CSC303 MAD Lab Manual sp22 v3

Hybrid Power Systems (COMSATS University Islamabad)

Lab Manual

CSC303-Mobile Application Development

CUI

Department of Computer Science
Islamabad Campus

Lab Contents:

JavaScript Basics and Functions, Js ES6 Arrow Functions and Classes, Js Arrays, React Js Basics, Creating the first Application of React native cli and expo cli environment, RN components view, text, button, textinput, alert, More on RN components TouchableOpacity, TouchableHightlight, TouchableNoFeedback, Hooks, StyleSheet, ScrollView, useEffect, FlatList, Image, Pressable, Introduction to React Navigation, Stack Navigator, Tab Navigator, Drawer navigator & Nested navigators, Themes, Dark/Light Theme and custom themes, AsyncStorage, Expo secure store, Working on Fetch Api, Context Api etc, Real time database (Firebase) and authentication

Student Outcomes (SO)

S.#	Description
2	Identify, formulate, research literature, and solve complex computing problems reaching substantiated conclusions using fundamental principles of mathematics, computing sciences, and relevant domain disciplines
3	Design and evaluate solutions for complex computing problems, and design and evaluate systems, components, or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations
4	Create, select, adapt and apply appropriate techniques, resources, and modern computing tools to complex computing activities, with an understanding of the limitations
5	Function effectively as an individual and as a member or leader in diverse teams and in multi-disciplinary settings.
9	Recognize the need, and have the ability, to engage in independent learning for continual development as computing professional.

Intended Learning Outcomes

Sr.#	Description	Blooms Taxonomy Learning Level
CLO -4	Apply the concepts of JavaScript and React Native for mobile applications.	*Applying*
CLO - 5	Develop advance mobile applications with multiple screens and APIs having persistent storage.	*Creating*

Lab Assessment Policy

The lab work done by the student is evaluated using rubrics defined by the course instructor, viva-voce, project work/performance. Marks distribution is as follows:

Assignments	Lab Mid Term Exam	Lab Terminal Exam	Total
25	25	50	100

Note: Midterm and Final term exams must be computer based.

List of Labs

Lab 01

JavaScript Basics and Functions

Objective:

The objective of this lab will learn the students about the basics of JavaScript programming basics and the functions with the help of examples and learning tasks.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working with JS basics (datatypes).
- Implementation of JS loops
- Implementation of JS functions

Instructor Note:

As a pre-lab activity, read Chapter 03 from the text book "JavaScript For Impatient Programmers, Edition 2022".

1) Useful Concepts:

In this lab, we will learn about basic JS variables and functions. We will discuss these types of JS variables.

- Number
  var a = value;
- String
  var b = "value"
- Object (Arrays)
  var c = {key: value} pair

To declare a variable in JavaScript; we will use keyword "var". Also in this lab we will look at JS functions.

To write a function, we have function calling and its definition. To call a function in JS, we will write it as follows.

- NameOfFunction() //for zero argument function
- NameOfFunction(argument) //for single argument function
- NameOfFunction(argument1, argument2,…..) //multiple argument function

To define a function we will use the following syntax.

- function NameOfFunction() //keyword ‘function’ and followed by function name which is called.

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*15 mins*	*Low*	*CLO4*
2	*15 mins*	*Low*	*CLO4*
3	*15 mins*	*Low*	*CLO4*

Activity 1:

1. ***Get used to the editor, run first JS program.***
2. ***Write biography about yourself and print on console.***
3. ***use 'var' to store your biography in variables, use appropriate primitive types.***
4. ***Create JS Object for your biography key-value pairs. At least 4-5 keys, nested JS Object for Address, DegreePrograms etc***
   e.g. { name: 'Some Name', age: 34, address: { }, degreeProgram: {} }
   Print biography on console (do not print entire object as it is)

Solution:

```
var bio = {
    name: 'Zaheer',
    gender: 'male',
    age: 33,
    address: {
        home: 'h#89, st98',
        city: 'LA',
        country: 'US'
    }
}
console.log(bio.address.city)
```

Output

LA

Activity 2:

Find the sum of all the multiples of x or y below z.
If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. (3+5+6+9). Find the sum of all the multiples of x or y below z.

```
function multiSum(z, x, y) {

    var sum=0;
```

```
    for (let index = 2; index < z; index++) {

        if(index%x==0 || index%y==0){

            sum+=index;

        };    };

    return sum; };

console.log(multiSum(10,3,5));
```

Output

23

Activity 3:

Implement min and max methods which returns minimum and maximum value of supplied arguments. Implement your own algorithm to find the minimum and maximum value.
- min(4,8,1,3) // returns 1
- max(4,6,5,3,2) // returns 6

```
function min(...params) {
    var min = params[0];
    for (let index = 0; index < params.length; index++) {
        if(min>params[index]){
            min=params[index]
        }
    }
    return min;
}
function max(...params) {
    var max = params[0];
    for (let index = 0; index < params.length; index++) {
        if(max<params[index]){
            max=params[index]
        }
    }
    return max;
}
console.log(max(4,8,1,3));
```

Output

8

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Implement a program with four functions (add, subtract, multiply and divide). Each function should have different number of arguments passed.

- *First function 'add' should check the undefined arguments within the defined function.*
- *Second function 'subtract' should use the ES6 default parameter to tackle the same problem.*
- *Third function 'multiply' should use the ES6 rest parameters to multiply each argument with one another.*
- *Fourth 'divide' should use the 'Arguments' object to finish the job.*

Lab Task 2

Implement a generic method named SolveThis() which takes a JS object. depending upon the key, it performs the operation and returns another object with the result.

For Example:

```
SolveThis({sum: [3,2,4], max: [2,4,3,5], min: [5,3,4,3]}) // returns { sum: 9, max: 5, min: 3 }
```

It should perform above implemented functions inside, such as, round, abs, ceil, floor, min, max, random etc

Hint:

```
// Create Object dynamically with dynamic keys
var res = {};
res['sum'] = 6;
res['min'] = 7;
console.log(res); // output: Object {sum: 6, min: 7}
```

Lab 02

JavaScript ES6 Functions & Classes

Objective:

The objective of this lab will learn the students about the basics of JavaScript ES6 functions and classes with the help of examples and learning tasks.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working with built-in functions
- Implementation of ES6 arrow functions
- Implementation of ES6 classes and OOP concepts

Instructor Note:

As a pre-lab activity, read Chapter 07 from the text book "JavaScript For Impatient Programmers, Edition 2022".

1) Useful Concepts:

In this lab we will see how we can write functions in different ways introduced in ES6. Similar to previous lab 1, where we have seen the functions calling and its definition, let's see how we can call and define functions and classes in ES6.

- We can write the function as expression with the following syntax.

```
const square = function(number) { return number * number }
var x = square(4)
```

- We can also write it as arrow functions. Let's take an example for that.

```
hello = () => {
   return "Hello World!";
}
console.log(hello())
```

- The arrow function gets much shorter when it has only single statement.

```
hello = () => "Hello World!";
console.log(hello())
```

- Class syntax has 2 components i.e. class expression and class declaration. To write a class in JS, we will follow this structure.

```
let Rectangle = class Rectangle2 {
   constructor(prop) {
      this.prop = prop;
   }};
```

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	***20 mins***	***Low***	***CLO4***
2	***20 mins***	***Low***	***CLO4***

Activity 1:

Implement following methods just like round() method using ES6 Arrow Function. These methods would be wrap around Math JS Functions:

- ***abs***
- ***ceil***
- ***floor***

Solution:

```
round=(...args)=>{
    arr=[]
    args.forEach((element,index)=>{
        arr[index]=Math.round(element);
        })
        return arr;

}

console.log(round(1.2, 4.3, 7.7))
```

Output

[1, 4, 8]

Activity 2:

Create a generic ES6 Function which takes the operation name to perform and the data as arguments for the methods created in above tasks.

Solution:

```
floor=(...args)=>{
    arr=[]
    args.forEach((element,index)=>{
        arr[index]=Math.floor(element);
        })
```

```
        return arr;}

genericOpertaions=(operation,...args)=>{
    if(operation==='round'){
        return round(...args);
        }
        else if(operation==='abs'){
            return abs(...args)    }
            else if(operation==='ceil'){
                return ceil(...args)    }
                else if(operation==='floor'){
                    return floor(...args)    }
                    else{
                        return "Invalid Operation"
                        }
}

console.log(genericOpertaions('floor',1.3,2.4,3.6,4.2,5.5));
```

Output

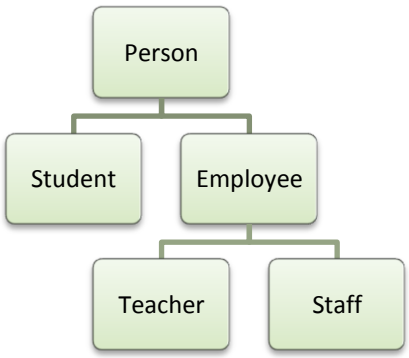
```
[ 1, 2, 3, 4, 5 ]
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Implement JavaScript function-based classes (NOT ES6 CLASSES) for the following scenario:



Student Management System containing Person, Student, Employee, Teacher, Staff and Courses classes.

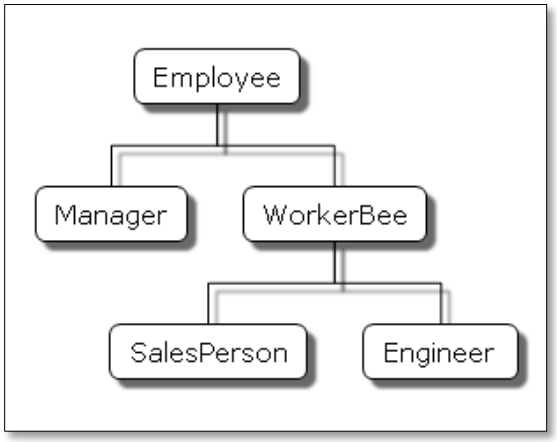
- *Student and Employee classes are the derived class of Person*
- *Teacher and Staff are derived classes of Employee*
- *Person class contains 3-4 common fields of Student and Employee. Initialize the fields with the default value*

- *Employee class contains 3-4 common fields of Teacher and Staff (such as, department, designation, salary etc)*
- *Student, Teacher and Staff contains the fields specific to their role*
- *Courses has aggregation type of association with Student and Teacher classes*

Finally, create two objects of Student, Teacher and Staff each and print their information through created objects on console.

Lab Task 2

Implement JavaScript ES6 classes for the following scenario:



- `Employee` has the properties `name` (whose value defaults to the empty string) and `dept` (whose value defaults to "general").
- `Manager` is based on `Employee`. It adds the `reports` property (whose value defaults to an empty array, intended to have an array of `Employee` objects as its value).
- `WorkerBee` is also based on `Employee`. It adds the `projects` property (whose value defaults to an empty array, intended to have an array of strings as its value).
- `SalesPerson` is based on `WorkerBee`. It adds the `quota` property (whose value defaults to 100). It also overrides the `dept` property with the value "sales", indicating that all salespersons are in the same department.
- `Engineer` is based on `WorkerBee`. It adds the `machine` property (whose value defaults to the empty string) and also overrides the `dept` property with the value "engineering".

Lab 03
JavaScript Arrays & React JS Basics

Objective:

The objective of this lab will learn the students about the basics of JavaScript Arrays and React JS basics with the help of examples and learning tasks.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Implementation of JS built-in array functions
- Working with React JS basics
- Implementation of React Js components, function based Hooks and class based states

Instructor Note:

As a pre-lab activity, read Chapter 12 from the text book "JavaScript For Impatient Programmers, Edition 2022".

1) Useful Concepts:

In this lab, we will work on the basics of React JS i.e. related to basic syntax of react JS and JS arrays functions. An array in JavaScript is an object. To write an array (object), we can use the square brackets and separate each item using a comma. The syntax is written as follows.

```
var number = [1,2,3,4];
```

We can apply different array functions on a single or multiple arrays with the following syntax.

- concat function → Array1.concat(Array2)
- copyWithin function → array.copyWithin(target, start, end)
- fill function → array.fill(value, start, end)
- slice → array.slice(start, end)
- pop → array.pop()
- push → array.push(item1, item2, ..., itemX)
- flat → array.flat()
- sort → array.sort()
- reverse → array.reverse()

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*15 mins*	*Low*	*CLO4*
2	*15 mins*	*Low*	*CLO4*
3	*15 mins*	*Low*	*CLO4*

Activity 1:

Write a JS function which takes an array of numbers and returns following:
Sum of all numbers

Example: [2, 4, 3]
Returns: 9

Solution:

```
const numbers = [10, 20, 30, 40]
add = (a, b) =>  a + b
const sum = numbers.reduce(add)
console.log(sum)
```

Output

```
100
```

Activity 2:

Define a function called cleanNames that accepts an array of strings containing additional space characters at the beginning and end. The cleanNames() function should use the array map method to return a new array full of trimmed names

Solution:

```
function cleanNames(arr){
  return arr.map(name => name.trim())
}
console.log(cleanNames([" avengers", "captain_america", "ironman",
"black panther"]))
```

Output

["avengers", "captain_america", "ironman", "black panther"]

Activity 3:

Complete the method/function so that it converts dash/underscore delimited words into camel casing. The first word within the output should be capitalized only if the original word was capitalized (known as Upper Camel Case, also often referred to as Pascal case).

Examples
"the-stealth-warrior" gets converted to "theStealthWarrior"

Solution:

```
function toCamelCase(str){
  str = str.split('');
  return str.map(function(el, i){
    if(el == '-' || el == '_'){
      el = str[i+1].toUpperCase();
      str.splice(i+1, 1);
    }
    return el;
  }).join('');
}

console.log(toCamelCase("the_stealth_warrior"))
```

Output

theStealthWarrior

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

You will be given an array of numbers. You have to sort the odd numbers in ascending order while leaving the even numbers at their original positions.

Example:
[7, 1] => [1, 7]
[5, 8, 6, 3, 4] => [3, 8, 6, 5, 4]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0] => [1, 8, 3, 6, 5, 4, 7, 2, 9, 0]

Lab Task 2

Modify App.js file and write your Biography (personal information, education, skills) in tabular form.

Lab Task 3

Reading: If you are not familiar with components and props, visit this link: https://reactjs.org/docs/components-and-props.html
Create a React App which Displays your CV.
Now, CV App contains several components to represent different type of information. Each Component should receive React Props to Display Information. Which means, Component displays information based on the props sent by the component (just like demonstrated in class).
CV should contain following Sections. Each Section would be a React Component:
Header: Picture, Name
Information: Address, Contact Details (phone, email etc)
Education: FSc, BSCS (Institute, Degree Name, Start Date, End Date, Description etc)
Experience: Company Name, Start and End Date, Responsibilities
Skills: Display a List of Skills
Hobbies: Display a List of Hobbies
Hint:
To Display Local image in React JS
**
Otherwise, it's same as html:
**

Lab 04

Creating the first ReactNative and Expo Cli Application

Objective:

The objective of this lab will learn the students about the complete installation of react native and expo apps. Also creating the first application in react native and expo environment.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Installation Guide
- Creating the first expo cli application
- Creating the first react native application
- Running the first application on device

Instructor Note:

As a pre-lab activity, read Chapter 03 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab, we will introduce React Native cli and Expo cli environment and installation setup. For expo cli, you need the latest npm installation from the website and then the expo installation with the command line. You can also use expo cli environment using snack expo website.

For React native cli, you need the npm installation and jdk installation. For that we can also use chocolatey library to install both the libraries at once. After this, we will need Android studio environment setup. We need three things from android studio i.e. android sdk, android sdk platform and android virtual device (if you don't have a physical device android or Ios). After the installation we have to Configure the 'ANDROID_HOME' environment variable. Once everything is setup, we can create a new react native application by using the command line to initialize the project.

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	***20 mins***	***Low***	***CLO4***
2	***25 mins***	***Low***	***CLO4***

Activity 1:

Create a Expo cli app named "AwesomeProject".

Solution:

```
Install Node v12 or greater
Run the command ( npm install -g expo-cli )

Followed by these commands,

expo init AwesomeProject

cd AwesomeProject
npm start # you can also use: expo start
```

Output

First Expo Application created successfully.

Activity 2:

Create a React Native cli application named "AwesomeProject".

Solution:

```
Development OS: windows
Target OS: android

Installting Dependencies: JDK and android studio

NDK, JDK:
We recommend installing Node via Chocolatey, a popular package
manager for Windows.

Open an Administrator Command Prompt (right click Command Prompt and
select "Run as Administrator"), then run the following command:
choco install -y nodejs-lts openjdk11

Android development environment:
Download and install Android Studio. While on Android Studio
installation wizard, make sure the boxes next to all of the following
items are checked:

  • Android SDK
  • Android SDK Platform
  • Android Virtual Device
  • Download SDK Platform 29>
```

```
Configure the ANDROID_HOME environment variable:

The React Native tools require some environment variables to be set up
in order to build apps with native code.

   • Open the Windows Control Panel.
   • Click on User Accounts, then click User Accounts again
   • Click on Change my environment variables
   • Click on New... to create a new ANDROID_HOME user variable that
     points to the path to your Android SDK:

Run this command for creating a new project:

npx react-native init AwesomeProject

Start your application:

npx react-native run-android
```

Output

```
theStealthWarrior
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create an account on Snack and create the first expo cli application on it. Rename the application as "FirstExpoApplication". Run the application on built-in android and IOS emulators. Also, scan the QR code and run the application on actual device (android/IOs). Lastly, remove the unnecessary code from snack and application should only have two react native components having view and text, having "Hello world" as output.

Lab Task 2

Create a React Native Cli application after following the instructions from here (https://reactnative.dev/docs/environment-setup) . Run the application on your android/IOs environment.

Tool for coding: *VS Code*

(https://code.visualstudio.com/download)

Lab 05

RN components view, text, button, textinput, alert

Objective:

The objective of this lab will learn the students about the react native components that is View, Text, Button, TextInput, Alert and more.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Basics of React Native application
- Working with RN components like view, text, button, textinput, alert

Instructor Note:

As a pre-lab activity, read Chapter 10 and 11 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab, we will start working with the react native components. To work with RN, we can either write a code inside class component or a function component. To write a RN simple "hello world" program, we can write the program inside the functional component as follows.

```
import React from 'react';
import { Text } from 'react-native';
const Cat = () => {
  return (     <Text>Hello, I am your cat!</Text>   ); }
export default Cat;
```

Similarly, if we want to write a RN program using a class component, we can do the following.

```
import React, { Component } from 'react';
import { Text } from 'react-native';
class Cat extends Component {
  render() {
    return (
      <Text>Hello, I am your cat!</Text>    );  } }
export default Cat;
```

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*45 mins*	*Medium*	*CLO4*

Activity 1:

Make a basic counter application in react native cli, Expo cli and Snack environment.

Solution: [App.js] code

```
import { StatusBar } from 'expo-status-bar';
import React, { useState } from 'react';
import { Button, StyleSheet, Text, View } from 'react-native';

export default function App() {

  const [getCount, setCount] = useState(0);

  return (
    <View style={styles.container}>
      <Text style={{fontSize:18, fontWeight:'bold'}}>Counter
Application</Text>
      <View style={styles.just} flexDirection='row'>
        <Button title='Decrease' color='red'
onPress={()=>{setCount(getCount-1)}} disabled={getCount==0}></Button>
        <Text>  {getCount}  </Text>
        <Button title='Increase' color='green'
onPress={()=>{setCount(getCount+1)}}></Button>
      </View>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  just: {
    padding:10,
    justifyContent:'space-evenly'
```

```
  },
});
```

Output

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Design the game Layout which contains:

- *Title of the Game*
- *User Input Text Area (User will input the number through onscreen buttons)*
- *Buttons (0-9) for input number*

Lab Task 2

Design a login screen having following components in given sequence.

- *Title of the application as "Login".*
- *'user name' and 'password' as a TextInput field.*
- *A 'submit' button to check whether entered credential are correct or not*
    - *Set a check on user name and password*
    - *If any of the inputText field is empty, then button should be disabled*
    - *Show alert "Success login", if the entered username is "cspeople" and password is "computerscience"*
    - *If any of them is incorrect, show wrong username or password.*

Lab 06
RN components and Hooks

Objective:

The objective of this lab will learn the students about the more react native components like touchableopacity, hooks and StyleSheet.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Implementation of React native components like touchableopacity etc
- Implementation of Hooks
- Implementation of StyleSheet

Instructor Note:

As a pre-lab activity, read Chapter 4 and 5 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab, we will learn about different react native components concepts, their usage and syntax. These components are as follows.

- TouchableOpacity → when we want to style the button, we use this component.
- TouchableHightlight → similar to touchableopacity, but when pressed down the opacity of the wrapped view is decreased, which allows the underlay color to show through, darkening or tinting the view.
- TouchableNoFeedback → Touchable but shows no effect.
- StyleSheet → for maintaining styles for different components
- ScrollView → In order to slide down the screen

In this lab, we will also work on React hooks for state maintenance inside functional component. To initialize a hook, we can use the following syntax.

const [getValue, setValue] = useState(defaultValue)

The defaultValue will suggest the type of data inside the getValue. We can use all JS datatypes for hooks.

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*45 mins*	*Medium*	*CLO4*

Activity 1:

Make a To-Do List App using React Native components. The application should add, delete and update items from the list using Hooks. Make sure the styling is same as shown in mobile screen.

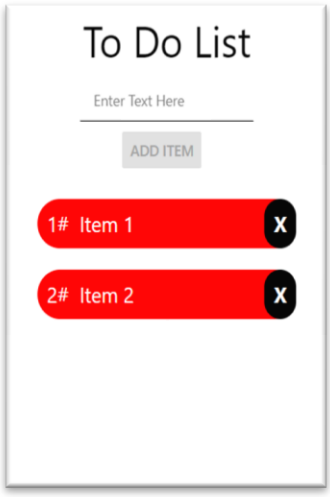
Solution:

```
import React, {useState} from 'react';
import { TouchableOpacity,ScrollView, Button, TextInput, Text, View, S
tyleSheet, Keyboard } from 'react-native';

export default function App() {
  const [item, setItem] = useState("")
  const [getlist, setList] = useState([])
  const [editItem, setEditItem] = useState(0)
  const updateList = () =>
  {
    setList(list => getlist.map(element => element.key === editItem ?
{key: element.key, data:item} : element));
    setItem("")
    setEditItem(0)
  }

  const updateItems = (item) =>
  {
    setItem(item.data)
    setEditItem(item.key)
  }
  const additems = () =>
  {
    setList([...getlist,{key:Math.random().toString(), data:item}])
    setItem("")
    Keyboard.dismiss()
  }

  const deleteItem = (key) =>
  {
    setList(list => getlist.filter(element => element.key != key))

  }
```

```
  return (
    <View style={{ alignItems:'center'}}>
    <Text style={{fontSize:40}}>
    To Do List
    </Text>

    <TextInput style={{borderBottomWidth:1, padding:10, margin:10}} pl
aceholder=" Enter Text Here " value={item} onChangeText={setItem}>
    </TextInput>
    <View style={{ flexDirection:"row"}}>
    <View style={{marginRight:10}}>
    <Button title={editItem===0 ? "Add item": "Update item"} onPress={
editItem===0 ? additems: updateList} disabled={item.length<=0}>
    </Button>
    </View>
    </View>
    <Text> {item}
    </Text>
    <ScrollView style={{width:'100%'}}>
        {getlist.map((item, index) =>
        <TouchableOpacity key={item.key} activeOpacity={0.5} onPress =
 {()=>updateItems(item)}>
        <View style ={{flexDirection:'row',  width:'80%', alignSelf:'c
enter',  backgroundColor:'red', margin:10, borderRadius:50, justifyCon
tent:'space-between'}}  key={item.key}>

        <Text style={{fontSize:20, margin:10, color:'white'}}>
        {index + 1}#  { item.data} {item.key}
        </Text>
        <TouchableOpacity onPress={() => deleteItem(item.key)}>
        <View style={{backgroundColor:'black', borderRadius:50,padding
:5, justifyContent:'center'}}>
        <Text style={{fontSize:20, margin:5, color:'white', fontWeight
:'bold'}}>X
        </Text>
        </View>
        </TouchableOpacity>

        </View>
        </TouchableOpacity>
         )}
    </ScrollView>
    </View>
  );
}
```

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

Output

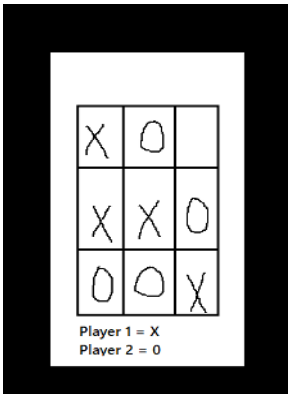


3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Design the basic layout of Tic tac toe application using React native. (9 buttons , Player text and their symbols 0 or X). Refer to the below screenshot for the design.



Lab Task 2

Design and develop a React native application for the following program given below. The student name and marks of 3 subjects should be input by the user. The submit button should be disabled if any of the field is empty. After submit, it should be added to the listView below. The Reset button should remove all the items from the listView.

Lab 07

useEffect, Image and FlatList

Objective:

The objective of this lab will learn the students about useEffect function and remainings react native components.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Understanding of useEffect
- Implementation of Image component and FlatList component

Instructor Note:

As a pre-lab activity, read Chapter 2 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab we will learn more about the react native components like images and flatlist. To use images inside react native app, we will use image component. The FlatList component, as the name suggests will show the list of items (array particularly) on the screen.

The flatlist component has two required props i.e. data and renderItem.

Later, in this lab we will also discuss about the usage of useEffect function. In the previous lab, we have talked about hooks for state maintenance. So, whenever there is a state change, we will have useEffect in process. The syntax for using the useEffect is as follows.

```
useEffect( () => {
//statements
}, [dependency])
```

The dependency argument is optional.

2) Solved Lab Activities

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*20 mins*	*Low*	*CLO4*
2	*25 mins*	*medium*	*CLO4*

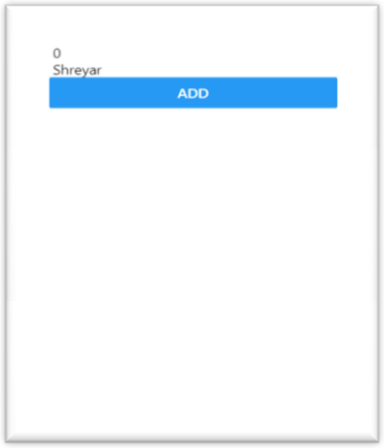
Activity 1:

A simple example of useEffect, where a name of a person is changed after a certain condition using Hooks.

Solution:

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

export default function Example() {
  const [count, setCount] = useState(0)
  const [name, setName] = useState("Shreyar")
  useEffect (() =>
{
    if(count == 5)
    {
      setName("Zeeshan")
    }
 } )
  return (
  <View style ={{margin:30}}>
    <Text> {count}
    </Text>
    <Text> {name} </Text>
    <Button title="Add" onPress={() => setCount(count + 1)}> </Button>
  </View>
  );
}
```

Output



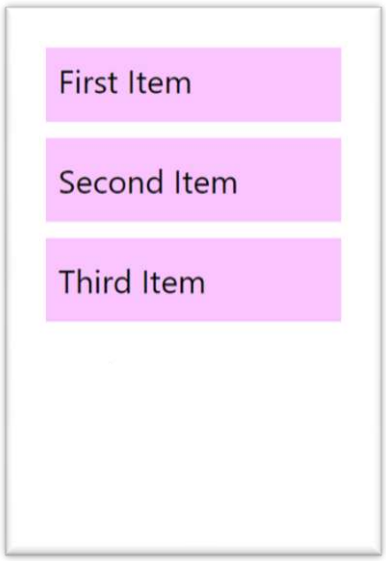
Activity 2:

A JSON data is shown in DATA variable below. Use the FlatList component to show the "title" of each item in JSON data.

```
import React from 'react';
import { SafeAreaView, View, FlatList, StyleSheet, Text, StatusBar }
from 'react-native';
const DATA = [
  {
    id: 'bd7acbea-c1b1-46c2-aed5-3ad53abb28ba',
    title: 'First Item',
  },
  {
    id: '3ac68afc-c605-48d3-a4f8-fbd91aa97f63',
    title: 'Second Item',
  },
  {
    id: '58694a0f-3da1-471f-bd96-145571e29d72',
    title: 'Third Item',
  },
];
const Item = ({ title }) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const App = () => {
  const renderItem = ({ item }) => (
    <Item title={item.title} />
  );

  return (
    <SafeAreaView style={styles.container}>
      <FlatList
```

```
        data={DATA}
        renderItem={renderItem}
        keyExtractor={item => item.id}
      />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: StatusBar.currentHeight || 0,
  },
  item: {
    backgroundColor: '#f9c2ff',
    padding: 20,
    marginVertical: 8,
    marginHorizontal: 16,
  },
  title: {
    fontSize: 32,
  },
});

export default App;
```

Output



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Discount Calculator App: When we go out for shopping especially when there is sale on different shops and outlets. Most of the times, discount percentage is specified along with original price. Every time we need to calculate the price after discount, and we do this repeatedly for each item on sale.

To solve this issue, we are going to develop a small React Native Application which will calculate the discount

This is a simple app at this stage and contains two input fields:

- *Original Price*
- *Discount Percentage*

Implement the functionality of the App. As soon as user type the Original Price or Discount Percentage, Display following information:

- *You Save*
- *Final Price*

Design and improve the look and feel of the App (Add nice header on the top, Color Scheme, Alignment of the Content, Font-Size etc).

Adding constraints on the input fields, such as, it can take only Numbers and Positive Numbers, Discount cannot be greater than 100. Calculated amount should be 2 decimal points.

This feature is not present in most of the apps available on App Stores. Sometimes, we move around during shopping and want to come back and see the after-discount price again of the item, to avoid typing again the required inputs, we want to maintain a history of the calculations we performed.
Now, the question is, when do we save the calculations? We can provide a Save Button to save the calculations and View History.
In this task, we save the calculations with the save button

A button will help the user to see the history. Use Modal (https://reactnative.dev/docs/modal) to display the history of calculations.
It should contain following information:

- *Original Price*
- *Discount Percentage*
- *Price After Discount*

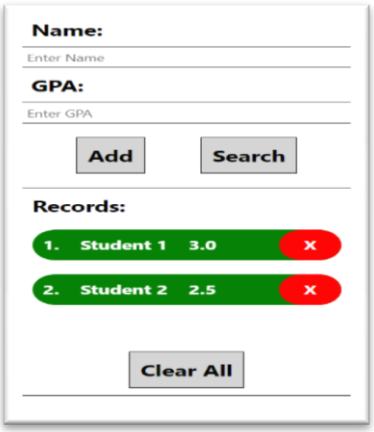
History could show something like this

Original Price	–	*Discount*	=	*Final Price*
5000	–	*25%*	=	*4000*
4000	–	*10%*	=	*3600*

1000 – 10% = 900

Lab Task 2

Design and develop the below application for maintaining the student records. The screens shows the addition, searching and deletion of any student record.



Lab 08

Intro. to React Navigation (Stack navigator)

Objective:

The objective of this lab will learn the students about the introduction of react navigation and its navigators

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working on screen headers
- Working on multiple screens application
- Implementation of Stack navigator
- Passing of data from one screen to another

Instructor Note:

As a pre-lab activity, read Chapter 6 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab, we will learn about the React navigation. Here we will see the presence of header and also involvement of multiple screens. Let's discuss the first navigator which is a "Stack" navigator. For that, we have to create a function named "createNativeStackNavigator()" inside the component. Then inside the JSX, we will use the "NavigationContainer" component. This components has two childs i.e. a navigator and screen components.

- The navigator component will tell about the routes of different screens and also the overall options on all the present screens.
- The screen component will contain name and component as a prop.

The basic syntax is as follows.

```
<NavigationContainer>
  <Stack.Navigator>
    <Stack.Screen name="Name" component={ComponentName} />
  </Stack.Navigator>
</NavigationContainer>
```

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*20 mins*	*Low*	*CLO5*
2	*25 mins*	*Medium*	*CLO5*

Activity 1:

Create a native stack navigator for a single screen and show its header.

Solution:

```
import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-
stack';

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
export default App;
```

Output

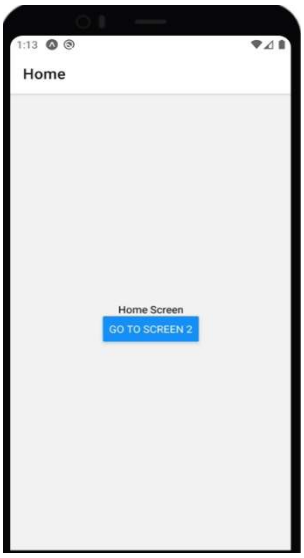
Activity 2:

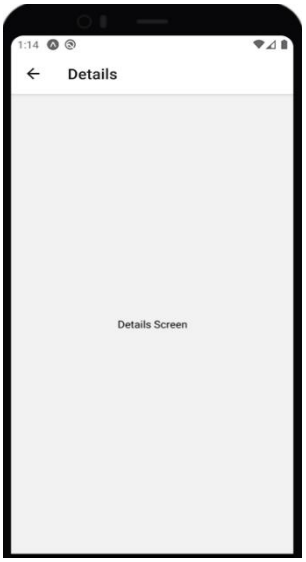
Use the same code from activity 1, add a second screen to the stack navigator. Add a button to the first screen and navigate to the newly added screen.

```
import * as React from 'react';
import { View, Text,Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-
stack';

function HomeScreen({navigation}) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'cen
ter' }}>
      <Text>Home Screen</Text>
      <Button title="Go to screen 2" onPress={()=> navigation.navigate
("Details")}> </Button>
    </View>
  );
}

function DetailsScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'cen
ter' }}>
      <Text>Details Screen</Text>
    </View>
```

```
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Output





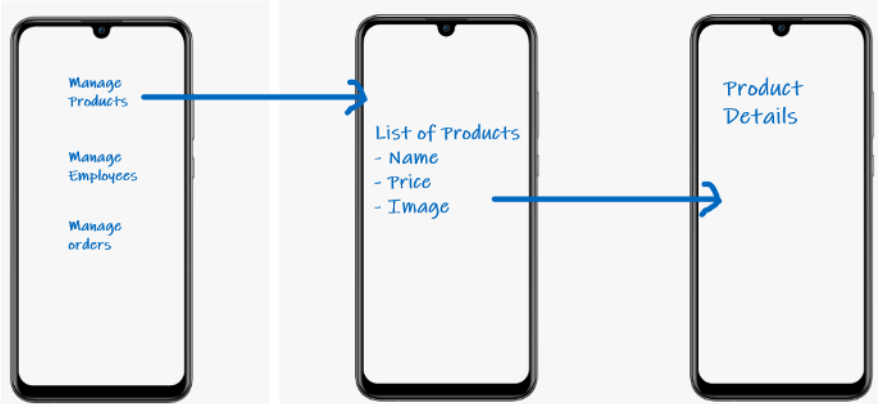
3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

We will develop an Admin Mobile Application for an Online Store. App will have multiple screens to perform different operations.

This is a brief sketch of the screens:



Create Stack Navigator to navigate between the screens.
You will be using hardcoded lists to display data. Data can be loaded and displayed on listing screen (products list, employees list, orders list). Further navigating from the listing screen will send the data to the next screen.
Implement proper navigation flow with the appropriate header information of each screen, such as, display product name on the header of product details screen and same for employee details screen.
You can get product images from: https://unsplash.com/ or use any other online resource for images. Or even you can get the products from any of your online stores, such as, Daraz/Elo etc
Home screen does not need to have the header.

Task # 1
Create screens. There would be around 7 screens.

1. *Home Screen*
2. *Products List*
3. *Product Details*
4. *Employees List*
5. *Employee Details*
6. *Orders List*
7. *Order Details*

Task # 2
Set up the Stack Navigator. Put all the screens in navigator and create dummy buttons for navigation between screens.
Home Screen containing buttons (do not use default buttons, design your own buttons with TouchableOpacity) to navigate to other screens. These are as displayed in the sketch.

Task # 3
Manage Products will navigate to the list of products screen. You can use hardcoded list of products to be displayed.
When user taps on the product, it navigates to the Product Details screens where detailed information of the product is displayed.

Task # 4

Manage Employees will navigate to the list of Employees screen. Where Name and the Designation of an employee is displayed.
When user taps on one of the employee, it navigates to the employee details screen

Task # 5
Manage Orders will navigate to the list of Orders screen. Where Order Number, Product Name, Price is displayed.
When user taps on one order, it navigates to the order details screen to see details of the order, such as, customer information, order date, shipping status etc

Lab 10

Tab, Drawer & Nested Navigators

Objective:

The objective of this lab will learn the students about the stack and drawer navigator and its customization.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Implementation of tab and drawer navigation
- Working on customization of navigators
- Implementation of nested navigators

Instructor Note:

As a pre-lab activity, read Chapter 6 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab, we will learn about the other two types of navigator which are tab and drawer navigator. To use these types of navigator, we have the basic syntax as follows.

- createBottomTabNavigator();

```
<Tab.Navigator>
   <Tab.Screen name="Screen1" component={ComponentName} />
   <Tab.Screen name="Screen2" component={ ComponentName } />
  </Tab.Navigator>
```

- createDrawerNavigator();

```
<Drawer.Navigator>
   <Drawer.Screen name=" Screen1" component={ ComponentName } />
   <Drawer.Screen name=" Screen2" component={ ComponentName } />
  </Drawer.Navigator>
```

The bottom tab navigator will show the different tabs on screen with different items shown. The drawer navigator will show a drawer, which we can drag from the left side of the screen.

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*45 mins*	*High*	*CLO5*

Activity 1:

From the activity 2 previous lab, implement nested navigators. Implement any one of the navigator between tab or drawer navigator along with stack navigator.

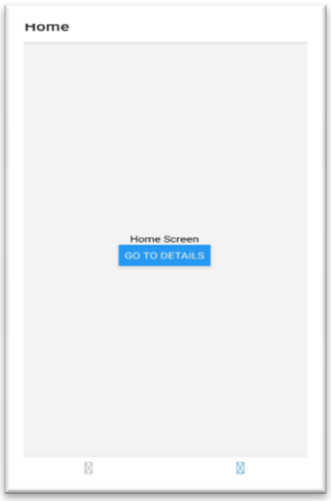
Solution:

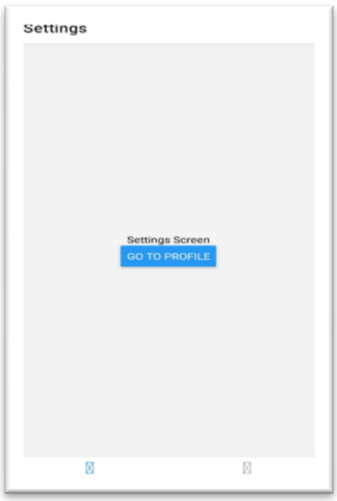
```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer, useFocusEffect } from '@react-
navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-
stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-
tabs';

function SettingsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Settings Screen</Text>
      <Button
        title="Go to Profile"
        onPress={() => navigation.navigate('Profile')}
      />
    </View>
  );
}

function ProfileScreen({ navigation }) {
  useFocusEffect(
    React.useCallback(() => {
      alert('Screen was focused');
      // Do something when the screen is focused
      return () => {
        alert('Screen was unfocused');
        // Do something when the screen is unfocused
        // Useful for cleanup functions
      };
    })
```

```
  );

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Profile Screen</Text>
      <Button
        title="Go to Settings"
        onPress={() => navigation.navigate('Settings')}
      />
    </View>
  );
}

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

function DetailsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Details Screen</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.push('Details')}
      />
    </View>
  );
}
const Tab = createBottomTabNavigator();
const SettingsStack = createNativeStackNavigator();
const HomeStack = createNativeStackNavigator();

export default function App() {
```

```
  return (
    <NavigationContainer>
      <Tab.Navigator screenOptions={{ headerShown: false }}>
        <Tab.Screen name="First">
          {() => (
            <SettingsStack.Navigator>
              <SettingsStack.Screen
                name="Settings"
                component={SettingsScreen}
              />
              <SettingsStack.Screen name="Profile"
component={ProfileScreen} />
            </SettingsStack.Navigator>
          )}
        </Tab.Screen>
        <Tab.Screen name="Second">
          {() => (
            <HomeStack.Navigator>
              <HomeStack.Screen name="Home" component={HomeScreen} />
              <HomeStack.Screen name="Details"
component={DetailsScreen} />
            </HomeStack.Navigator>
          )}
        </Tab.Screen>
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Output





3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create an application of your own choice with the tab, drawer and stack navigators implemented. Configuration of header bar should be done according to this https://reactnavigation.org/docs/headers

There should be header buttons. Follow this link https://reactnavigation.org/docs/header-buttons

There should be some kind of data passing params, initial params and update params between the navigators.
Follow this link https://reactnavigation.org/docs/params

Lab 11

Themes

Objective:

The objective of this lab will learn the students about the built-in themes and customization of themes in react native applications.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working with built-in themes
- Implementation of custom themes

Instructor Note:

As a pre-lab activity, read Chapter 7 from the text book "React Native in Action, Nader Dabit, 1st Edition (2019), Manning Publications".

1) Useful Concepts:

In this lab we will work on the theme of the application. Themes allow you to change the colors of various components provided by React Navigation. You can use themes to:

- Customize the colors match your brand
- Provide light and dark themes based on the time of the day or user preference

A theme is a JS object containing a list of colors to use. It contains the following properties:

- dark (boolean): Whether this is a dark theme or a light theme
- colors (object): Various colors used by react navigation components:
    - primary (string): The primary color of the app used to tint various elements. Usually you'll want to use your brand color for this.
    - background (string): The color of various backgrounds, such as background color for the screens.
    - card (string): The background color of card-like elements, such as headers, tab bars etc.
    - text (string): The text color of various elements.
    - border (string): The color of borders, e.g. header border, tab bar border etc.

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*45 mins*	*Medium*	*CLO5*

Activity 1:

Implement built-in themes (light and dark theme) using react native on mobile application.

Solution:

```
import * as React from 'react';
import { Button, View, Text, TouchableOpacity } from 'react-native';
import {
  NavigationContainer,
  DefaultTheme,
  DarkTheme,
  useTheme,
} from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-
stack';
import { createDrawerNavigator } from '@react-navigation/drawer';
import { AppearanceProvider, useColorScheme } from 'react-native-
appearance';

function SettingsScreen({ route, navigation }) {
  const { user } = route.params;
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Settings Screen</Text>
      <Text>userParam: {JSON.stringify(user)}</Text>
      <Button
        title="Go to Profile"
        onPress={() => navigation.navigate('Profile')}
      />
    </View>
  );
}

function ProfileScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Profile Screen</Text>
    </View>
```

```
  );
}

function MyButton() {
  const { colors } = useTheme();

  return (
    <TouchableOpacity style={{ backgroundColor: colors.card }}>
      <Text style={{ color: colors.text }}>Button!</Text>
    </TouchableOpacity>
  );
}

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
      <Text>Home Screen</Text>
      <MyButton />
      <Button
        title="Go to Settings"
        onPress={() =>
          navigation.navigate('Root', {
            screen: 'Settings',
            params: { user: 'jane' },
          })
        }
      />
    </View>
  );
}

const Drawer = createDrawerNavigator();
const Stack = createNativeStackNavigator();

function Root() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Profile" component={ProfileScreen} />
      <Stack.Screen name="Settings" component={SettingsScreen} />
    </Stack.Navigator>
  );
}

export default function App() {
```

```
  const scheme = useColorScheme();

  return (
    <AppearanceProvider>
      <NavigationContainer theme={scheme === 'dark' ? DarkTheme :
DefaultTheme}>
        <Drawer.Navigator initialRouteName="Root">
          <Drawer.Screen name="Home" component={HomeScreen} />
          <Drawer.Screen
            name="Root"
            component={Root}
            options={{ headerShown: false }}
          />
        </Drawer.Navigator>
      </NavigationContainer>
    </AppearanceProvider>
  );
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Make a three screen application and implement a custom theme on it using the following properties. Make sure to apply theme on header and components. Also, use the current theme in your own components.

```
const MyTheme = {
  dark: false,
  colors: {
    primary: 'rgb(255, 45, 85)',
    background: 'rgb(242, 242, 242)',
    card: 'rgb(255, 255, 255)',
    text: 'rgb(28, 28, 30)',
    border: 'rgb(199, 199, 204)',
    notification: 'rgb(255, 69, 58)',
  },
};
```

Lab 12

AsyncStorage and Expo Secure Store

Objective:

The objective of this lab will learn the students about the local storage in react native and expo cli application.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working on local storage
- Implementation of AsyncStorage
- Implementation of Expo secure store

Instructor Note:

As a pre-lab activity, read Chapter 4 part 2 from the text book "Fullstack React Native, Shoemaker, Sophia, Djirdeh, Houssein, (2019), Published by Fullstack.io".

1) Useful Concepts:

Async Storage can only store string data, so in order to store object data you need to serialize it first. For data that can be serialized to JSON you can use JSON.stringify() when saving the data and JSON.parse() when loading the data.

- To store data, setItem() is used both to add new data item (when no data for given key exists), and to modify existing item (when previous data for given key exists). The syntax is as follows.

```
const storeData = async (value) => {
  try {     await AsyncStorage.setItem('@storage_Key', value)
  } catch (e) {
    // saving error  } }
```

- To read data, getItem returns a promise that either resolves to stored value when data is found for given key, or returns null otherwise.

```
const getData = async () => {
  try {     const value = await AsyncStorage.getItem('@storage_Key')
    if(value !== null) {       // value previously stored     }   }
catch(e) {
    // error reading value  } }
```

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*45 mins*	*High*	*CLO5*

Activity 1:

Create a simple application for saving an input field data using AsyncStorage and retrieving the data from it.

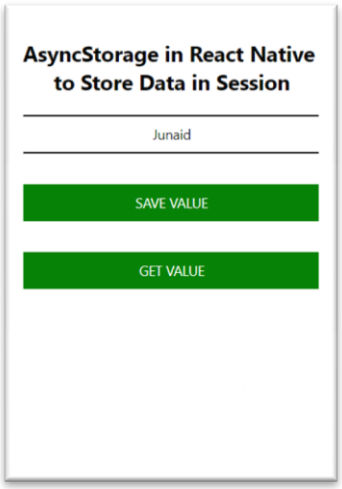
Solution:

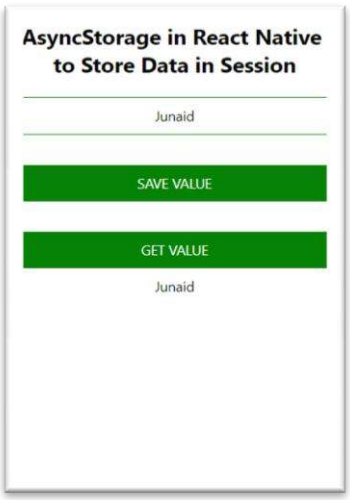
```
import React, { useState } from 'react';
import {
  SafeAreaView,
  StyleSheet,
  View,
  TextInput,
  Text,
  TouchableOpacity,
} from 'react-native';

import AsyncStorage from '@react-native-community/async-storage';

const App = () => {
  const [textInputValue, setTextInputValue] = useState('');
  const [getValue, setGetValue] = useState('');

  const saveValueFunction = () => {
    if (textInputValue) {
      AsyncStorage.setItem('any_key_here', textInputValue);
      setTextInputValue('');
      alert('Data Saved');
    } else {
      alert('Please fill data');
    }
  };

  const getValueFunction = () => {
    AsyncStorage.getItem('any_key_here').then(
      (value) =>
        setGetValue(value)
    );
  };

  return (
```

```
    <SafeAreaView style={{ flex: 1 }}>
      <View style={styles.container}>
        <Text style={styles.titleText}>
          AsyncStorage in React Native to Store Data in Session
        </Text>
        <TextInput
          placeholder="Enter Some Text here"
          value={textInputValue}
          onChangeText={(data) => setTextInputValue(data)}
          underlineColorAndroid="transparent"
          style={styles.textInputStyle}
        />
        <TouchableOpacity
          onPress={saveValueFunction}
          style={styles.buttonStyle}>
          <Text style={styles.buttonTextStyle}> SAVE VALUE </Text>
        </TouchableOpacity>
        <TouchableOpacity onPress={getValueFunction} style={styles.but
tonStyle}>
          <Text style={styles.buttonTextStyle}> GET VALUE </Text>
        </TouchableOpacity>
        <Text style={styles.textStyle}> {getValue} </Text>
      </View>
    </SafeAreaView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 10,
    backgroundColor: 'white',
  },
  titleText: {
    fontSize: 22,
    fontWeight: 'bold',
    textAlign: 'center',
    paddingVertical: 20,
  },
  textStyle: {
    padding: 10,
    textAlign: 'center',
  },
  buttonStyle: {
    fontSize: 16,
```

```
    color: 'white',
    backgroundColor: 'green',
    padding: 5,
    marginTop: 32,
    minWidth: 250,
  },
  buttonTextStyle: {
    padding: 5,
    color: 'white',
    textAlign: 'center',
  },
  textInputStyle: {
    textAlign: 'center',
    height: 40,
    width: '100%',
    borderWidth: 1,
    borderColor: 'green',
  },
});

export default App;
```

Output



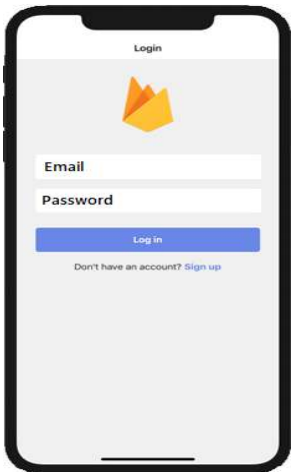


3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Design and develop registration / login screens and store user name, email and password fields in AsyncStorage. There should be a check on password and confirm password during registration process. The email and password should be verified from saved AsyncStorage data. Show alert if email and password is correct else show incorrect.



Lab Task 2

Repeat the same process by creating the Expo-cli application. Store the input field data into Expo secure store instead of AsyncStorage. Show the output similar to above screens.

Lab 13

Working on APIs

Objective:

The objective of this lab will learn the students about the application programming interface.

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working and implementation of different APIs.

Instructor Note:

As a pre-lab activity, read Chapter 5 part 1 from the text book "Fullstack React Native, Shoemaker, Sophia, Djirdeh, Houssein, (2019), Published by Fullstack.io".

1) Useful Concepts:

React Native provides the Fetch API for your networking needs. Fetch will seem familiar if you have used XMLHttpRequest or other networking APIs before. You may refer to MDN's guide on Using Fetch for additional information.

- In order to fetch content from an arbitrary URL, you can pass the URL to fetch:

```
fetch('https://mywebsite.com/mydata.json');
```

- Fetch also takes an optional second argument that allows you to customize the HTTP request. You may want to specify additional headers, or make a POST request:

```
const getMoviesFromApi = () => {

  return fetch('jsonDataUrlHere')
    .then((response) => response.json())
    .then((json) => {
      return json.objectName;
    })
    .catch((error) => {
      console.error(error);
    });

};
```

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*45 mins*	*Medium*	*CLO5*

Activity 1:

A sample JSON data is provided on this link https://reactnative.dev/movies.json . Fetch the data using the api and show the movies titles and their release year.

Sample data:

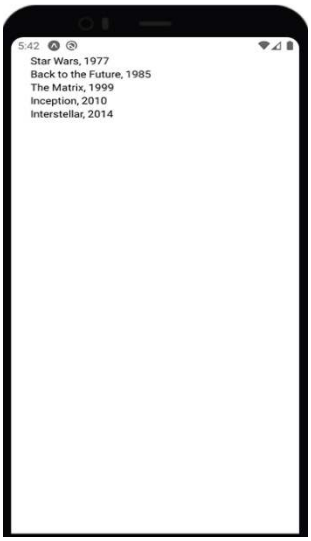

```
{
  "title": "The Basics - Networking",
  "description": "Your app fetched this from a remote endpoint!",
  "movies": [
    { "id": "1", "title": "Star Wars", "releaseYear": "1977" },
    { "id": "2", "title": "Back to the Future", "releaseYear": "1985" },
    { "id": "3", "title": "The Matrix", "releaseYear": "1999" },
    { "id": "4", "title": "Inception", "releaseYear": "2010" },
    { "id": "5", "title": "Interstellar", "releaseYear": "2014" }
  ]
}
```


Solution:

```
import React, { useEffect, useState } from 'react';
import { ActivityIndicator, FlatList, Text, View } from 'react-
native';

export default App = () => {
  const [isLoading, setLoading] = useState(true);
  const [data, setData] = useState([]);

  const getMovies = async () => {
     try {
      const response = await fetch('https://reactnative.dev/movies.jso
n');
      const json = await response.json();
      setData(json.movies);
    } catch (error) {
      console.error(error);
    } finally {
      setLoading(false);
    }
  }
```

```
  useEffect(() => {
    getMovies();
  }, []);

  return (
    <View style={{ flex: 1, padding: 24 }}>
      {isLoading ? <ActivityIndicator/> : (
        <FlatList
          data={data}
          keyExtractor={({ id }, index) => id}
          renderItem={({ item }) => (
            <Text>{item.title}, {item.releaseYear}</Text>
          )}
        />
      )}
    </View>
  );
};
```

Output



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Developing a small application with the help of fetch API. In this application, we will be using free fake API for testing and prototyping. Free fake API provides several API endpoints to test. Each API endpoint return JSON object or an Array of JSON Objects. In this task, we will be using few API endpoints to learn the use of fetch API. We will be creating multiple screens with React Navigation.

Sub-Task # 1 – (Screen 1: List of todo Items)

Use todos API endpoint to display their list in a FlatList. Todo endpoint returns an array of JS Objects. Display only the title of the todo item in FlatList.


```
[
  {
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
  },
  {
    "userId": 1,
    "id": 2,
    "title": "quis ut nam facilis et officia qui",
    "completed": false
  },
…
…
…
]
```


Sub-Task # 2 – (Screen 2: todo Item Details)

Make the todo list as tap-able in previous screen. When user taps on it, move the user to the next screen where you can display the details of the item. These fields can be displayed in this screen:

- Title
- Completion status
- Username

You can get a specific todo item details from API Endpoint OR you can send the details from previous screen. Here is the endpoint to retrieve todo item through the id:

https://jsonplaceholder.typicode.com/todos/1

To display the username, we can get the user details from following endpoint:

https://jsonplaceholder.typicode.com/users/1

where the last parameter is userId in todos list.

Sub-Task # 3 – (Screen 3: User Details)

In previous screen, make the username as tap-able. When user taps on the Username, navigate user to the user details screen and display the information of the user.

API endpoint returns the JS object containing user information:


```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}
```


References:

Snack Example – API Call using FlatList:
https://snack.expo.io/@zaheersani/api-call-and-flatlist-(function-based)
Snack Example – API Call to handle single JS Object:
https://snack.expo.io/@zaheersani/api-call---handling-single-object
Networking in React Native:
https://reactnative.dev/docs/network

Lab Task 2

Develop a React Native application to display basic COVID-19 statistics.
Use following Rapid APIs to fetch the information:

a) *COVID-19: https://rapidapi.com/Gramzivi/api/covid-19-data*
b) *World Population: https://rapidapi.com/aldair.sr99/api/world-population*

This is the first screen of the application which displays COVID-19 statistics. Along with total number you should also display the percentage with respect to the population.

For example: Total Confirmed Cases of the World are 19,05,74,922 and Total World Population is 7,79,47,98,739. Along with the Total Number, we should also calculate and display the percentage for each of the following:

- Confirmed Cases
- Recovered
- Critical Cases
- Deaths
- Last Updated

Sample Request to COVID-19 API:

```
fetch("https://covid-19-data.p.rapidapi.com/totals", {
        "method": "GET",
        "headers": {
                "x-rapidapi-key": "<Your API Key>",
                "x-rapidapi-host": "covid-19-data.p.rapidapi.com"
        }
})
.then(response => {
        console.log(response);
})
.catch(err => {
        console.error(err);
});
```

Sample Response from COVID-19 API:

```
[
    {
        "confirmed": 190574922
        "recovered": 165321635
        "critical": 84517
        "deaths": 4525202
        "lastChange": "2021-06-15T10:59:18+02:00"
        "lastUpdate": "2021-06-15T11:00:06+02:00"
    }
]
```

References:

Snack Example – API Call using FlatList:
https://snack.expo.io/@zaheersani/api-call-and-flatlist-(function-based)
Snack Example – API Call to handle single JS Object:
https://snack.expo.io/@zaheersani/api-call---handling-single-object
Networking in React Native:
https://reactnative.dev/docs/network

Lab 14
Firebase

Objective:

The objective of this lab will learn the students about the backend development of react native and expo cli application usin firebase

Activity Outcomes:

The activities provide hands - on practice with the following topics

- Working with Firebase
- Implementation of backend script to connec with react native application

Instructor Note:

As a pre-lab activity, read the documentation at https://rnfirebase.io/

1) Useful Concepts:

React Native Firebase is the officially recommended collection of packages that brings React Native support for all Firebase services on both Android and iOS apps.

Starting from version 10.0.0, React Native Firebase packages share a single common version, with aggregated release notes available:

2) Solved Lab Activites

Sr. No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	***45 mins***	***High***	***CLO5***

Activity 1:

Develop a react native application to make a connection with firebase (real time) database. After connection, insert and retrieve the data from the database.

Solution:

Go to Firebsae, create a project => Go to Realtime database=>copy the API key url from there.

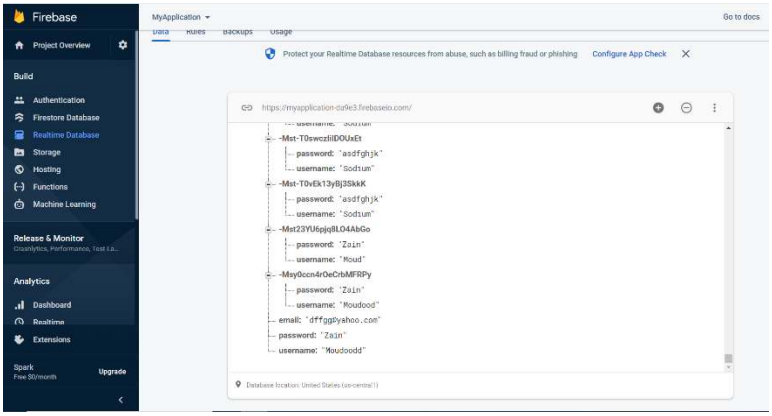
```
import React,{useState} from 'react';
import { Text, View, StyleSheet, Button } from 'react-native';


export default function App() {
```

```
  const [data, setData] = useState("")

const getData = ()=> {
  var requestOptions = {
  method: 'GET'
 //change the above method to 'POST' for inserting the data
// uncomment the below code as well.
// body:JSON.stringify(
  //{
  //  username:"Moudood",
   // password:"Zain"
  //})


};

fetch("https://myapplication-
da9e3.firebaseio.com/users.json", requestOptions)
  .then(response => response.json())
  .then(result =>
  {
    console.log(result)
    setData(result.username)
  })
  .catch(error => console.log('error', error));
}

  return (
    <View style={styles.container}>
      <Text style={styles.paragraph}>
        Hello {data}
      </Text>

      <Button title="Press me" onPress={getData}>
      </Button>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
```

```
  },
  paragraph: {
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
  },
});
```

Output

Data inside the Realtime Firebase.



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Recall TODO App which was developed in the course. Implement it using Firebase Realtime Database.

1. Insert New Item

Instead of insertion locally at memory level, Insert a TODO item into the Firebase.

2. Update Item

Update the item on Firebase.

3. Delete Item

Remove item from the Firebase.

4. Listing

Display list of all TODO Items from the Firebase

5. Implement Pull Down to Refresh

Make the list of TODO Item as pull down to refresh list, when user pulls it down, data should be retrieved from Firebase and updates the list.

References:

Snack Example - Firebase Realtime Database:

https://snack.expo.io/@zaheersani/firebase-demo