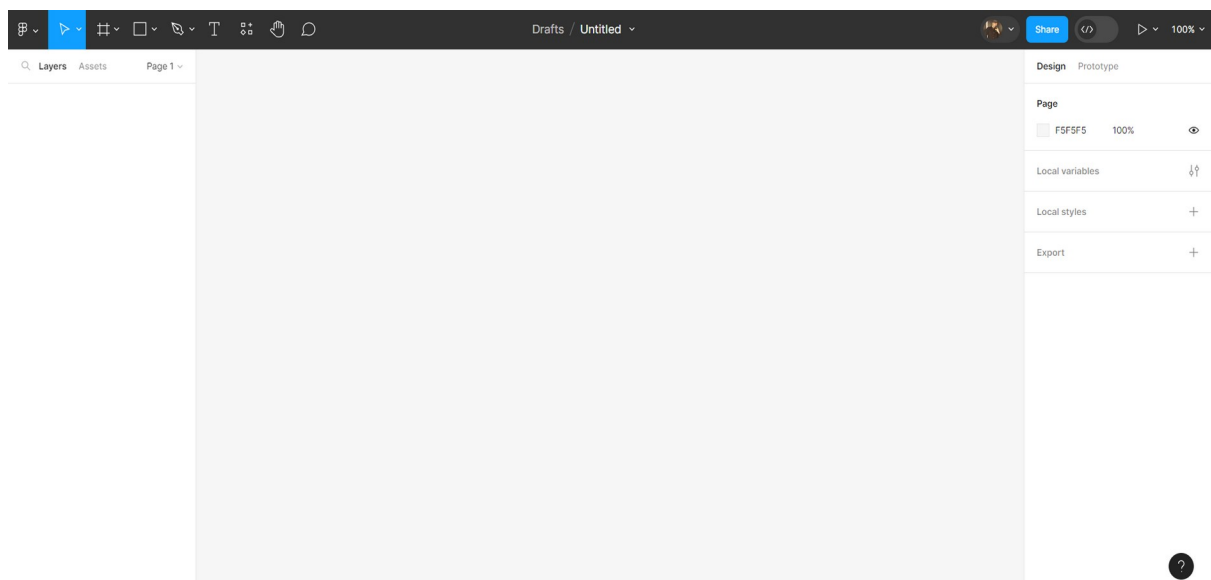
- **Figma:** Figma is a web-based design tool that allows users to create interactive user interface prototypes. It enables real-time collaboration and can be accessed from anywhere and on any device.

## Steps:-

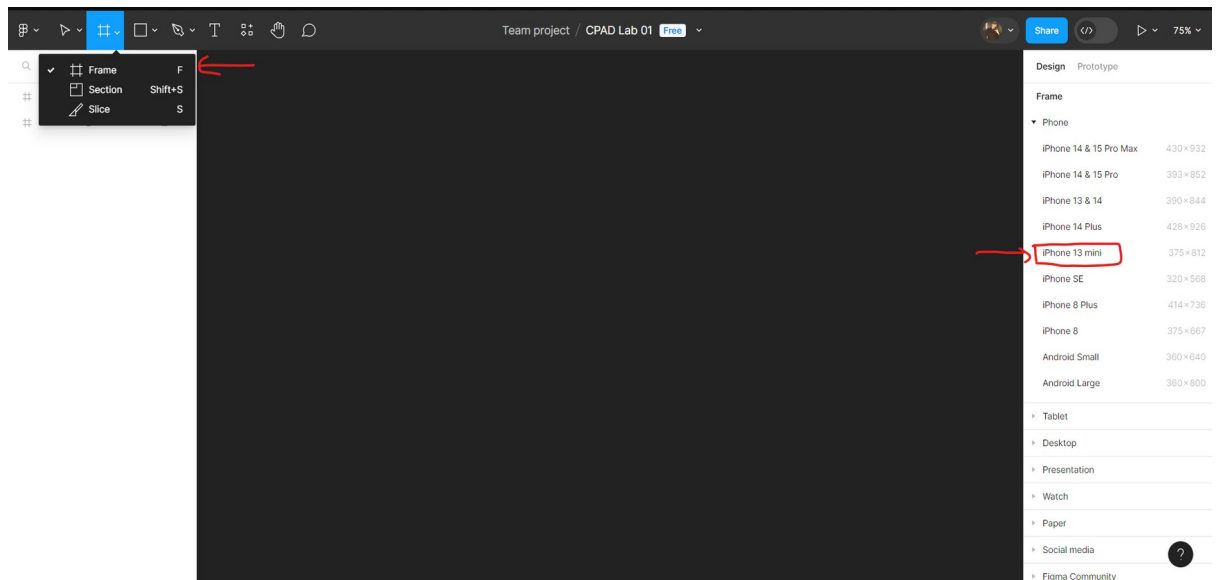
### Step 1: Selecting design file



### Step 2: Getting familiar with the UI of design file in Figma.



### Step 3: Selecting frame for App



### Step 4: Creating second layer of App

Register

### Register

Create a new Account

Email

Password

Phone No

Register

Already have a account? Sign in

Sign Page

### Sign In

Sign In to your account

chetansingh1230@gmail.com

.....

Login

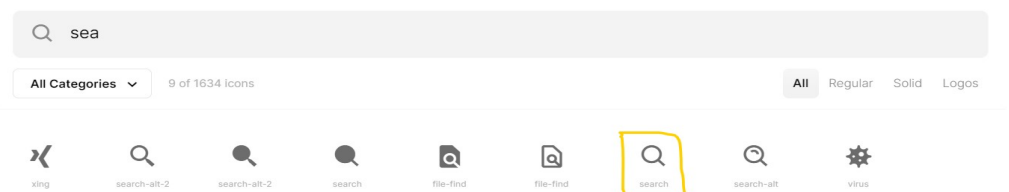
Don't have a account? Sign Up

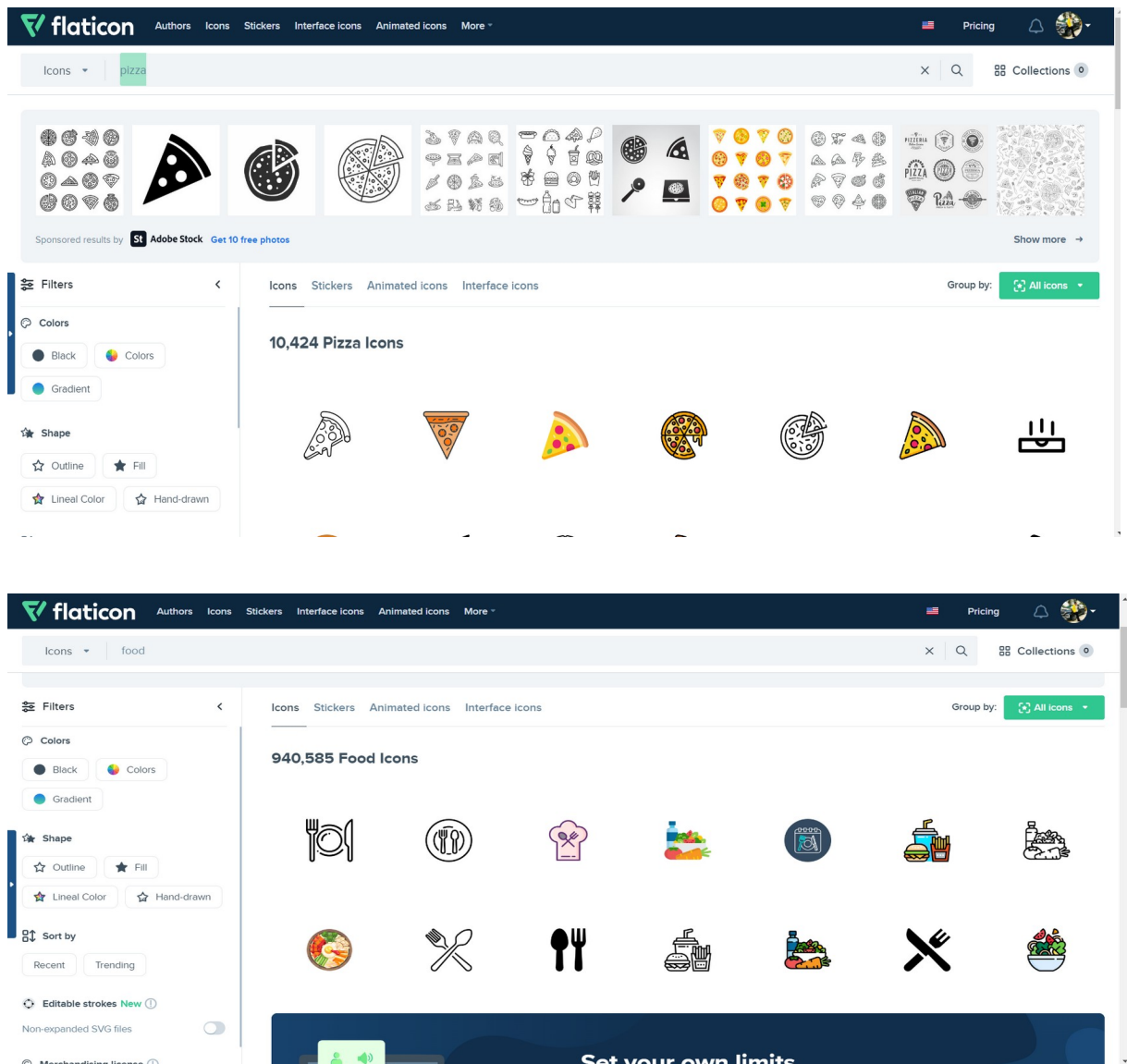
## Step 5: Selecting icons for our UI frames.

[Github](#) [Usage](#)

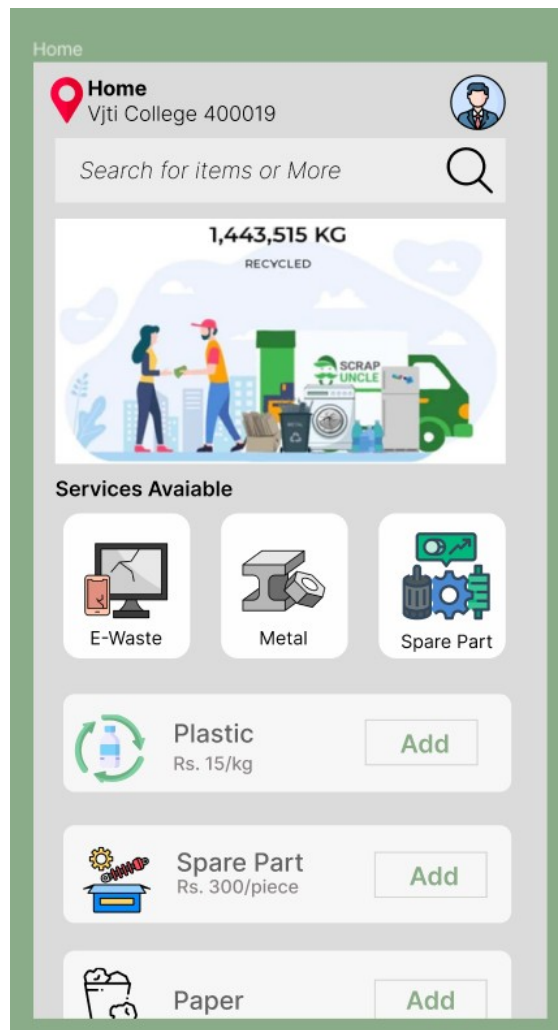
# High Quality Web Icons

Simple Open Source icons carefully crafted for designers & developers





**Step 6: Home page of Scrap Management App.**



## **Conclusion:**

Created wireframe and scrap management UI design using figma. This leads to the development of finely tuned, user-centric interfaces that effectively helps to the understand needs of both designers and end-users.

## **References:**

1. Figma :- <https://www.figma.com/files/recents-and>
2. For UI Icons - <https://boxicons.com/>  
- <https://www.flaticon.com/search?word=pizza>

**Cross Platform App Development Lab Experiment No. 2**

**Aim:** Setting up the development environment of React Native and .NET MAUI on Windows and building first simple application.

**Objectives:**

1. The objective of this lab is to install the development environment for React Native on a Windows system and build a simple application using frameworks.

**Theory:****1. React Native:**

React Native is a JavaScript framework for building mobile applications that can run natively on iOS and Android devices.

It allows developers to use React along with native platform capabilities to create high-performance, cross-platform apps.

**2. Node.js and npm:**

Node.js is a runtime environment that allows executing JavaScript code server-side.

npm (Node Package Manager) is the default package manager for Node.js, facilitating the installation and management of JavaScript packages.

**3. Chocolatey:**

Chocolatey is a package manager for Windows that automates the process of software installation, configuration, and updates.

It provides a command-line interface for managing software packages in a manner similar to Linux package managers.

**4. Android Studio:**

Android Studio is an integrated development environment (IDE) designed for Android app development.

It provides a comprehensive set of tools for designing, building, testing, and debugging Android applications, streamlining the development process.

**Requirements:**

1. Computer with internet access.
2. Windows Operating System
3. Visual Studio IDE
4. Node.js and npm for React Native
5. NET 6 SDK for .NET MAUI
6. Android Studio for Android development
7. Xcode for iOS development (on macOS)
8. Git for version control

**Tools:**

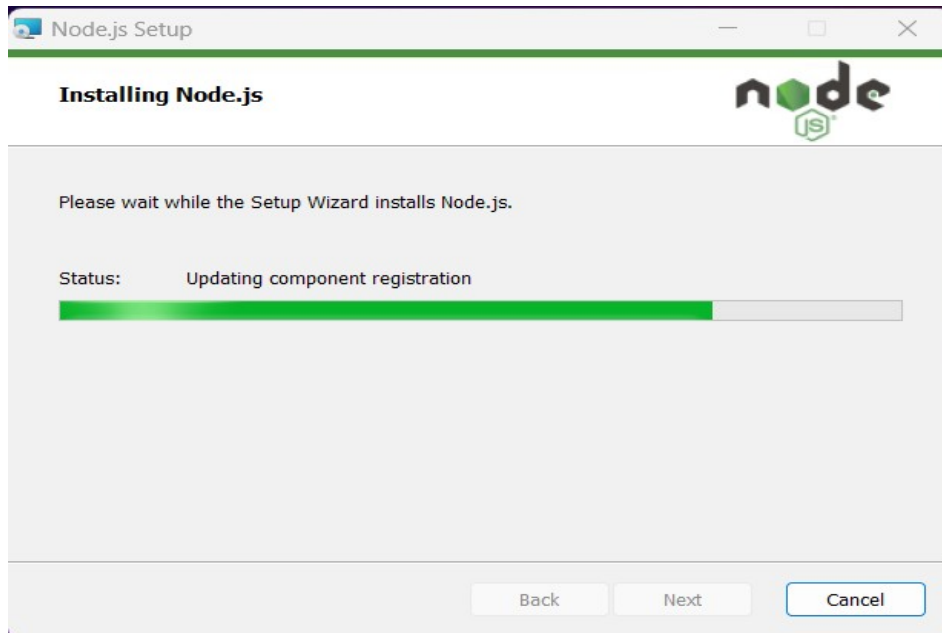
1. Visual Studio IDE

2. Node.js
3. NPM
4. Android Studio

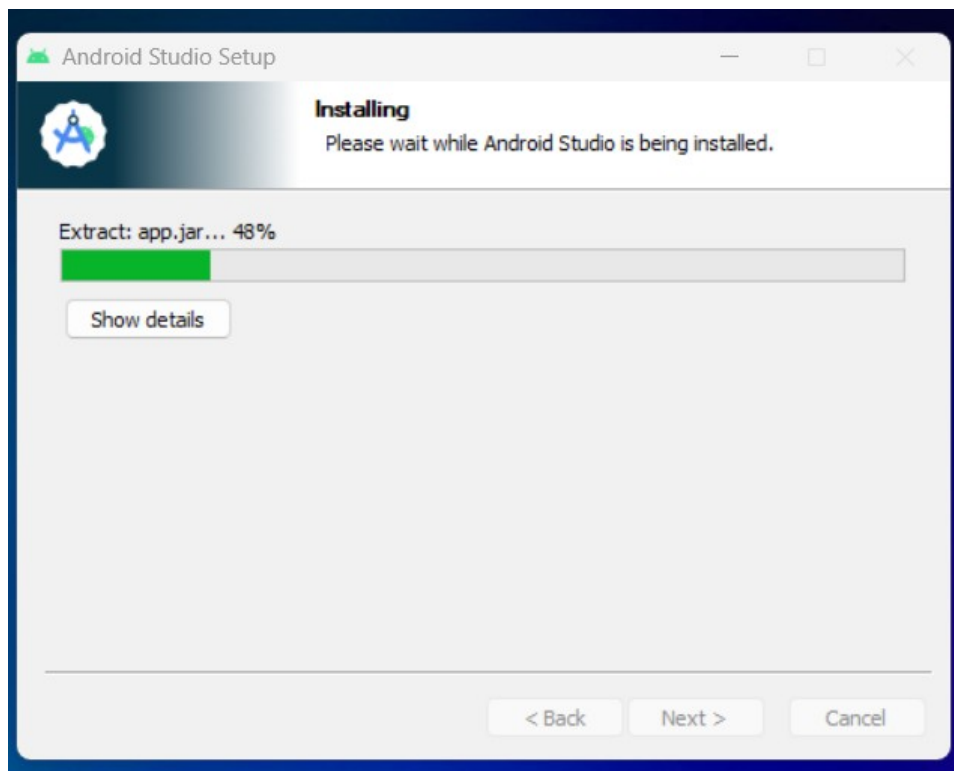
**Steps/Code:**

**1. Install Required Software:**

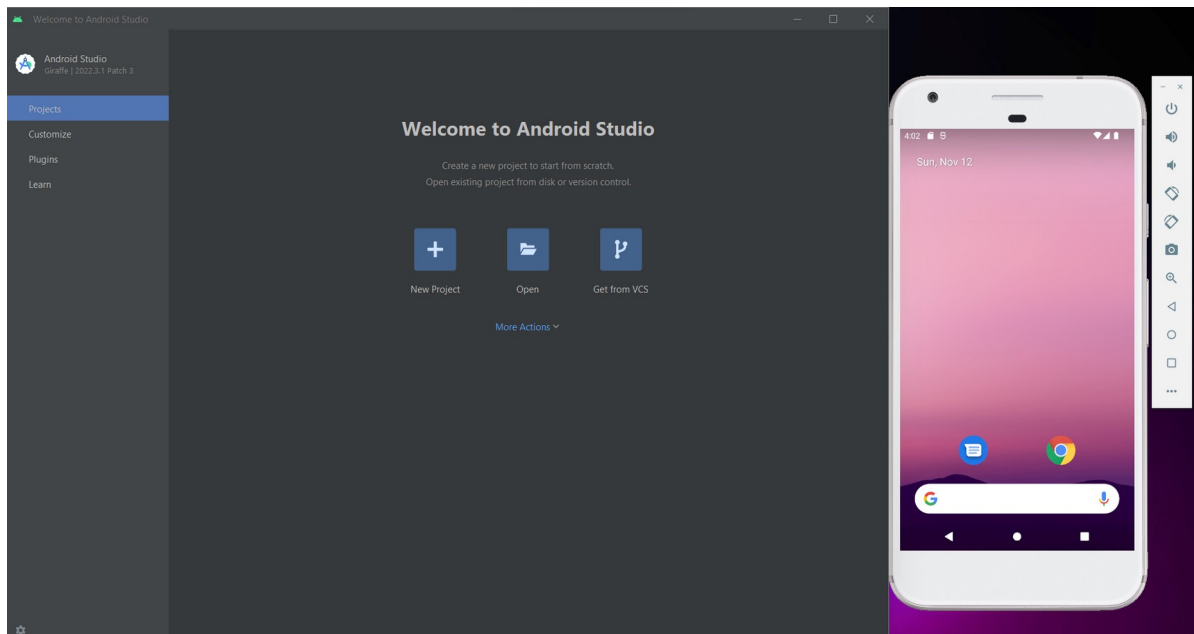
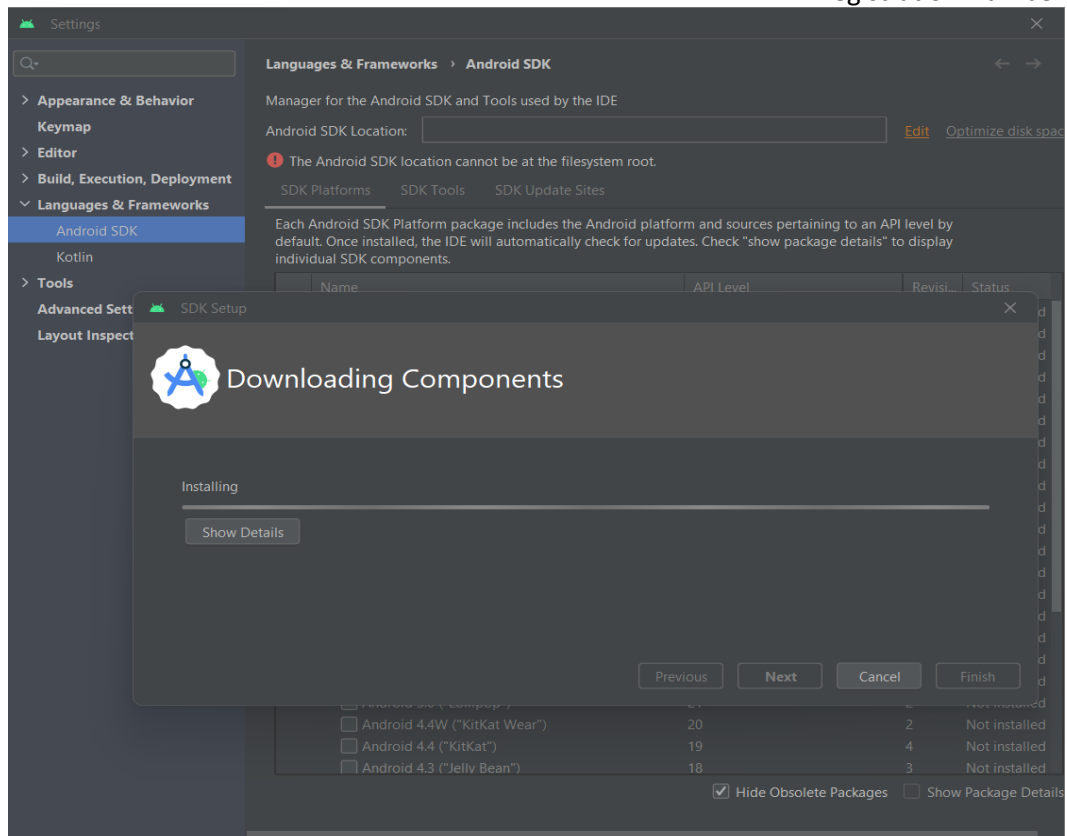
- Install Node.js and npm.



- Set up Android Studio for Android development.



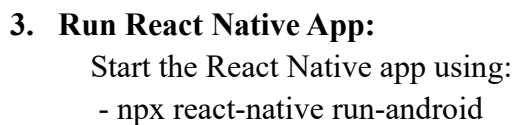




## 2. Create a React Native Project:

Open a terminal and using the following commands:

- npx react-native init newapp
- cd newapp



React Native provides a powerful solution for developers to build versatile and efficient mobile applications using familiar technologies.

### References:

1. React Native: <https://reactnative.dev/docs/environment-setup>
2. Android Studio: <https://developer.android.com/studio>
3. Node : <https://nodejs.org/en/download>
4. Package Manager: <https://chocolatey.org/contact/trial>

## **Cross Platform App Development No. 3**

**Aim:** Programs regarding concepts of JavaScript, JSX, XAML and C.

**Objectives:**

The objective of this experiment is to have an understanding of JavaScript, JSX, XAML, and C.

**Requirements:**

1. Computer with internet access.
2. Windows Operating System
3. Visual Studio IDE
4. Node.js and yarn for React Native
5. Android Studio for Android development
6. Git for version control

**Tools:**

1. Visual Studio IDE
2. Node.js
3. Android Studio
4. Git

**Theory:**

**1. JavaScript:**

- **Role in React Native:** JavaScript is the primary programming language used in React Native. React Native enables developers to build mobile applications using a combination of JavaScript and React, a popular JavaScript library for building user interfaces.

- **Key Features:** JavaScript is an event based and dynamic scripting language. It allows developers to create interactive and responsive user interfaces. In React Native, JavaScript is used to define the logic, components, and behavior of the mobile application.

**2. JSX (JavaScript XML):**

- **Integration in React Native:** In React Native, JSX is used to define the structure of UI components. It provides a more readable way to describe the UI hierarchy and component relationships.

**3. XAML (eXtensible Application Markup Language) and C:**

- React Native primarily uses JavaScript, XAML and C are used in Xamarin to create cross-platform mobile applications. Xamarin allows developers to write the business logic in C and define the user interface using XAML.

### **Class Components:**

#### **1. Login Page (Class Component):**

- State Management: Class components use the ``state`` property to manage local component state. In the login page, state variables (``username`` and ``password``) are defined to store user input.
- Event Handling: The ``handleLogin`` method is triggered when the login button is pressed. It typically contains the logic for authenticating the user based on the provided credentials.

#### **2. Register Page (Class Component):**

- State Management: Similar to the login page, the register page uses the ``state`` property to manage local state variables (``username``, ``email``, and ``password``).
- Event Handling: The ``handleRegister`` method is responsible for executing the registration logic when the user presses the register button.

### **Functional Components:**

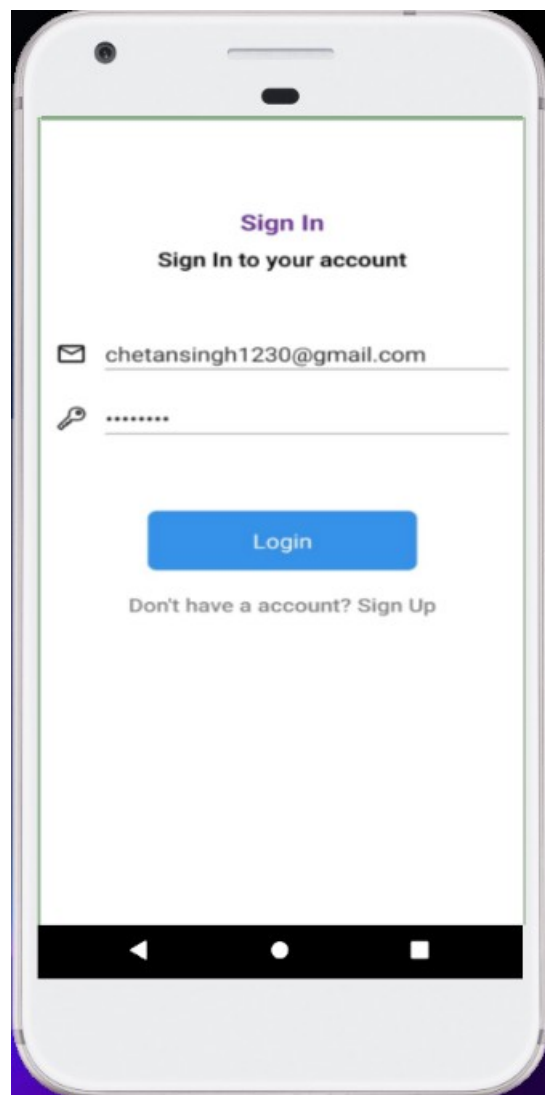
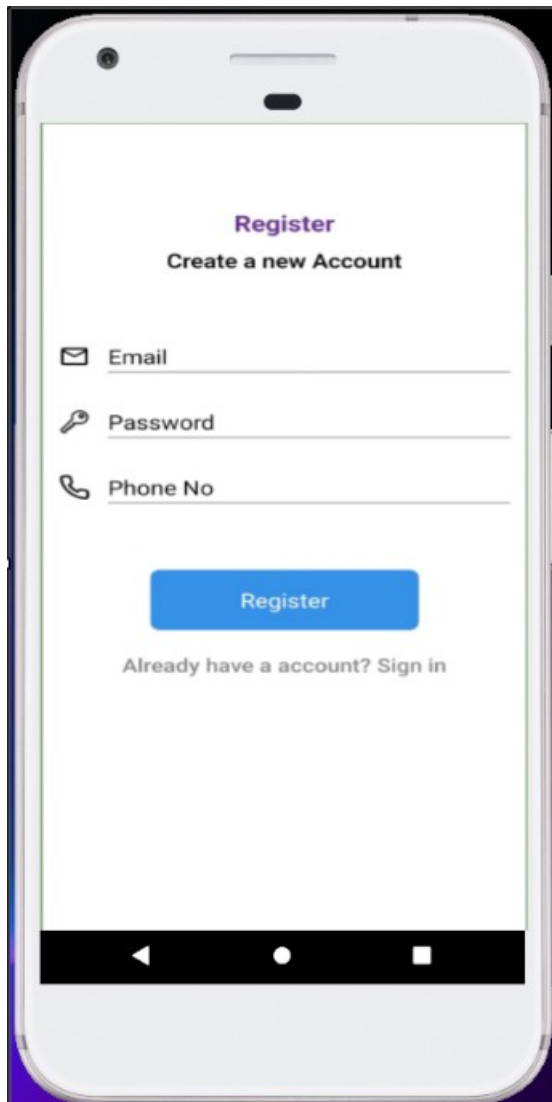
#### **1. Login Page (Functional Component):**

- State with Hooks: Functional components use hooks like ``useState`` to manage local component state. ``useState`` is employed to create state variables (``username`` and ``password``).
- Event Handling: The ``handleLogin`` function is used for handling the login action. It captures the entered username and password and executes the authentication logic.
- Return JSX: Functional components use the ``return`` statement to specify the component's structure. JSX is used to define the UI components, including ``TextInput`` for input fields and ``Button`` for triggering login.

## 2. Register Page (Functional Component):

- State with Hooks: Similar to the login page, the register page utilizes `useState` to manage state variables (`username`, `email`, and `password`).
- Event Handling: The `handleRegister` function is responsible for executing the registration logic when the user presses the register button.
- Return JSX: The component's UI is defined using JSX within the `return` statement. It includes `TextInput` components for user input and a `Button` component for registration.

### Implementation/Code:



### For Login:-

```
import { Icons } from "@expo/vector-icons";  
import React, { useEffect, useState } from "react";
```

```

import { MaterialCommunityIcons } from "@expo/vector-icons";
import { useNavigation } from "@react-navigation/native";
import { signInWithEmailAndPassword } from "firebase/auth";
import { auth } from "../firebase";
const LoginScreen = () => {
  const [email, setEmail] = useState("");
  const [loading, setLoading] = useState(false);
  const [password, setPassword] = useState("");
  const navigation = useNavigation();
  useEffect(() => {
    setLoading(true);
    const unsubscribe = auth.onAuthStateChanged((authUser) => {
      if(!authUser){
        setLoading(false);
      }
      if(authUser){
        navigation.replace("Home");
      }
    });
    return unsubscribe;
  },[])

  const login = () => {
    signInWithEmailAndPassword(auth,email,password).then((userCredential) => {
      console.log("user credential",userCredential);
      const user = userCredential.user;
      console.log("user details",user)
    })
  }

  return (
    <SafeAreaView
      style={{
        flex: 1,
        backgroundColor: "white",
        alignItems: "center",
        padding: 10,
      }} >
      {loading ? (
        <View
          style={{alignItems:"center",justifyContent:"center",flexDirection:"row",flex:1}}>
            <Text style={{marginRight:10}}>Loading</Text>
            <ActivityIndicator size="large" color={"red"}/>
          </View>
        ) : (
          <KeyboardAvoidingView>
            <View
              style={{
                justifyContent: "center",

```

```

        alignItems: "center",
        marginTop: 100,
      }}
    >
    <Text style={{ fontSize: 20, color: "#662d91", fontWeight: "bold" }}>
      Sign In
    </Text>

    <Text style={{ fontSize: 18, marginTop: 8, fontWeight: "600" }}>
      Sign In to your account
    </Text>
  </View>

  <View style={{ marginTop: 50 }}>
    <View style={{ flexDirection: "row", alignItems: "center" }}>
      <MaterialCommunityIcons
        name="email-outline"
        size={24}
        color="black"
        <Pressable onPress={() => navigation.navigate("Register")}
style={{ marginTop: 20 }}>
        <Text
          style={{
            textAlign: "center",
            fontSize: 17,
            color: "gray",
            fontWeight: "500"}} >
          Don't have a account? Sign Up
        </Text>
      </Pressable>
    </View>
  </KeyboardAvoidingView> )}
</SafeAreaView> );};
export default LoginScreen;
const styles = StyleSheet.create({});

```

**For Registration:-**

```

import { Feather } from '@expo/vector-icons';
import { Ionicons } from "@expo/vector-icons";
import React, { useState } from "react";

```

```

import { MaterialCommunityIcons } from "@expo/vector-icons";
import { useNavigation } from "@react-navigation/native";
import { createUserWithEmailAndPassword } from "firebase/auth";
import { doc, setDoc } from "firebase/firestore";
const RegisterScreen = () => {
  const [email,setEmail] = useState("");
  const [password,setPassword] = useState("");
  const [phone,setPhone] = useState("");
  const navigation = useNavigation();
  const register = () => {
    if(email === "" || password === "" || phone === ""){
      Alert.alert(
        "Invalid Details",
        "Please fill all the details",
        {
          text: "Cancel",
          onPress: () => console.log("Cancel Pressed"),
          style: "cancel"
        },
        { text: "OK", onPress: () => console.log("OK Pressed") }
      )
    }
    createUserWithEmailAndPassword(auth,email,password).then((userCredential)
    console.log("user credential",userCredential);
    const user = userCredential._tokenResponse.email;
    const myUserId = auth.currentUser.uid;

    setDoc(doc(db,"users",`${myUserId}`),{
      email:user,
    }
  )
  <View style={{ flexDirection: "row", alignItems: "center" }}>
    <Feather name="phone" size={24} color="black" />
    <TextInput
      value={phone}
      onChangeText={(text) => setPhone(text)}
      placeholder="Phone No"
      placeholderTextColor="black"
      style={{
        fontSize: password ? 18 : 18,
        borderBottomWidth: 1,
        borderBottomColor: "gray",
        marginLeft: 13,
        width: 300,
        marginVertical: 10,
      }}
    />
  </View>
  <Pressable
    onPress={register}
    style={{

```



```

        width: 200,
        backgroundColor: "#318CE7",
        padding: 15,
        borderRadius: 7,
        marginTop: 50,
        marginLeft: "auto",
        marginRight: "auto",
    } >
    <Text style={{ fontSize: 18, textAlign: "center", color: "white" }}>
        Register
    </Text>
</Pressable>
<Pressable onPress={() => navigation.goBack()} style={{ marginTop: 20 }}>
    <Text
        style={{
            textAlign: "center",
            fontSize: 17,
            color: "gray",
            fontWeight: "500", }}>

        Already have a account? Sign in
    </Text>
</Pressable>
</View>
</KeyboardAvoidingView>
</SafeAreaView> );};
export default RegisterScreen
const styles = StyleSheet.create({});

```

**Conclusion:**

From this Experiment we learnt different concepts of JSX, JS where we learnt how one should use functional components if we are writing a presentational component which doesn't have its own state or needs to access a lifecycle hook.

**References:**

- <https://nodejs.org/en>
- <https://developer.android.com/studio>

## **Cross Platform App Development Lab Experiment No.4**

**Aim:** Basic user interface design using react components, Stylesheet, Flexbox and XAML, manipulation of components using States and Props.

**Objectives:**

1. Understand the basics of React components.
2. Learn how to use Stylesheet for styling.
3. Grasp the concepts of Flexbox for layout design.
4. Familiarize yourself with XAML for markup.
5. Explore the manipulation of components using React States and Props.

**Theory:**

**- React Components:**

- Building blocks of a React application.
- Encapsulate reusable code.
- Can be class components or functional components.

**- Stylesheet:**

- Used for styling React components.
- Helps in maintaining a consistent look and feel.
- Can include CSS or other styling languages.

**- Flexbox:**

- A layout model for designing complex layouts.
  - Provides an efficient way to distribute space among items in a container.
- Simplifies the design of responsive and dynamic layouts.

**- XAML:**

- Extensible Application Markup Language.
- Used for designing user interfaces in .NET applications.
- A declarative XML-based language.

**- States and Props:**

- States:
  - Manage the internal state of a component.
  - Allow components to change their output over time.
- Props:

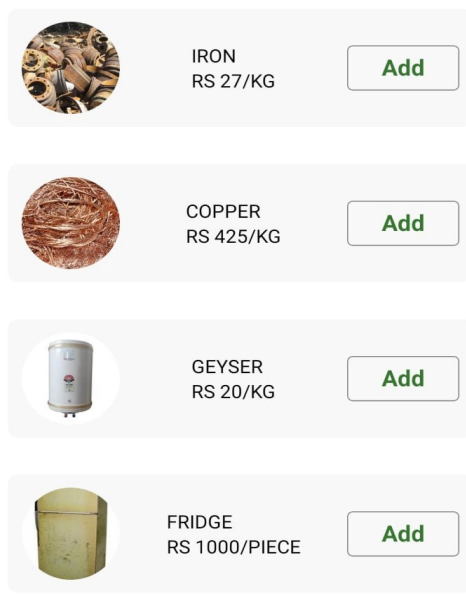
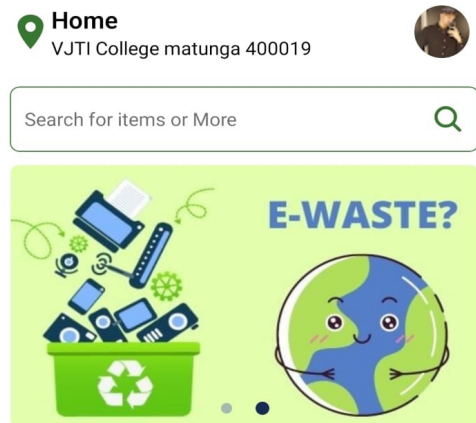
- Short for properties.
- Enable the passing of data from parent to child components.

**Requirements:**

- A text editor (e.g., Visual Studio Code).
- Node.js and npm installed.
- Basic understanding of HTML and JavaScript.
- Familiarity with the React library.

**Tools:**

- Text editor (e.g., Visual Studio Code).
- Node.js and npm.

**Implementation/Code:-****Code:**

## - Using Flexbox for designing

```

return (
  // Location and profile
  <ScrollView style={{backgroundColor: "F0F0F0", flex:1,marginTop:1}}>
    <StatusBar hidden />

    <View style={{ flexDirection: "row", alignItems: "center", padding:
10 }}>
      <Icons name="location-sharp" size={30} color="#32712c" />
      <View>
        <Text style={{ fontSize: 18, fontWeight: "600" }}>Home</Text>
        <Text>{displayCurrentAddress}</Text>
      </View>

      <Pressable style={{ marginLeft: "auto", marginRight: 7 }}>
        <Image
          style={{ width: 40, height: 40, borderRadius: 20 }}
          source={{
            uri:
"https://lh3.googleusercontent.com/ogw/AKPQZvzhvGMWRESqI4jU33yjRU876j-
tzbGwV0948GYgVw=s32-c-mo",
          }}
        />
      </Pressable>
    </View>
    { /* Search bar */ }
  </ScrollView>
)

```

## - Using in style to use CCS

```

<View
  style={{
    padding: 10,
    margin: 10,
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-between",
    borderWidth: 0.8,
    borderColor: "#32712c",
    borderRadius: 7,
  }}
>
  <TextInput placeholder="Search for items or More" />

```

## - Using inline style to style list of scrap items.

```
const ScrapItem = ({ item }) => {  
  return (  
    <View>  
      <Pressable  
        style={{  
          backgroundColor: "#F8F8F8",  
          borderRadius: 8,  
          padding: 10,  
          flexDirection: "row",  
          alignItems: "center",  
          justifyContent: "space-between",  
          margin: 14,  
        }}  
      >  
        <View>  
          <Image  
            style={{ width: 70, height: 70, borderRadius: 50 }}  
            source={{ uri: item.image }}  
          />  
        </View>  
        <View>  
          <Text>{item.name}</Text>  
          <Text>{item.price}</Text>  
        </View>  
      </Pressable>  
    </View>  
  )  
}
```

### **Conclusion:**

In this project, we learnt the fundamentals of React components, Stylesheet for styling, Flexbox for layout, XAML for markup, and the usage of States and Props for component manipulation. This provides a solid foundation for building interactive and well-styled user interfaces using React.

### **References:**

1. **React Documentation:** [<https://reactjs.org/docs/getting-started.html>]  
(<https://reactjs.org/docs/getting-started.html>)
2. **Flexbox Guide:** [<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>]  
(<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>)

3. **XAML Overview:** [<https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml>](<https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml>)

### **Cross Platform App Development Lab Experiment No. 5**

**Aim:** Adding user interactivity in the base components and customization of layout using functions provided by React Native and XAML.

#### **Objectives:**

1. Implement user interactivity in React Native components.
2. Customize layout using functions provided by React Native and XAML.

#### **Theory:**

- React Native Components:
  - Building blocks for mobile app development.
  - Includes interactive elements like buttons, input fields, etc.
  - Can be extended with user-defined logic.
- Layout Customization with Functions:
  - React Native provides functions for dynamic layout customization.
  - XAML offers layout customization through specific functions and attributes.
  - These functions enable responsiveness and adaptability.

#### **Requirements:**

- React Native development environment set up.
- A text editor (e.g., Visual Studio Code).
- Understanding of JavaScript and React Native concepts.

#### **Tools:**

- React Native development environment.
- Text editor (e.g., Visual Studio Code).

#### **Implementation/Code:-**

- User can register

```

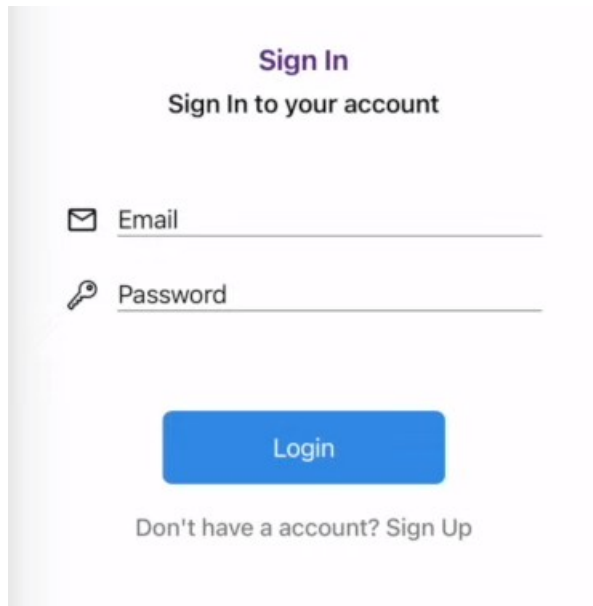
<Pressable
  onPress={register}
  style={{
    width: 200,
    backgroundColor: "#318CE7",
    padding: 15,
    borderRadius: 7,
    marginTop: 50,
    marginLeft: "auto",
    marginRight: "auto",
  }}
>
  <Text style={{ fontSize: 18, textAlign: "center", color: "white" }}>
    Register
  </Text>
</Pressable>

<Pressable onPress={() => navigation.goBack()} style={{ marginTop: 20 }}>
  <Text
    style={{
      textAlign: "center",
      fontSize: 17,
      color: "gray",
      fontWeight: "500",
    }}
  >
    Already have a account? Sign in
  </Text>
</Pressable>

```

- User can sign in if they are already registered





```

<Pressable
  onPress={login}
  style={{
    width: 200,
    backgroundColor: "#318CE7",
    padding: 15,
    borderRadius: 7,
    marginTop: 50,
    marginLeft: "auto",
    marginRight: "auto",
  }}
>
  <Text style={{ fontSize: 18, textAlign: "center", color: "white" }}>
    Login
  </Text>
</Pressable>

```

```

    <Pressable onPress={() => navigation.navigate("Register")} style={{ marginTop:
20 }}>
      <Text
        style={{
          textAlign: "center",
          fontSize: 17,
          color: "gray",
          fontWeight: "500",
        }}
      >
        Don't have a account? Sign Up
      </Text>
    </Pressable>

```

- User can add scrap type they want to add and sell to the seller



```
<Pressable style={{ width: 80}}>
  <Text
    style={{
      borderColor: "gray",
      borderWidth: 0.8,
      borderRadius:4,
      marginVertical: 10,
      color: "#32712c",
      textAlign: "center",
      padding:5,
      fontSize:17,
      fontWeight:"bold"
    }}
  >
    Add
  </Text>
</Pressable>
</Pressable>
</View>
```

### **Conclusion:**

By adding user interactivity to base components and the layout customization functions provided by React Native and XAML, learned enhance the user experience and create more dynamic and adaptable interfaces for mobile applications.

### **References:**

1. React Native Documentation:

[<https://reactnative.dev/docs/getting-started>](<https://reactnative.dev/docs/getting-started>)

2. XAML Layout and Customization:

[<https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml>](<https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml>)

## **Cross Platform App Development Lab Experiment No. 6**

**Aim:** Integrating third party components into our application and XAML pages using shared resources.

### **Objectives:**

1. Understand the process of integrating third-party components.
2. Explore the use of shared resources in XAML pages.

### **Theory:**

- Integrating Third-Party Components:
  - Involves adding pre-built components from external sources.
  - Enhances functionality without building everything from scratch.
  - Requires proper integration and configuration.
- Shared Resources in XAML:
  - Resources shared across multiple XAML pages.
  - Can include styles, templates, and other reusable elements.
  - Promotes consistency in design and functionality.

### **Requirements:**

- Access to the third-party component library.
- Existing XAML-based application.

### **Tools:**

- Visual Studio or any XAML-compatible IDE.
- Necessary packages or tools for integrating third-party components.

### **Implementation/ Code:-**

```
const path = require('path');
module.exports = ({ env }) => {
  const client = env('DATABASE_CLIENT', 'sqlite');
  const connections = {
    mysql: {
      connection: {
        connectionString: env('DATABASE_URL'),
        host: env('DATABASE_HOST', 'localhost'),
        port: env.int('DATABASE_PORT', 3306),
        database: env('DATABASE_NAME', 'strapi'),
        user: env('DATABASE_USERNAME', 'strapi'),
        password: env('DATABASE_PASSWORD', 'strapi'),
        ssl: env.bool('DATABASE_SSL', false) && {
```

```

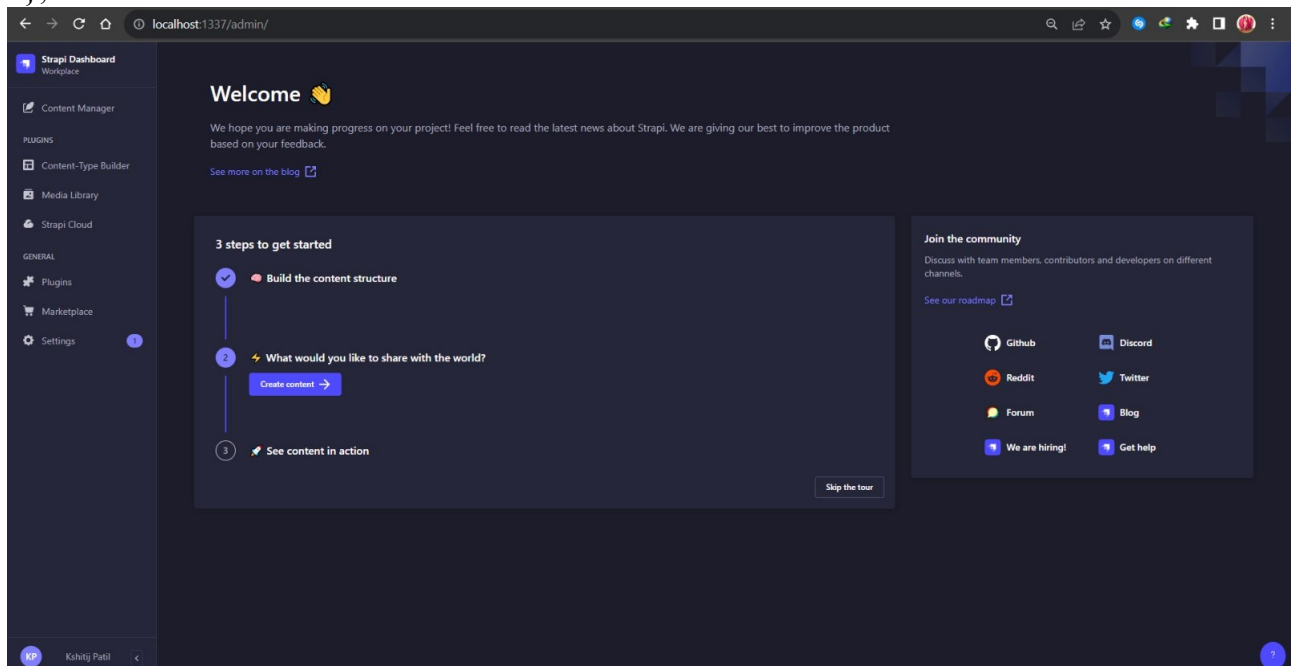
    key: env('DATABASE_SSL_KEY', undefined),
    cert: env('DATABASE_SSL_CERT', undefined),
    ca: env('DATABASE_SSL_CA', undefined),
    capath: env('DATABASE_SSL_CAPATH', undefined),
    cipher: env('DATABASE_SSL_CIPHER', undefined),
    rejectUnauthorized: env.bool(
      'DATABASE_SSL_REJECT_UNAUTHORIZED',
      true
    ),
  },
},
pool: { min: env.int('DATABASE_POOL_MIN', 2), max:
env.int('DATABASE_POOL_MAX', 10) },
},
mysql2: {
  connection: {
    host: env('DATABASE_HOST', 'localhost'),
    port: env.int('DATABASE_PORT', 3306),
    database: env('DATABASE_NAME', 'strapi'),
    user: env('DATABASE_USERNAME', 'strapi'),
    password: env('DATABASE_PASSWORD', 'strapi'),
    ssl: env.bool('DATABASE_SSL', false) && {
      key: env('DATABASE_SSL_KEY', undefined),
      cert: env('DATABASE_SSL_CERT', undefined),
      ca: env('DATABASE_SSL_CA', undefined),
      capath: env('DATABASE_SSL_CAPATH', undefined),
      cipher: env('DATABASE_SSL_CIPHER', undefined),
      rejectUnauthorized: env.bool(
        'DATABASE_SSL_REJECT_UNAUTHORIZED',
        true
      ),
    },
  },
},
pool: { min: env.int('DATABASE_POOL_MIN', 2), max:
env.int('DATABASE_POOL_MAX', 10) },
},
postgres: {
  connection: {
    connectionString: env('DATABASE_URL'),
    host: env('DATABASE_HOST', 'localhost'),
    port: env.int('DATABASE_PORT', 5432),
    database: env('DATABASE_NAME', 'strapi'),
    user: env('DATABASE_USERNAME', 'strapi'),
    password: env('DATABASE_PASSWORD', 'strapi'),
    ssl: env.bool('DATABASE_SSL', false) && {
      key: env('DATABASE_SSL_KEY', undefined),
      cert: env('DATABASE_SSL_CERT', undefined),

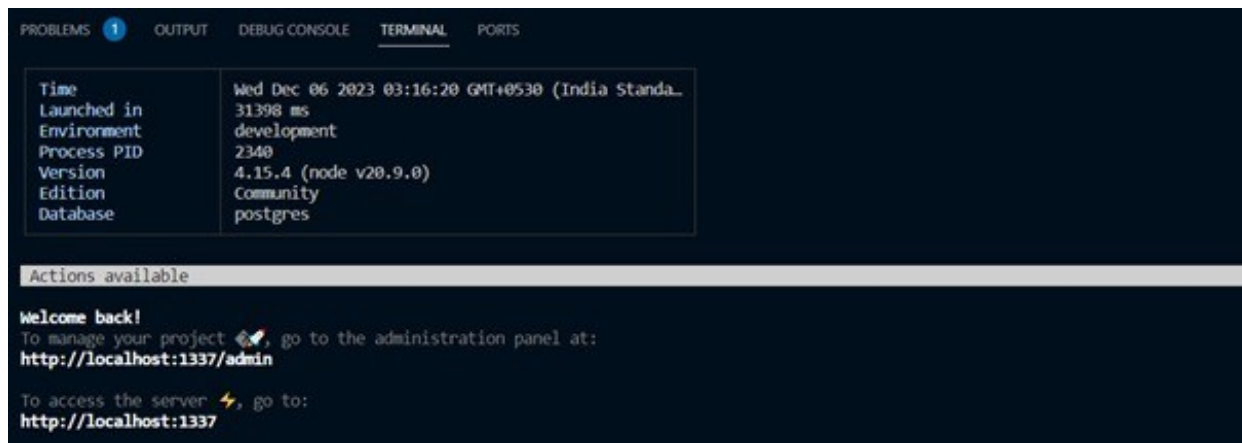
```

```

ca: env('DATABASE_SSL_CA', undefined),
capath: env('DATABASE_SSL_CAPATH', undefined),
cipher: env('DATABASE_SSL_CIPHER', undefined),
rejectUnauthorized: env.bool(
  'DATABASE_SSL_REJECT_UNAUTHORIZED',
  True    ), },
schema: env('DATABASE_SCHEMA', 'public'),},
pool: { min: env.int('DATABASE_POOL_MIN', 2), max:
env.int('DATABASE_POOL_MAX', 10) },
},
sqlite: {
  connection: {
    filename: path.join(
      __dirname,
      '..',
      env('DATABASE_FILENAME', '.tmp/data.db') ),},
  useNullAsDefault: true,}};
return {
  connection: {
    client,
    ...connections[client],
    acquireConnectionTimeout: env.int('DATABASE_CONNECTION_TIMEOUT', 60000),
  },
};

```





### Conclusion:

We successfully integrated third-party components and shared resources in XAML pages, we design of our application. This approach helps us save development time and promotes consistency in the user interface.

### References:

1. **Microsoft XAML documentation:** [<https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml>](<https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml>)

## **Cross Platform App Development Lab Experiment No.7**

**Aim:** Implementing multiple screen navigation and nested navigation using solutions provided by React Navigation and .NET MAUI.

### **Objectives:**

1. Implement multiple screen navigation in a React Native application.
2. Explore the solutions provided by React Navigation for nested navigation.
3. Implement multiple screen navigation in a .NET MAUI application.
4. Utilize .NET MAUI's solutions for nested navigation.

### **Theory:**

#### **- React Navigation:**

- Library for navigation in React Native applications.
- Supports stack, tab, drawer, and other types of navigators.
- Allows easy management of multiple screens.
- Nested Navigation in React Navigation:
  - Involves navigating within screens of a navigator.
  - Achieved using nested navigators like Stack Navigator within Tab Navigator.

#### **- .NET MAUI Navigation:**

- .NET Multi-platform App UI (MAUI) framework for cross-platform app development.
- Supports navigation patterns similar to Xamarin Forms.
- Offers navigation containers and pages for screen navigation.
- Nested Navigation in .NET MAUI:
  - Involves navigating between pages and utilizing navigation containers.
  - Hierarchical navigation structure for managing nested navigation.

### **Requirements:**

- React Native development environment for React Navigation.
- .NET MAUI development environment for .NET MAUI navigation.

**Tools:**

- React Navigation library for React Native.
- Visual Studio or Visual Studio Code for .NET MAUI development.

**Implementation/ Code:-**

```
import { StyleSheet, Text, View } from 'react-native'
import React from 'react'
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { NavigationContainer } from '@react-navigation/native';
import HomeScreen from './screens/HomeScreen';
import PickupScreen from './screens/PickUpScreen';
import CartScreen from './screens/CartScreen';
import LoginScreen from './screens/LoginScreen';
import RegisterScreen from './screens/RegisterScreen';
import ProfileScreen from './screens/ProfileScreen';

const StackNavigator = () => {
  const Stack = createNativeStackNavigator();
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Login" component={LoginScreen}
options={{headerShown:false}}/>
        <Stack.Screen name="Home" component={HomeScreen}
options={{headerShown:false}}/>
        <Stack.Screen name="PickUp" component={PickUpScreen}
options={{headerShown:false}}/>
        <Stack.Screen name="Cart" component={CartScreen}
options={{headerShown:false}}/>

        <Stack.Screen name="Register" component={RegisterScreen}
options={{headerShown:false}}/>
        <Stack.Screen name="Profile" component={ProfileScreen}
options={{headerShown:false}}/>






      </Stack.Navigator>
    </NavigationContainer>
  )
}

export default StackNavigator

const styles = StyleSheet.create({})
```



- Used stack navigator for navigating from one screen to another.

<b>Sign In</b>	<b>Register</b>
<b>Sign In to your account</b>	<b>Create a new Account</b>
 <input type="text" value="Email"/>	 <input type="text" value="Email"/>
 <input type="password" value="Password"/>	 <input type="password" value="Password"/>
	 <input type="text" value="Phone No"/>
<input type="button" value="Login"/>	<input type="button" value="Register"/>
<a href="#">Don't have a account? Sign Up</a>	<a href="#">Already have a account? Sign in</a>

- First we, clicked on **Don't have a account? Sign Up** then we are navigated to another page that is **register page**

### **Conclusion:**

By implementing multiple screen navigation using React Navigation for React Native built-in navigation solutions, we enhance the user experience and provide a structured flow within the applications.

### **References:**

1. React Navigation documentation: <https://reactnavigation.org/docs/getting-started>
2. .NET MAUI documentation: <https://docs.microsoft.com/en-us/dotnet/maui>

## **Cross Platform App Development Lab Experiment No. 8**

**Aim:** Implementation of asynchronous functions and outside API calls in JavaScript, React Native, and .NET MAUI.

### **Objectives:**

1. Implement asynchronous functions in JavaScript.
2. Execute API calls in a React Native application asynchronously.
3. Apply asynchronous functions in a .NET MAUI project.
4. Perform outside API calls in .NET MAUI asynchronously.

### **Theory:**

#### **- Asynchronous Functions:**

- Functions that operate independently of the main program flow.
- Allow concurrent execution of multiple tasks.
- Improve performance and responsiveness by preventing blocking.

#### **- API Calls in React Native:**

- Utilize libraries like Axios or fetch for HTTP requests.
- Implement asynchronous functions to handle API responses.

#### **- Asynchronous Programming in .NET MAUI:**

- Utilize asynchronous programming with `async` and `await` keywords.
- Use the HttpClient class for making asynchronous API calls.

### **Requirements:**

- A basic understanding of JavaScript for React Native.
- Familiarity with C# for .NET MAUI.

### **Tools:**

- For JavaScript and React Native:
  - Any text editor (e.g., Visual Studio Code).
  - React Native development environment.
- For .NET MAUI:
  - Visual Studio or Visual Studio Code.
  - .NET MAUI development environment.

**Implementation/Code:-**

- Creating an API call for location service for our application

```
const [displayCurrentAddress, setdisplayCurrentAddress] = useState(
  "VJTI College matunga 400019"
);
const [locationServicesEnabled, setLocationServicesEnabled] = useState(false);
useEffect(() => {
  checkIfLocationEnabled();
  getCurrentLocation();
}, []);

const checkIfLocationEnabled = async () => {
  let enabled = await Location.hasServicesEnabledAsync();
  if (!enabled) {
    Alert.alert(
      "Location services not enabled",
      "Please enable the location services",
      [
        {
          text: "Cancel",
          onPress: () => console.log("Cancel Pressed"),
          style: "cancel",
        },
        { text: "OK", onPress: () => console.log("OK Pressed") },
      ],
      { cancelable: false }
    );
  } else {
    setLocationServicesEnabled(enabled);
  }
};
```

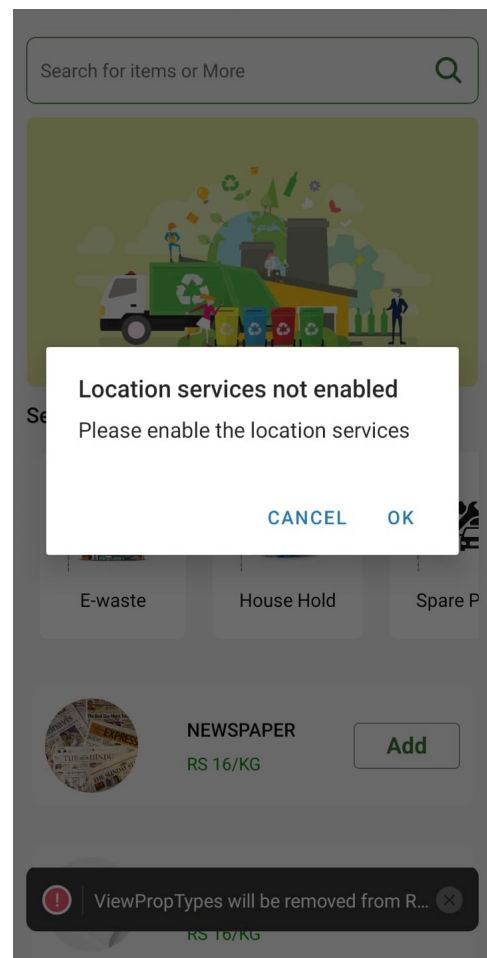
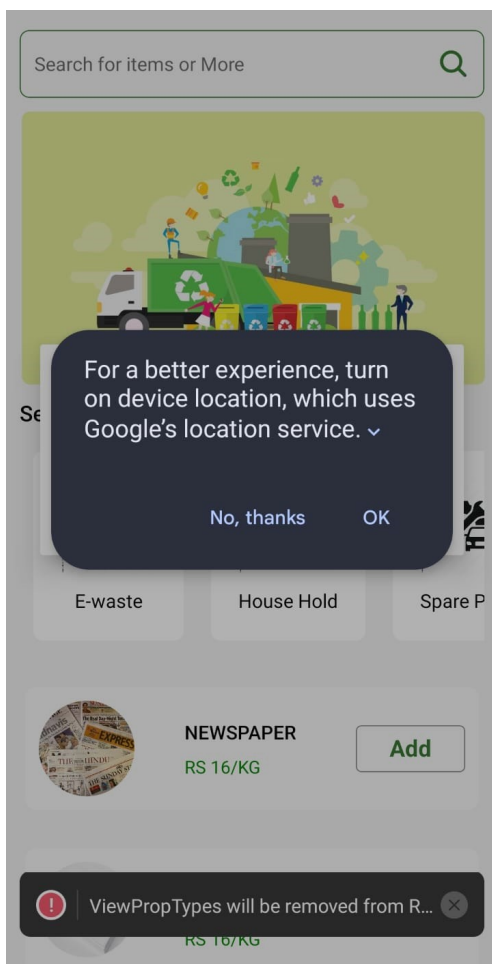
- Code for the part fetching location

```
const getCurrentLocation = async () => {
  let { status } = await Location.requestForegroundPermissionsAsync();

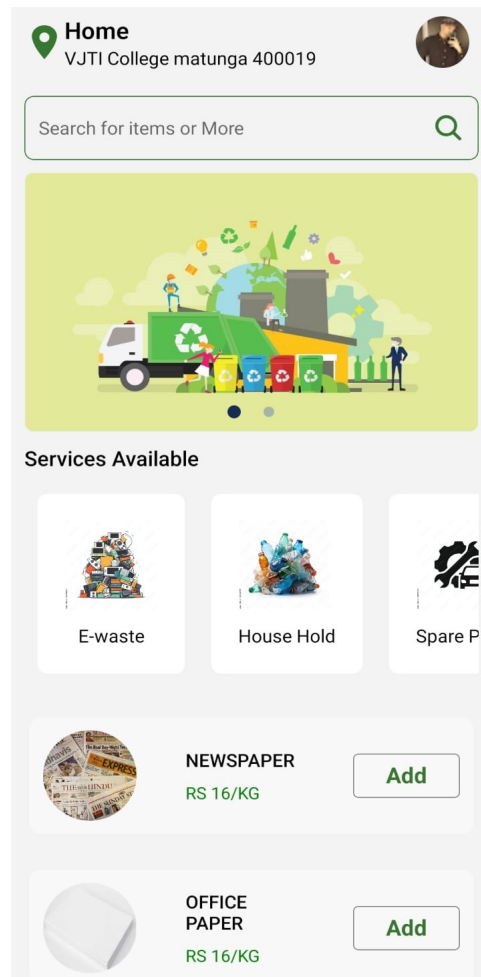
  if (status !== "granted") {
    Alert.alert(
      "Permission denied",
      "allow the app to use the location services",
      [
        {
          text: "Cancel",
          onPress: () => console.log("Cancel Pressed"),
          style: "cancel",
        },
        { text: "OK", onPress: () => console.log("OK Pressed") },
      ]
    );
  }
};
```

```
    ],  
    { cancelable: false }  
  );  
}  
  
const { coords } = await Location.getCurrentPositionAsync();  
// console.log(coords)  
if (coords) {  
  const { latitude, longitude } = coords;  
  
  let response = await Location.reverseGeocodeAsync({  
    latitude,  
    longitude,  
  });  
}
```

- So we can press OK and give permission to access location of our device.
- For the same we can cancel the pop-up we can see in below screenshots



- Now, we can see location on the top of application.



### **Conclusion:**

We learnt to implement of asynchronous functions and outside API calls in React Native and handling outside API calls in JavaScript, React Native, we make sure efficient execution of tasks, improving the overall responsiveness of our applications.

### **References:**

1. MDN Web Docs - Asynchronous programming in JavaScript:  
<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous>
2. React Native Networking: <https://reactnative.dev/docs/network>

## **Cross Platform App Development Lab Experiment No.9**

**Aim:** Exploration of life cycle functions for components, hook functions, and XAML page.

### **Objectives:**

1. Understand the life cycle functions of components in software development.
2. Explore hook functions and their role in component-based architectures.
3. Gain insights into the life cycle of XAML pages in the context of UI development.

### **Theory:**

- **Life Cycle Functions for Components:** Components go through various stages during their life cycle, such as initialization, mounting, updating, and unmounting. Understanding these stages is crucial for efficient development and optimization.

- **Hook Functions:** Hook functions are special functions that developers can use to perform actions at specific points in a component's life cycle. Common hooks include `'componentDidMount'`, `'componentDidUpdate'`, and `'componentWillUnmount'` in React.

- **XAML Pages Life Cycle:** XAML (eXtensible Application Markup Language) pages, commonly used in UI development, have their own life cycle. This involves loading, initialization, rendering, and disposal of the page.

- There are four main components and hooks in React Native:

**useEffect:** This hook allows you to perform side effects in functional components, such as fetching data or updating the DOM.

**useContext:** This hook allows you to access context values in functional components.

**useReducer:** This hook allows you to manage state in a more complex way than using `useState`.

**useState:** This hook allows you to add state to functional components.

### **Requirements:**

1. Development environment for the chosen framework (e.g., React for components, Xamarin for XAML).
2. Code editor (e.g., Visual Studio Code, Visual Studio).
3. Basic understanding of the chosen framework and language.

**Tools:**

1. React for component-based development.
2. Xamarin for XAML pages.
3. Code editor of choice.

**Implementation/Code:**


- Using useEffect hook to implement authentication on our application


**useEffect:** This hook allows you to perform side effects in functional components, such as fetching data or updating the DOM.


```
useEffect(() => {  
  setLoading(true);  
  const unsubscribe = auth.onAuthStateChanged((authUser) => {  
    if(!authUser){  
      setLoading(false);  
      if(authUser){  
        navigation.replace("Home");  
      }  
    });  
    return unsubscribe;  
  }, [])
```

**Register**

Create a new Account

 Email

 Password

 Phone No

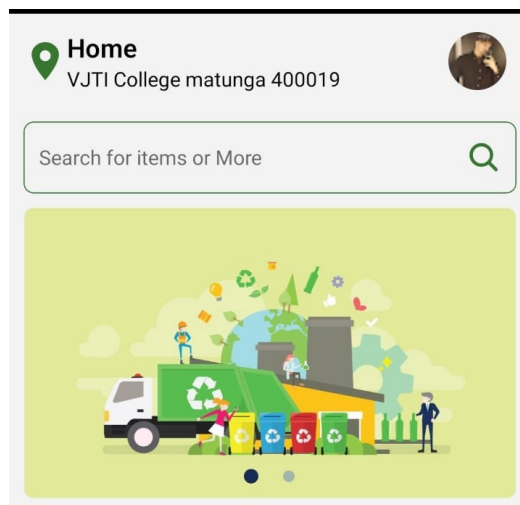
Register

Already have a account? Sign in

- Using useState hook to display **Current Location**.

- **useState**: This hook allows you to add state to functional components.

```
const [displayCurrentAddress, setdisplayCurrentAddress] = useState(  
  "VJTI College matunga 400019"  
);  
const [locationServicesEnabled, setLocationServicesEnabled] =  
useState(false);  
useEffect(() => {  
  checkIfLocationEnabled();  
  getCurrentLocation();  
}, []);
```



### **Conclusion:**

We learned about components, hook functions, and XAML pages is essential for building efficient applications. Proper utilization of life cycle events and hooks can lead to optimized resource management and improved user experiences.

### **References:**

#### **1. React Documentation:**

<https://reactjs.org/docs/react-component.html>](<https://reactjs.org/docs/react-component.html>)

#### **2. Xamarin.Forms Page Life Cycle:**

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/page-lifecycle>](<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/page-lifecycle>)



## **Cross Platform App Development Lab Experiment No.10**

**Aim:** Implementation and integration with Firebase backend and application deployment.

### **Objectives:**

1. Implement Firebase backend services in an application.
2. Integrate the application with Firebase for data storage and authentication.
3. Deploy the application to a hosting service.

### **Theory:**

#### **- Firebase Backend:**

- A cloud-based platform provided by Google.
- Offers services like real-time database, authentication, cloud functions, and more.
- Suitable for mobile and web applications.

#### **- Integration with Firebase:**

- Connect the application to Firebase using SDKs.
- Utilize Firebase services such as Firestore for database and Firebase Authentication for user management.

#### **- Application Deployment:**

- The process of making an application available for use.
- Involves hosting the application on servers accessible over the internet.
- Firebase Hosting is a common solution for web applications.

### **Requirements:**

- Firebase account for backend services.
- Application codebase with Firebase SDK integrated.
- Deployment account or service for hosting.

### **Tools:**

- Firebase Console for backend configuration.
- Firebase SDK for application integration.

### **Implementation/Code:**

- Configuration file of firebase database

```
// Import the functions you need from the SDKs you need

import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDF5ZoBL9U9u5zAa4b5J47HE6pxhFV6cIY",
  authDomain: "scrapmanagement-2f0ae.firebaseio.com",
  projectId: "scrapmanagement-2f0ae",
  storageBucket: "scrapmanagement-2f0ae.appspot.com",
  messagingSenderId: "61024284114",
  appId: "1:61024284114:web:9db9ab612a04c79b6dac0b"
};

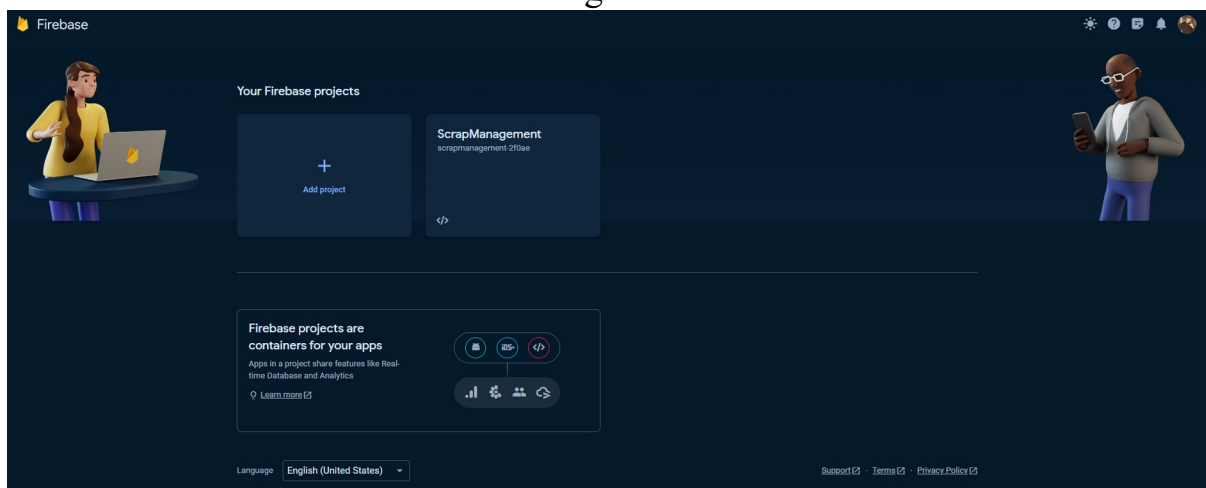
// Initialize Firebase
const app = initializeApp(firebaseConfig);

const auth = getAuth(app);

const db = getFirestore();


export { auth, db };
```


- Firebase console Interface for creating database.



- All user's data is stored in stored in firebase database. (Like Login)

**Sign In**  
Sign In to your account

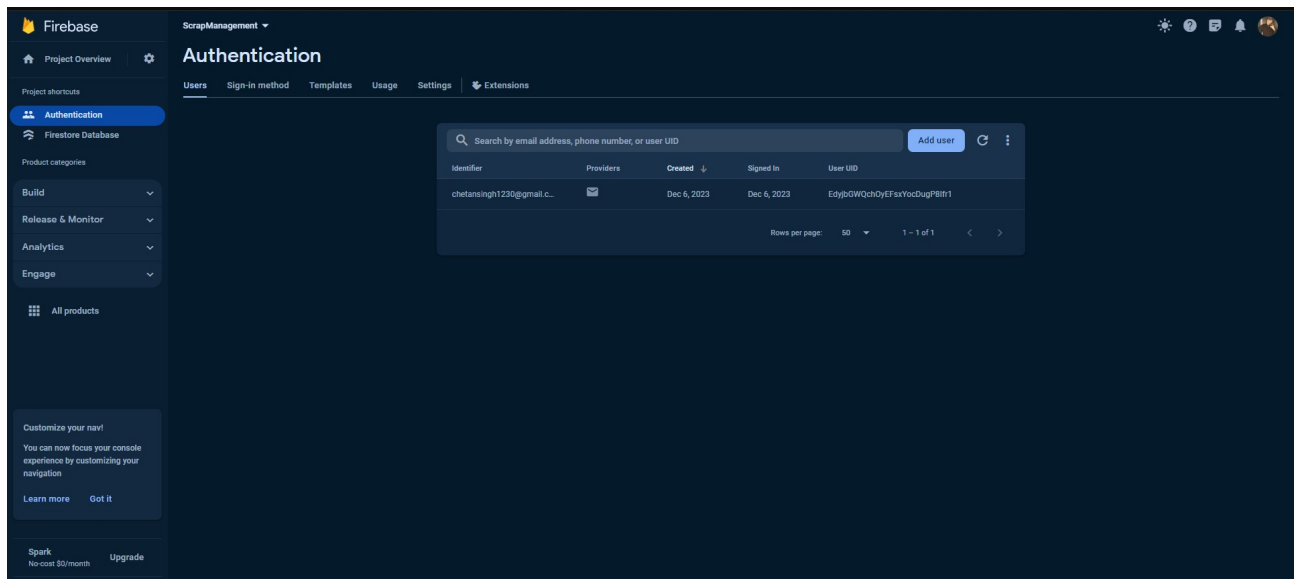





**Login**

Don't have a account? [Sign Up](#)

- User data stored in firebase



The screenshot shows the Firebase Authentication console. The left sidebar contains navigation links for Project Overview, Authentication, Firestore Database, and Product categories. The main area displays the 'Users' tab with a search bar and a table of users. The table has columns for Identifier, Providers, Created, Signed In, and User UID. One user is listed with the email 'chetansingh123@gmail.c...' and a User UID of 'EdyBQWQchOyEFsxYouDagP8tr1'. The bottom of the console shows a 'Spark' plan with a 'No-cost \$0/month' label and an 'Upgrade' button.

Identifier	Providers	Created	Signed In	User UID
chetansingh123@gmail.c...		Dec 6, 2023	Dec 6, 2023	EdyBQWQchOyEFsxYouDagP8tr1

### **Conclusion:**

We learnt to successfully implement and integrating Firebase backend services, we create a scalable and accessible solution with features like real-time data storage and user authentication.

### **References:**

1. **Firebase Documentation:**

<https://firebase.google.com/docs>

2. **Firebase Hosting Documentation:**

<https://firebase.google.com/docs/hosting>

<https://firebase.google.com/docs/hosting>