12-2018

ORGANIZE EVENTS MOBILE APPLICATION

Thakshak Mani Chandra Reddy Gudimetla
005647976@coyote.csusb.edu

ORGANIZE EVENTS MOBILE APPLICATION

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Thakshak Mani Chandra Reddy Gudimetla

December 2018

ORGANIZE EVENTS MOBILE APPLICATION

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Thakshak Mani Chandra Reddy Gudimetla

December 2018

Approved by:

Dr. David Turner, Advisor, Computer Science and Engineering

Dr. Josephine Mendoza, Committee Member

Dr. Yunfei Hou, Committee Member

ABSTRACT

In a big organization there are many events organized every day. To know about the events, we typically need to check an events page, rely on flyers or on distributed pamphlets or through word of mouth. To register for an event a user now a days typically does this online which involves inputting user details. At the event, the user either signs a sheet of paper or enters credentials in a web page loaded on a tablet or other electronic device. Typically, this is a time-consuming process with many redundancies like entering user details every time the user wants to register for a new event and re-entering the details at the event. This project designs a system that eliminates these redundancies and improves event management.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my mentor, supporter and advisor Dr. David Turner for encouraging and guiding me for completing this project. I would also like to thank my committee members Dr. Josephine Mendoza and Dr. Yunfei Hou for their valuable suggestions and support.

I would also like to thank my parents, Mr. Yogendra Gudimetla and Mrs. Chandra Kala Gudimetla and my sister, Venneesha Gudimetla who stood by my side in every situation and supported me mentally and financially.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Background

The two major smartphone platforms used worldwide are Android and iOS. So, if we want to develop a smartphone application which is intended to be used by most people we at least need to develop it in Android and iOS platforms. It will be very difficult to develop the applications in each platform separately. So, we need a cross platform application development framework. React Native is chosen as it reduces the pain of developing the applications for each platform independently. We can write our code in JSX and react.js to create applications for our required platform.

Purpose

The purpose of this project is to develop a smartphone application which makes the process of announcing, registering, and attending to any events offered by any organization events faster and easier.

Existing System

This project uses CSUSB events to demonstrate the system. Events on campus are posted by the department organizing the event in their department website. The CSUSB Events website collects the information from these department websites and creates organized and browsable events based on the

event date. When you want to register for an event you need to click on an event which then redirects you to the details on the department page, where you enter your details to register for that event. On the day of the event you need to show your Coyote ID card at the event entrance and enter your details in the webpage on the tablet device or just write them down on an attendance sheet.

In the existing system entering the details for registration and again entering details for attending the event is a redundant process.
Moreover, filling in details before checking-in guests is a very slow process which causes a bottleneck.

CHAPTER TWO

SYSTEM ANALYSIS

Proposed System

This project aims at developing a smartphone application for the event management functions of announcing, registering, and checking-in to public events. The features of this application require proper authentication to access. This application implements the one-click register button to make the registration procedure easy. When attending an event, the front desk will display a bar code generated by the application and the attendee just scans that barcode with the application to attend the event.

System Requirement Specification

Hardware Requirements

- A PC with a minimum of 8GB RAM, 20GB free HDD space and a 2Ghz processor (Android Development Only).
- A Macintosh Computer with a minimum of 8GB RAM, 20GB free HDD space and a 2Ghz processor (Android and iOS Development).
- Android phone with a minimum of 1GB RAM and an iPhone 5 or newer.

Software Requirements

- Firebase Backend
- Frameworks: React Native

- Development: VS Code IDE, Node.js, npm (node package manager), Android SDK, Jest.

Dependencies

Dependencies are the list of frameworks and libraries required to run the application.

Project Dependencies

Project dependencies are the list of frameworks and libraries required to run the production build of the application.

Figure 1. Project Dependencies

React. React.js is a highly efficient and productive Java Script library used for building single page web applications. This was developed at Facebook. Normally DOM (Document Object Model) manipulations are very costly, React reduces the DOM writes by creating a virtual DOM and makes only the minimal

required changes in the DOM using virtual DOM. This is achieved by using a render function that calculates the minimum required changes.

React Native. React Native is a cross platform mobile application development framework. Unlike other cross platform mobile application development frameworks like Ionic, Cordova, etc., which renders the app using the built-in browser functionality (called Hybrid Apps), React Native uses native components to render the app. Since native components are faster, developing apps in React Native is better when compared to Hybrid Apps.

JSC Android. React Native Framework on android uses an old version of JSC (Java Script Core) Java Script Engine so to incorporate the new updated builds into the React Native Framework we use this library as a dependency.

MobX. As the codebase of a React Native app gets larger, we need a state management solution for managing the application state during the lifetime of the app. MobX is the easiest alternative to state management solution available to React.

React Navigation. React Native Navigation library is used for navigation between application pages in a React Native Application. It is completely customizable and extensible. In our application, this library is used to manage navigation between different pages.

React Native Camera. React Native Camera is an all in one native camera library for React Native. This library is used in our application for scanning the QR Code when attending an event.

React Native QR Code. React Native QR Code library is a QR Code generator library which generates a scannable QR Code with a given string. In this application this library is used to generate the event QR Code from the event code.

Firebase. Backend with authentication and Real Time Database. It manages the infrastructure for us which makes building mobile apps faster.

React Native Firebase. React Native Firebase library provides an API to communicate with Firebase backend. This API makes writing code which requires backend communication simple and easy.

```
1  import RNFirebase from 'react-native-firebase'
2
3  const firebase = RNFirebase.initializeApp({
4    apiKey: "[redacted]",
5    authDomain: "csusbevents.firebaseapp.com",
6    databaseURL: "https://csusbevents.firebaseio.com",
7    projectId: "csusbevents",
8    storageBucket: "csusbevents.appspot.com",
9    messagingSenderId: "[redacted]"
10 });
11
12 export default firebase;
```

Figure 2. Firebase Configuration

Moment. Moment.js is a date and time library used in this application for parsing and formatting the event date and time.

Tcomb Form Native. Tcomb Form Native is a library for creating forms for getting data from the user. This library features built in form validation and smart rendering capabilities. My application uses this for getting the event information entered by the moderator when creating a new event.

Project Development Dependencies

Project development dependencies are the list of frameworks and libraries required to run the development build of the application.

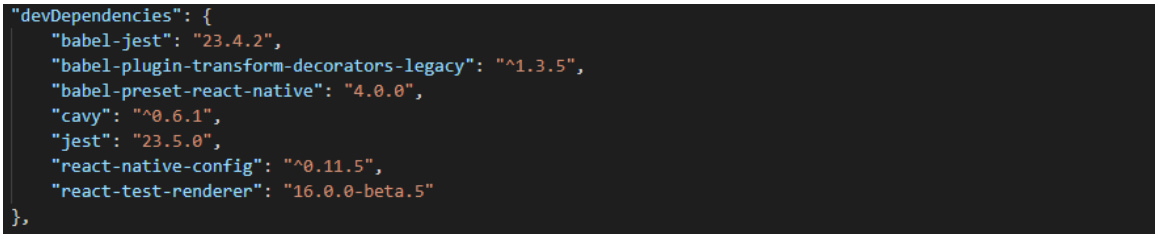


Figure 3. Project Development Dependencies

Babel Preset React Native. Some developers use the latest syntax in Java Script when writing code for the application but if the framework uses an old version of the language and does not know the new syntax it throws an error. The developer must spend a lot of time converting the syntax to be understood by the old engine. Babel transforms syntax written in any version of Java Script to raw Java Script understandable by most Java Script engines.

Babel Plugin Transform Decorators Legacy. An individual objects functionality can be extended using decorators. Decorators are written right before a class, function or variable declaration. Decorators usually start with an '@' symbol followed by the decorator name. ES7 Decorators are not supported by default in React Native. So, we need this plugin to transform decorators. MobX works by using decorators.

React Test Renderer. React test renderer converts react components to pure Java Script objects. These are mainly useful in creating a snapshot of the component which is used in unit testing applications.

Jest. Jest is a highly productive unit testing framework for testing JavaScript applications. Jest features snapshot testing which checks the rendering of the component by taking a snapshot and a JSON tree produced directly from the component code.

Babel Jest. A plugin to run tests with Jest and provides support to ES6 and ES7 syntax.

CHAPTER THREE

SYSTEM DESIGN

The systematic approach chosen before the actual system is developed in the process of software development is called system design. The components, modules, architecture, and the data flowing through this system are defined in this stage.

Scenarios

Login

1. User launches the CSUSB Events Application.
2. Application displays a login page to the user.
3. User enters Coyote ID and password and clicks the login button.
4. Application sends the login information to the server for authentication.
5. Server authenticates the user and sends the response to the application.
6. Application shows the homepage of the authenticated user.

View All Events

1. User is successfully authenticated.
2. Application shows the user homepage.
3. User clicks on the “All Events” button.
4. Application displays all the events to the user.

View My Events

1. User is successfully authenticated.

2. Application shows the user homepage.
3. User clicks on the “My Events” button.
4. Application filters the events in which the user is involved and displays the filtered events to the user. Involved means that the user is either creator of the event or registered for the event.

Register/Un-Register

1. User clicks on an event.
2. Application shows the event details and a button to register/ un-register.
3. User clicks on the register/un-register button.
4. Application updates the event data by adding/removing the user from list of attendees.

Attend Event

1. User clicks on an event.
2. Application displays event details and an “Attend” button if the user is registered to the event.
3. User clicks on the “Attend Button”.
4. Application opens a bar code scanner for scanning the event QR code.
5. User scans the event QR code displayed at the event.
6. Application checks the scanned code with the event code.
7. Application updates the event attendance list on the server with the user id and displays scan success message.

Add Event

1. Moderator clicks on “Add Event” button.
2. Application displays a form for entering the event details.
3. Moderator enters all the details in the form.
4. Moderator clicks on “Create Event” button.
5. Application updates the server with the new event.

Receive Attendee

1. Moderator clicks on an event to receive attendees.
2. Application generates a QR code from the event code.
3. Application displays the generated QR code for the attendee to scan.

Logout

1. User clicks on Logout button.
2. Application sends logout signal to server and waits for response.
3. Server ends user session and returns status to application.
4. Application clears user data and shows the login page.

Data Flow Diagram

In a process or a system, the flow of data or information is mapped with the help of a Data Flow Diagram.

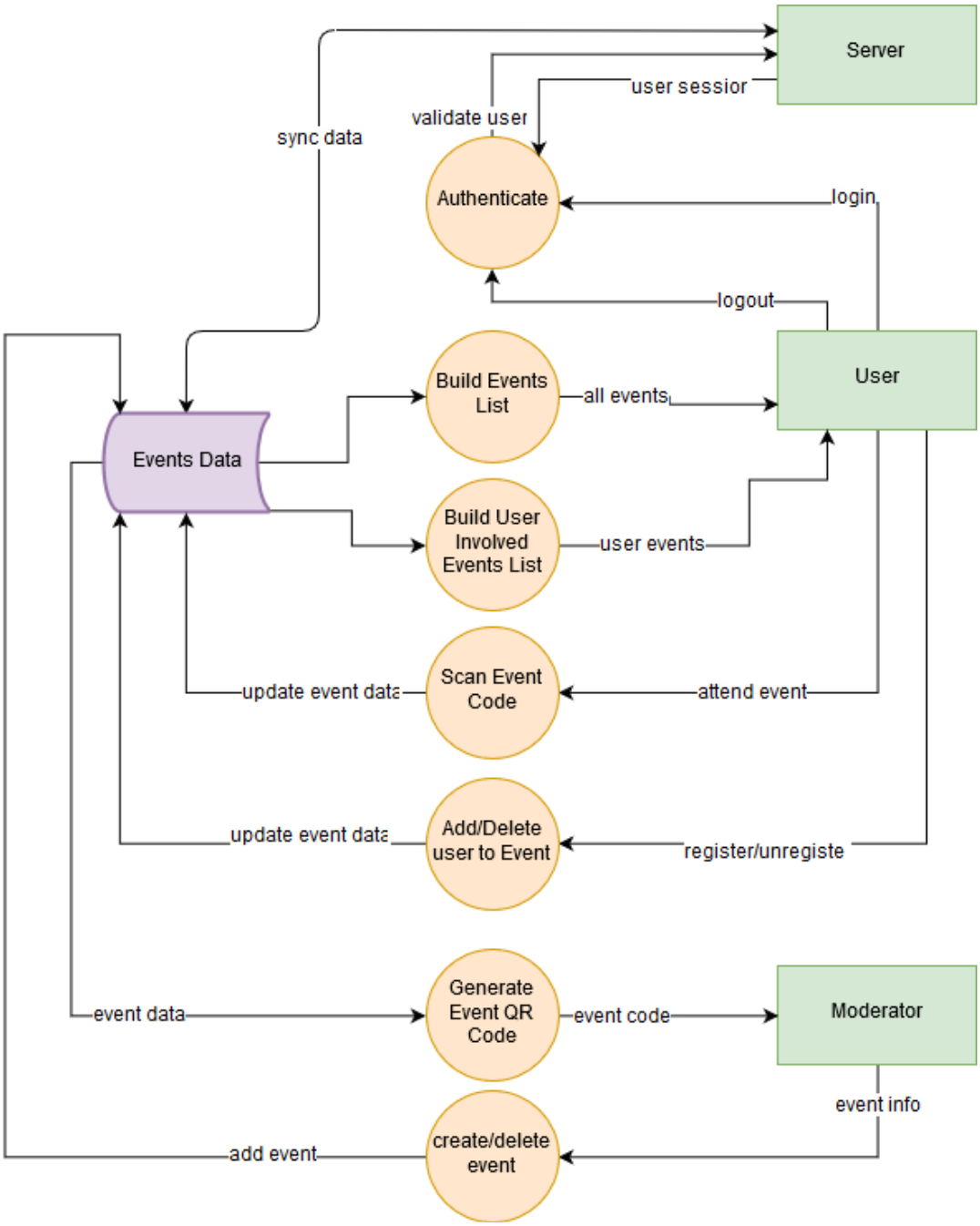


Figure 4. Data Flow Diagram

Use Case Diagram

The elements in a system interact to get a job done in a system. These interactions can be depicted using a Use Case Diagram.

Figure 5. Use Case Diagram

Sequence Diagrams

Sequence Diagrams depict the interactions among various objects in the system in time sequences.

Login

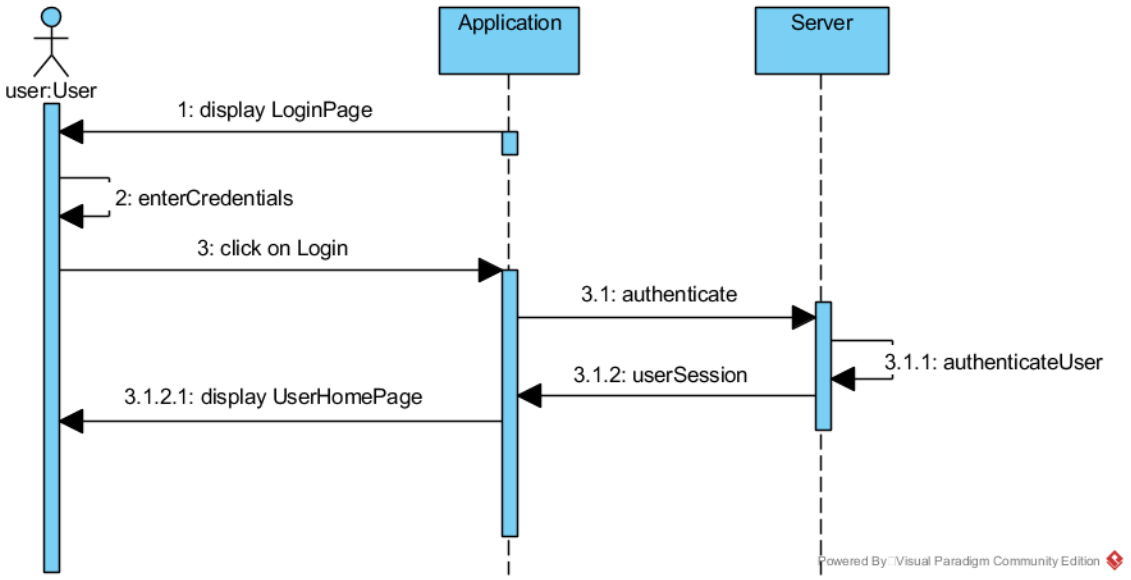


Figure 6. Sequence Diagram – Login

Logout

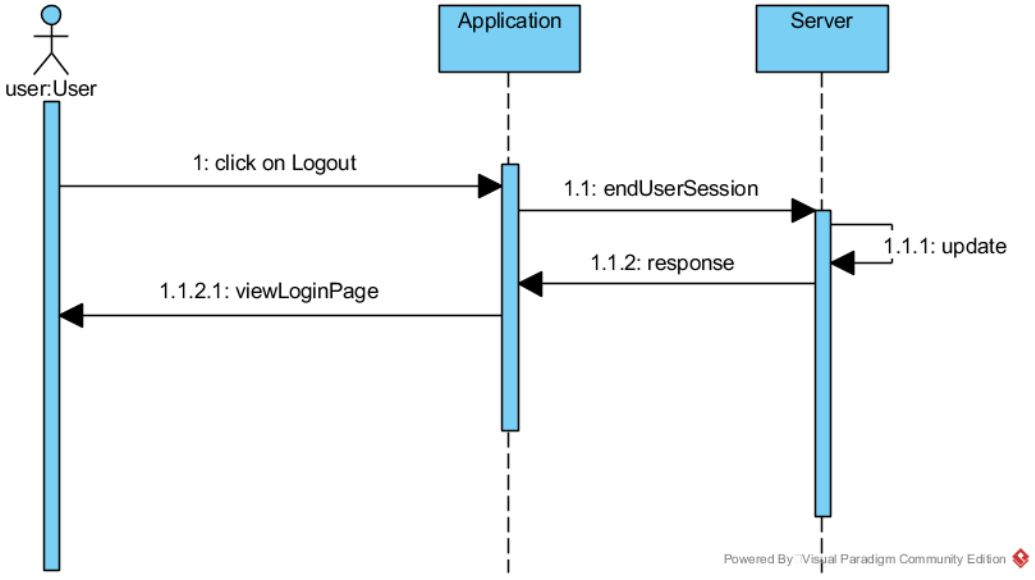


Figure 7. Sequence Diagram – Logout

View All Events

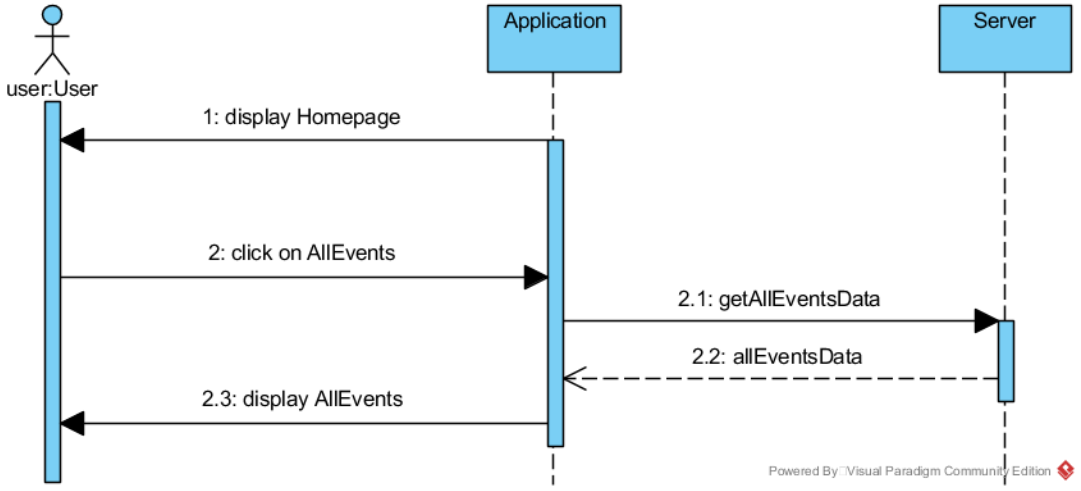


Figure 8. Sequence Diagram – View All Events

View My Events

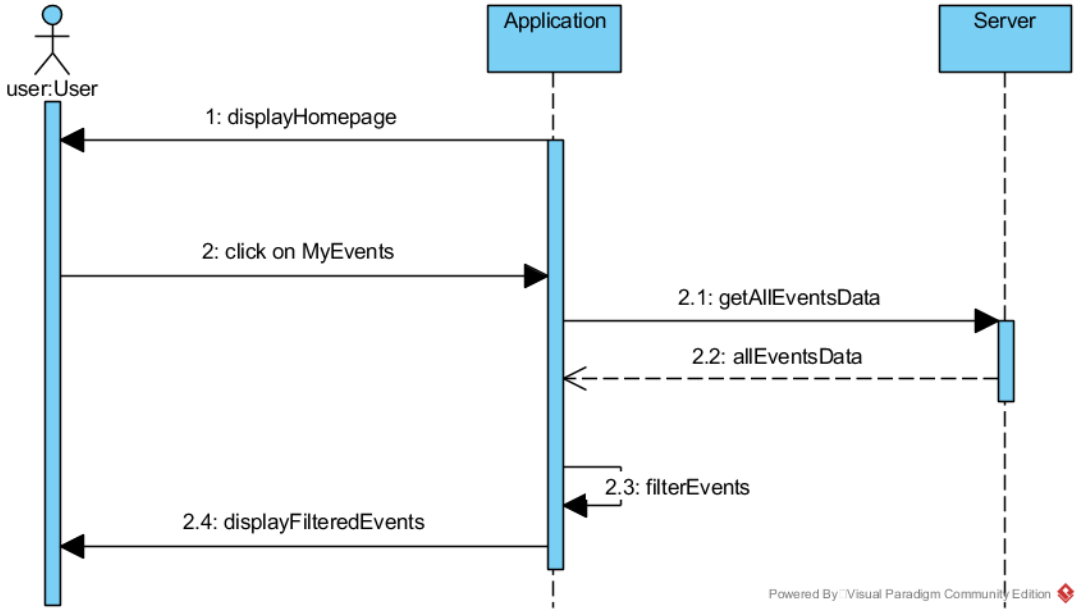


Figure 9. Sequence Diagram – View My Events

Add Event

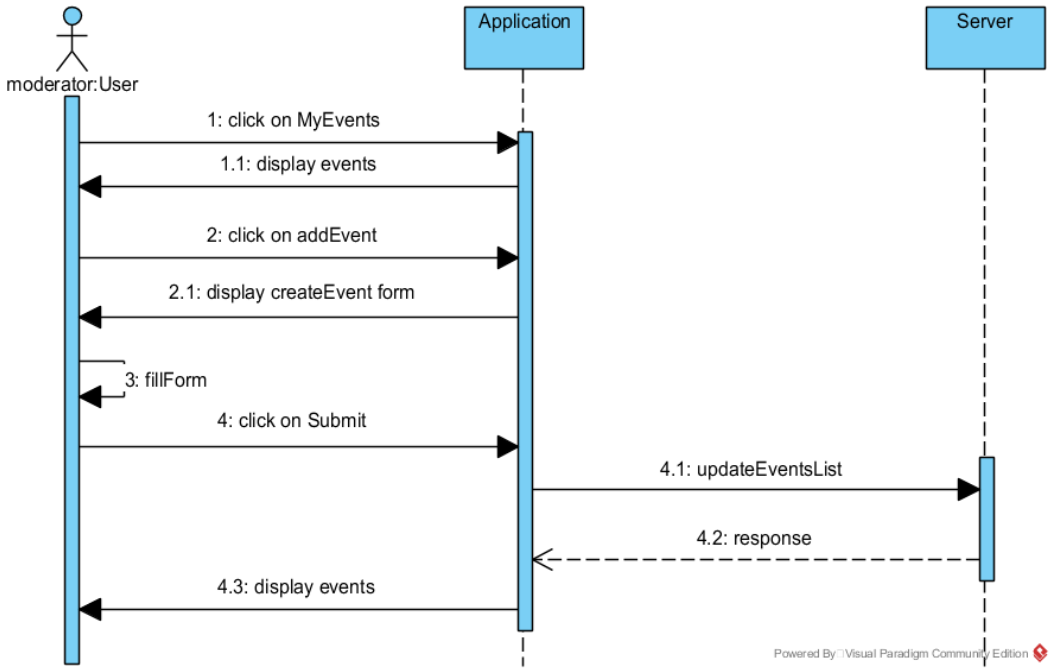


Figure 10. Sequence Diagram – Add Event

Register/Unregister

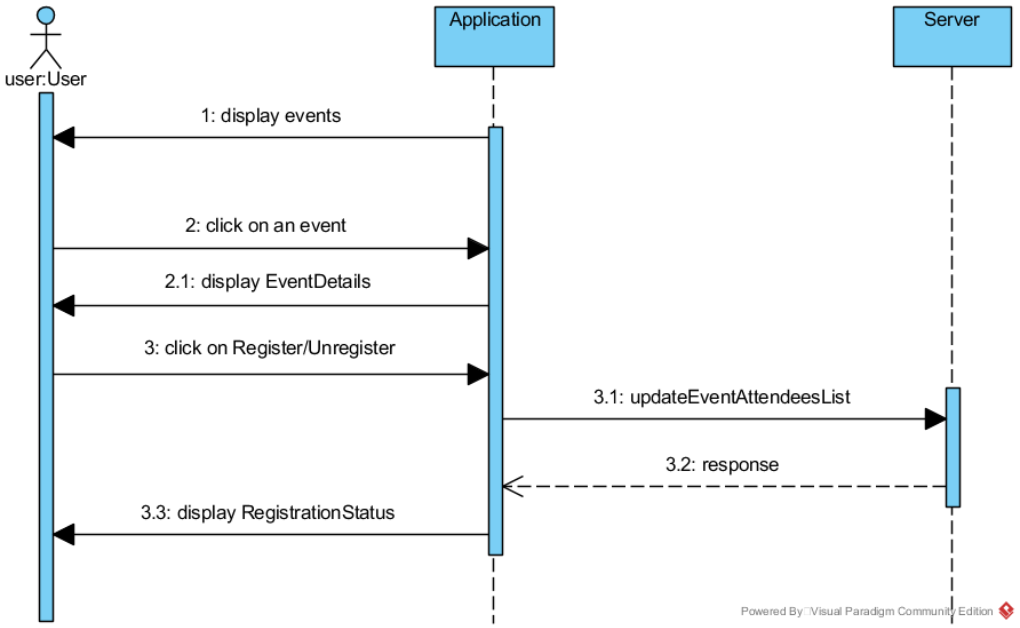


Figure 11. Sequence Diagram – Register/Unregister

Attend Event

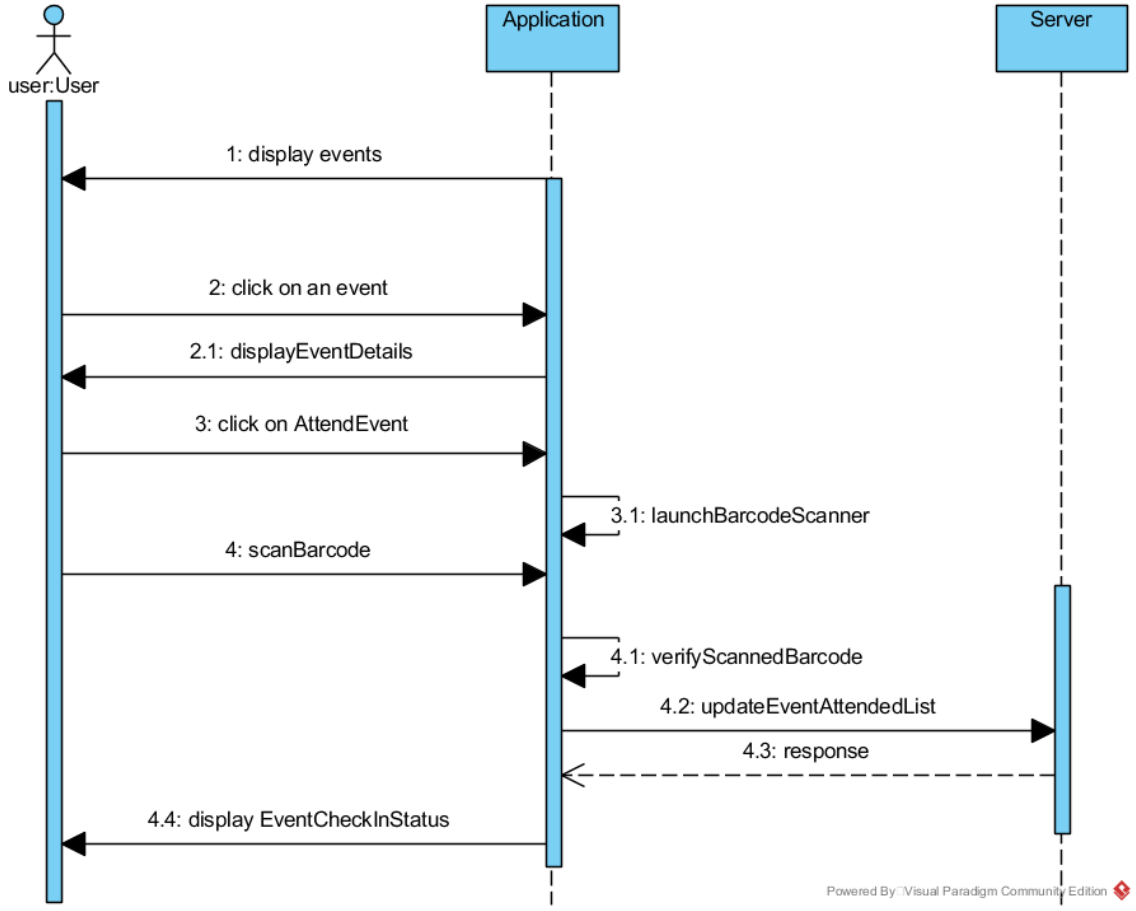


Figure 12. Sequence Diagram – Attend Event

Receive Attendees

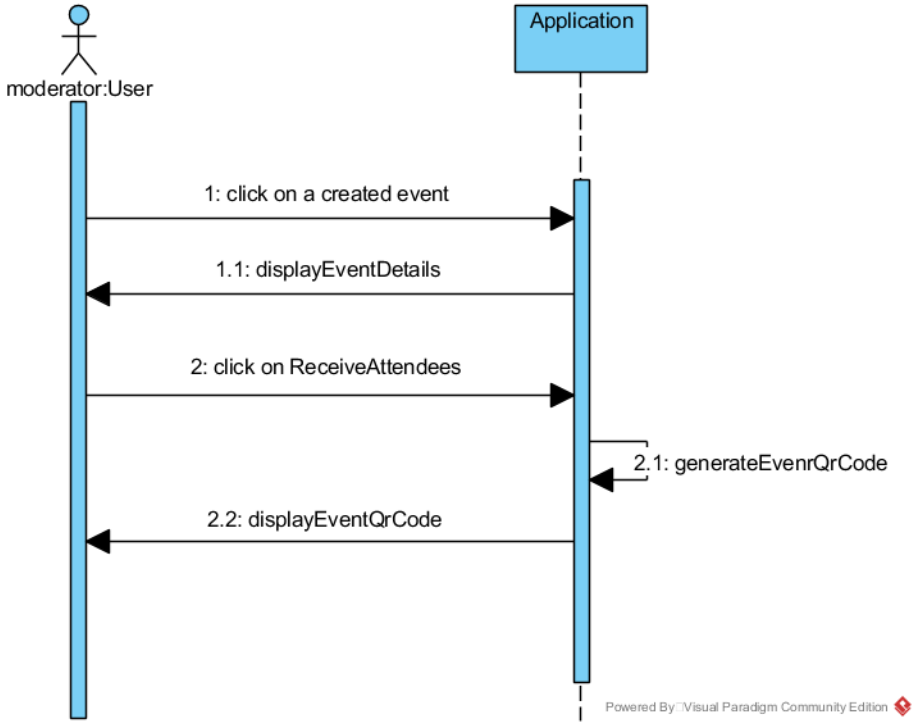


Figure 13. Sequence Diagram – Receive Attendees

Identified Custom Components

AppButton

A custom button component is required because we need the same styling for any text button throughout the application.

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

export default class AppButton extends Component{
    render() {
        return (
            <View style={{ marginTop: 5, margin:10 }}>
                <Button
                title={this.props.title}
                onPress={this.props.onPress}
                />
            </View>
        );
    }
}
```

Figure 14. AppButton Component

ListEvents

We have identified that the purpose of All Events page and My Events page is to list events so a component is required which takes some events and display them.

```
import React, { Component } from 'react';
import { FlatList, StyleSheet, View, Text, TouchableOpacity } from 'react-native';

export default class ListEvents extends Component {
    render() {
      return (
        <View style={styles.container}>
            <FlatList
                data={this.props.eventList.map((item, i) => Object.assign({key:i}, item))}
                renderItem={({item}) =>
                  <TouchableOpacity style={styles.item}
                    onPress={() => {this.props.navigation.navigate('eventDetails', { event: item,});}}>
                    <Text style={styles.eventName}>{item.name.substring(0,32)}{item.name.length<32 ? "" : "..."}</Text>
                    <Text style={styles.eventOrganizer}>{item.organizer}</Text>
                    <Text style={styles.eventDate}>{item.when.date}</Text>

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

Figure 15. ListEvents Component

Navigation

Switch Navigator

Switch Navigators switch between different screens and do not save the information about the switching done. Because of this feature Switch Navigator is used for Login purpose. In our implementation we have a Loading Screen which is displayed when fetching the user data from the server, a Login Screen which is used for logging in to the application and a Stack Navigator.

```
import { createSwitchNavigator } from 'react-navigation'

// import the different screens
import LoadingScreen from './loadingScreen'
import Login from './login'
import RootStack from './appNavigator'

// create our app's navigation
export const RootSwitch = createSwitchNavigator(
  {
    LoadingScreen,
    Login,
    rootStack: RootStack,
  },
  {
    initialRouteName: 'LoadingScreen'
  }
)
```

Figure 16. Switch Navigator

Stack Navigator

After successfully logging in to the application we need a navigator which can store the data about moving between screens and Stack Navigator is the right choice which pushes the switching from one screen to another into a stack which makes the navigation in the application easy.

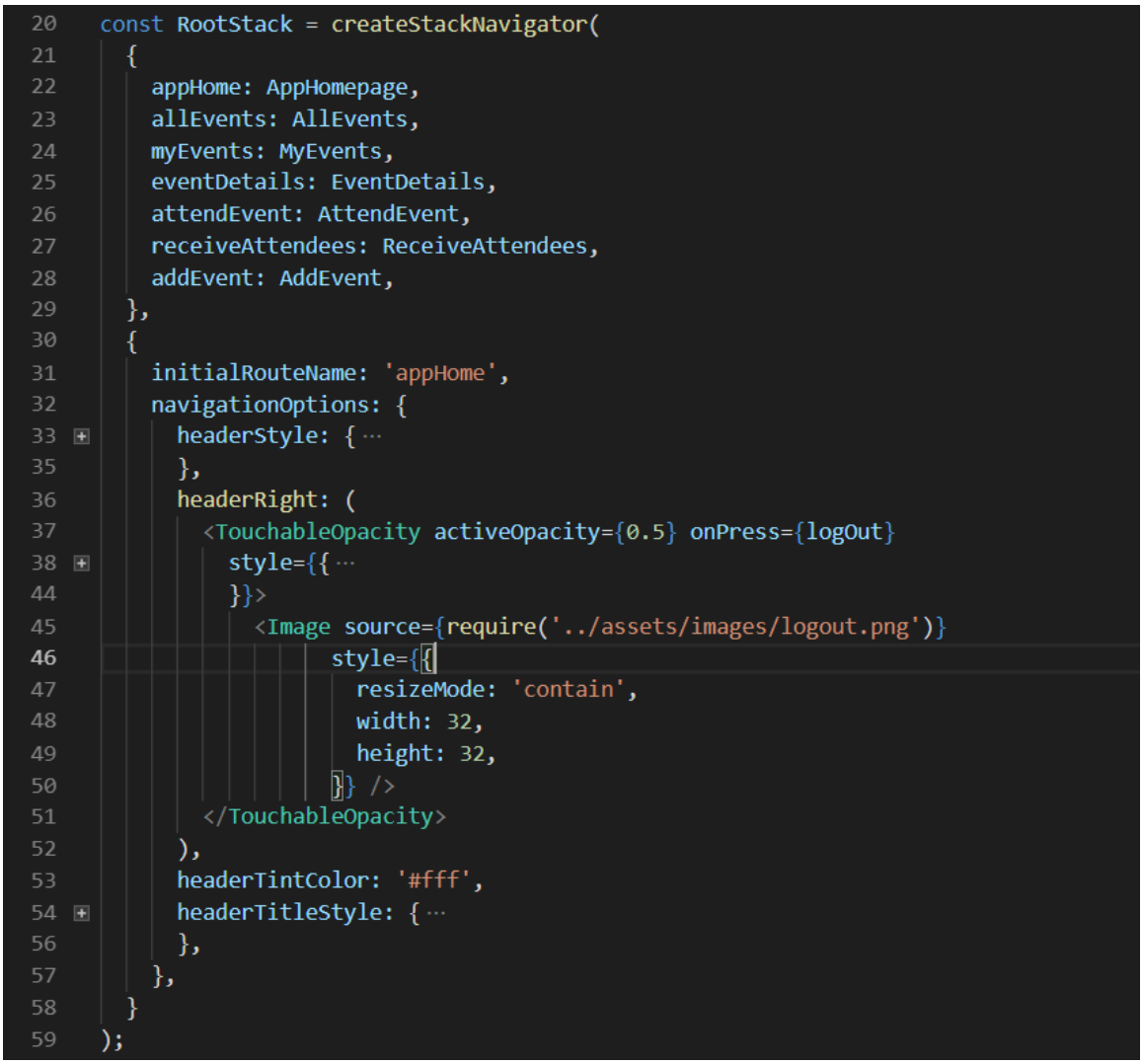


Figure 17. Stack Navigator

Loading

A screen with a spinner in the center to show that the application is loading resources.

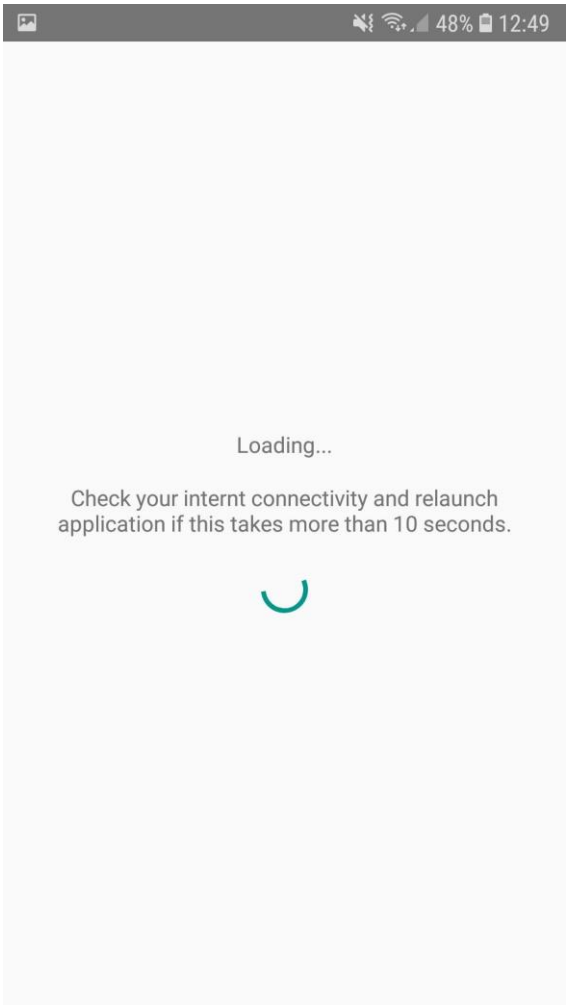


Figure 18. Loading Screen

Login

A screen to login into the application with student Coyote ID and password.

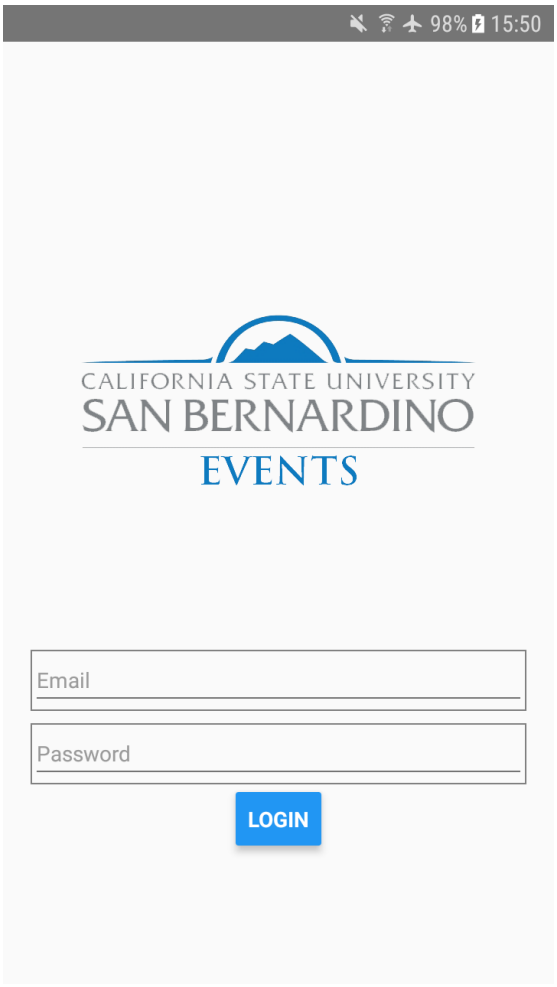


Figure 19. Login Screen

Homepage

In this screen the user is greeted with a welcome note followed by his name and he can navigate to All Events screen or My Events screen by tapping on the relevant button.

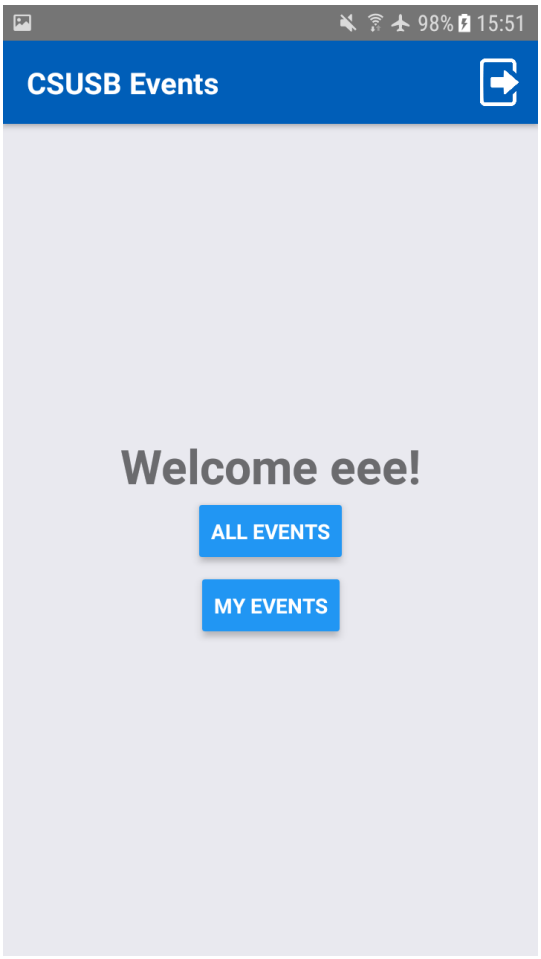


Figure 20. Homepage

All Events

A screen which contains brief information about all the events in the database.

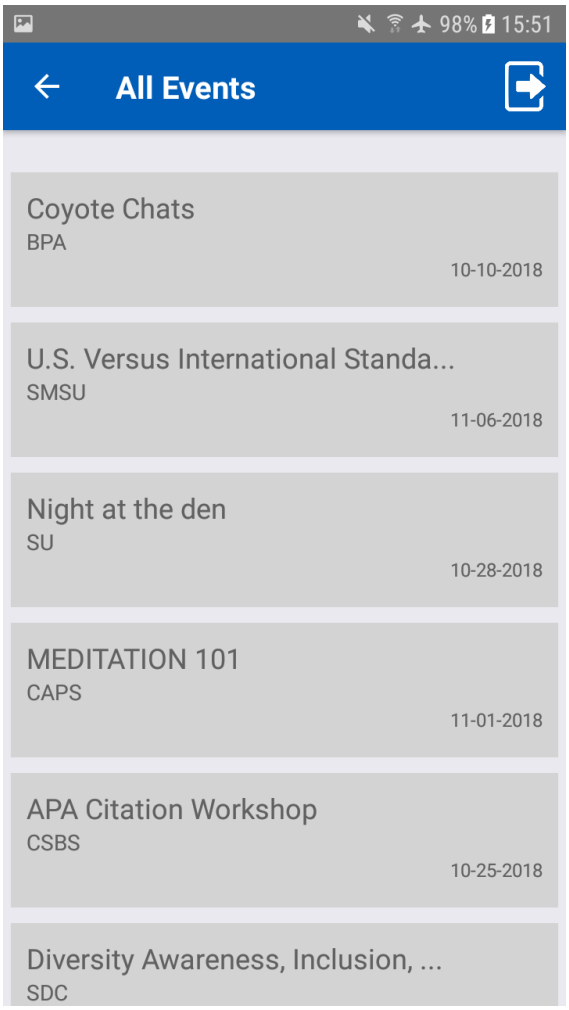


Figure 21. AllEvents

My Events

A screen which contains brief information about the events in which the user is involved. If the user is a moderator a circular floating button is displayed at the bottom right of the page to create a new event.

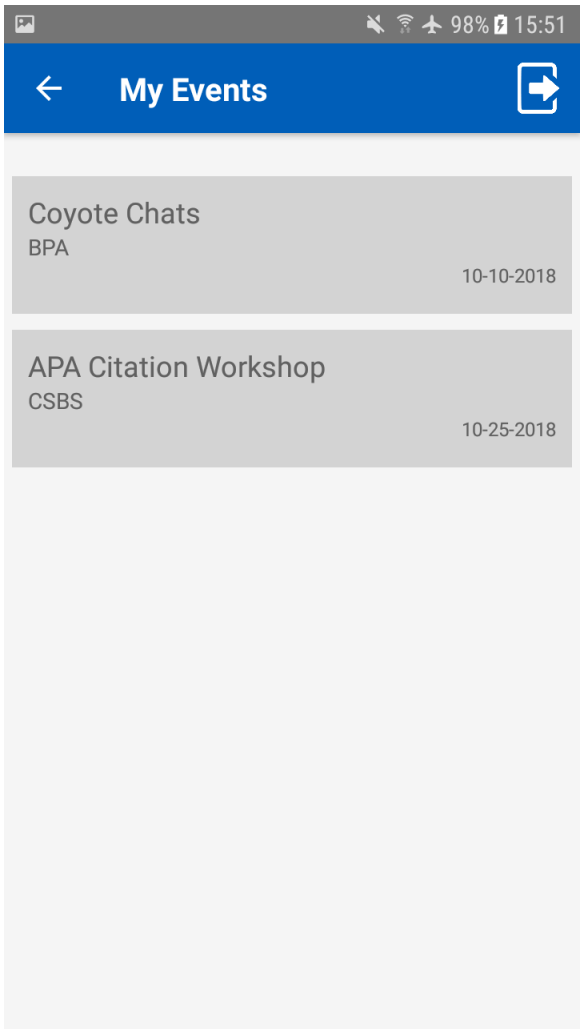


Figure 22. My Events

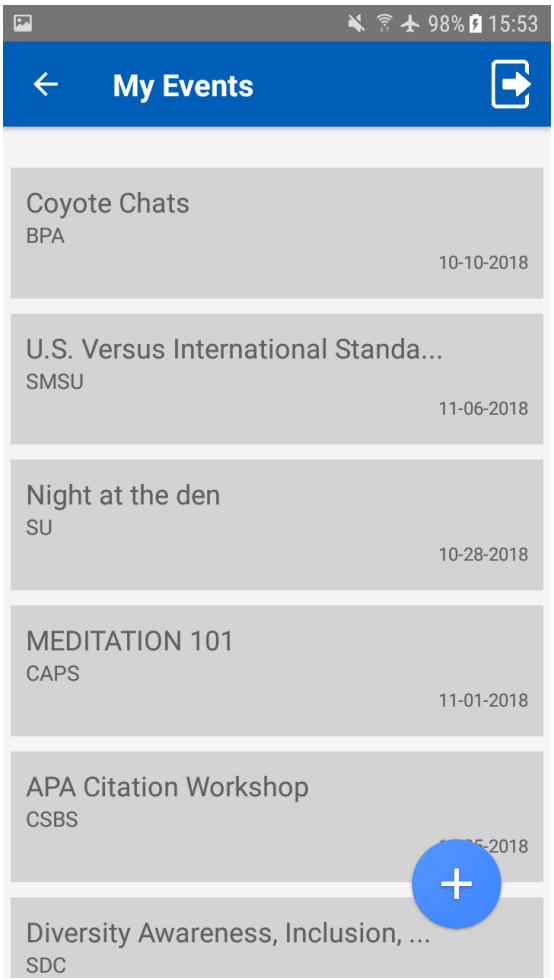


Figure 23. My Events (Moderator)

Event Details

This screen is displayed when the user clicks on an event. This screen contains the details of an event such as name, description, organizer, location, date and time of the event, and buttons to register or unregister and attend event for the attendees. A moderator can see receive attendees button and delete event button.

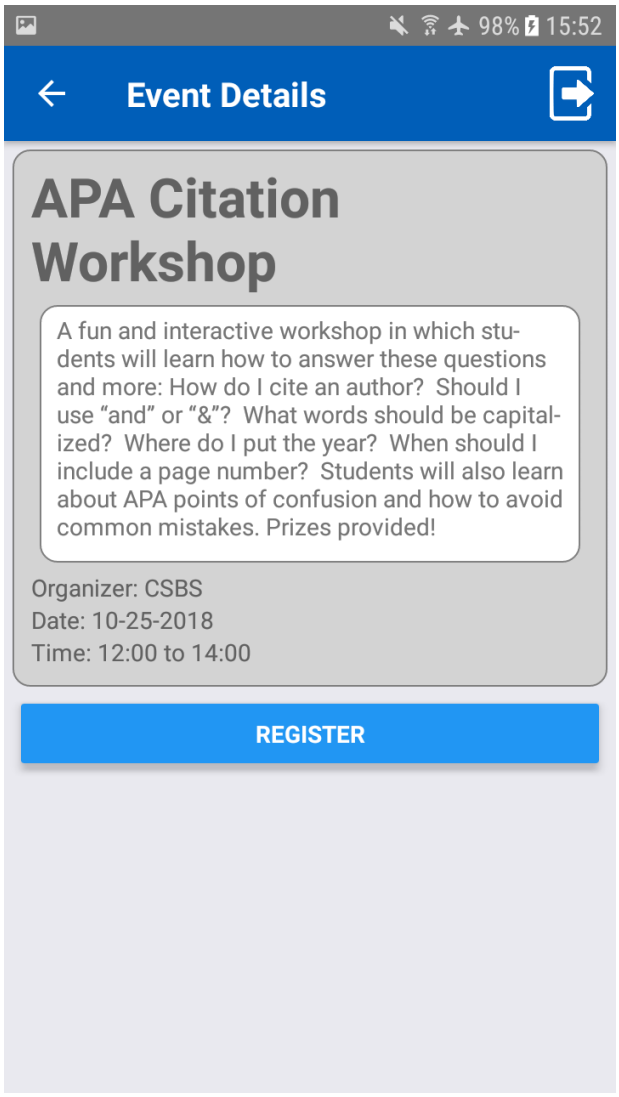


Figure 24. Event Details (Unregistered)

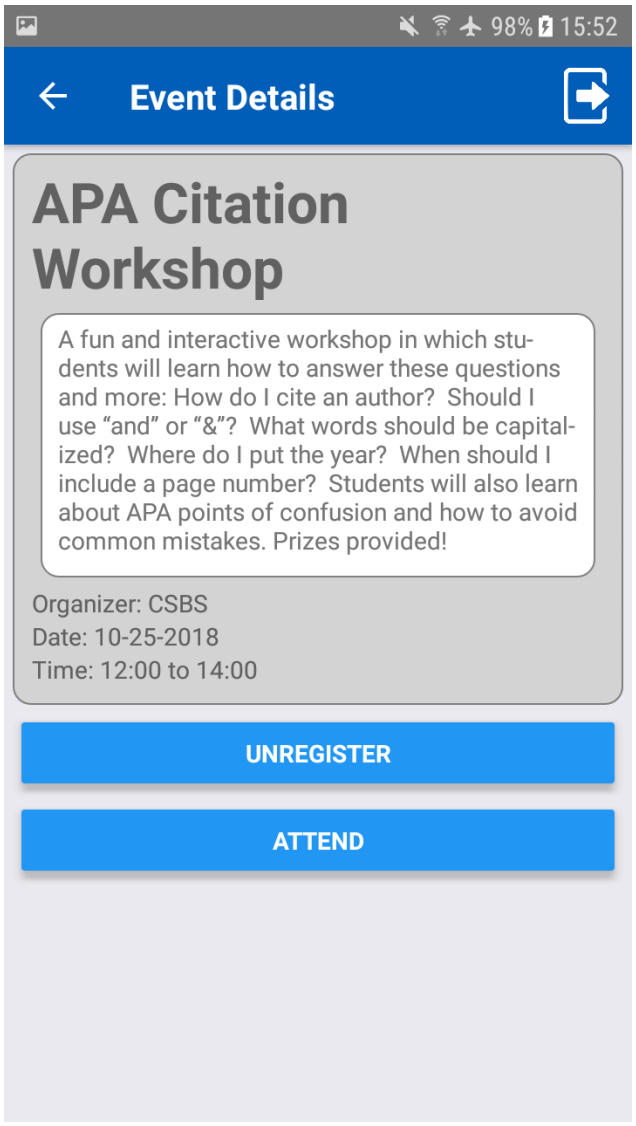


Figure 25. Event Details (Registered)

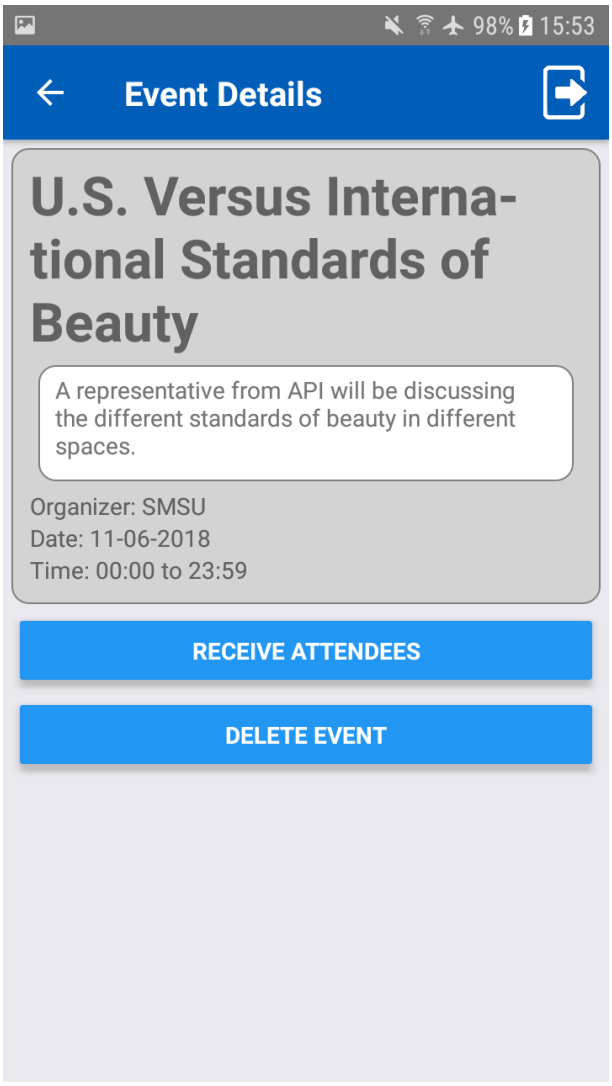


Figure 26. Event Details (Creator)

Create Event

A form to get input from moderator to register a new event. It takes details like event name, organizer, location, description, date and time.

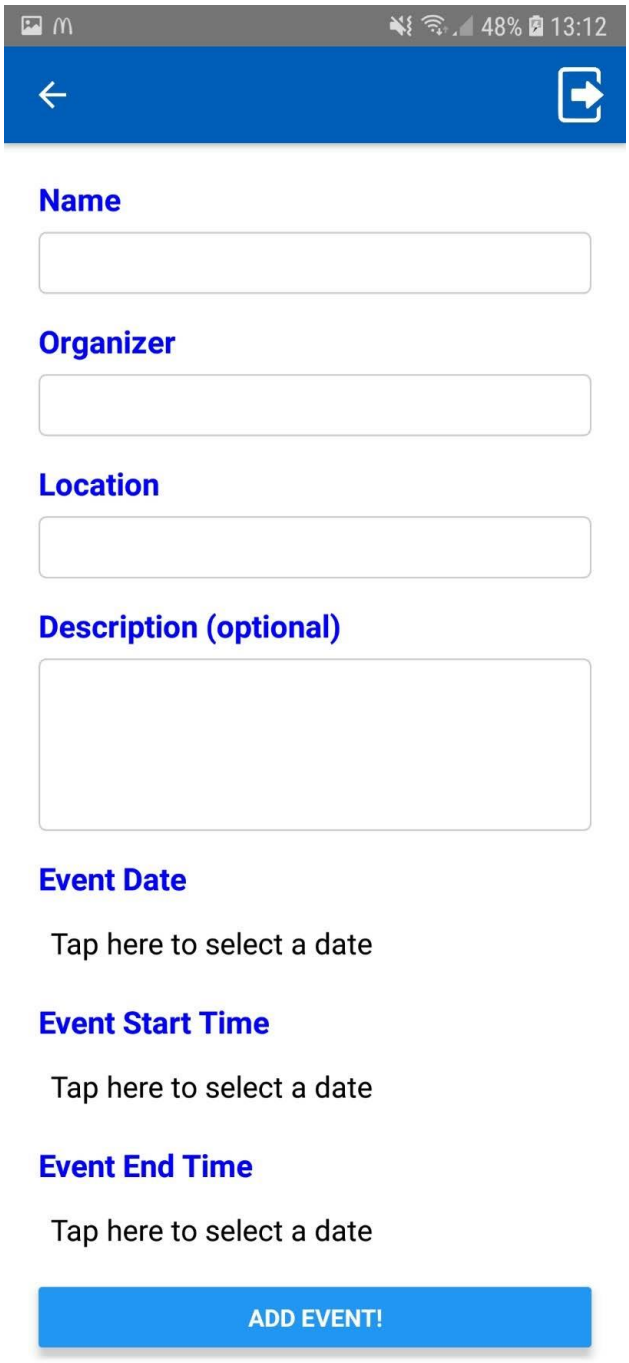


Figure 27. Create Event

Attend Event

A screen which launches a QR code scanner and shows a checked in alert if the correct event barcode is scanned.

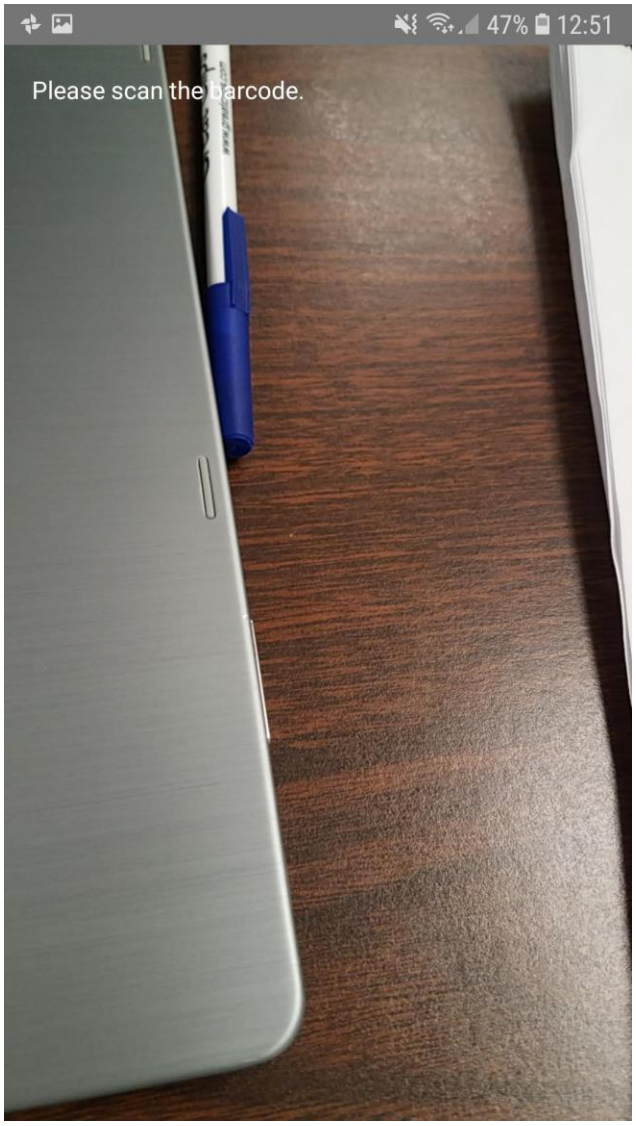


Figure 28. Attend Event

Receive Attendees

A screen which displays a generated QR code from the event code for the attendee to scan.

Figure 29. Receive Attendees

Database Storage

The CSUSB Events application requires that all the information on the users and events to be available in the Firebase Real-Time Database.

Database Hierarchy

Since firebase uses a NoSQL Database, data is stored as JSON. We created a main object “csusbevents” which contains two objects “users” and “events”.

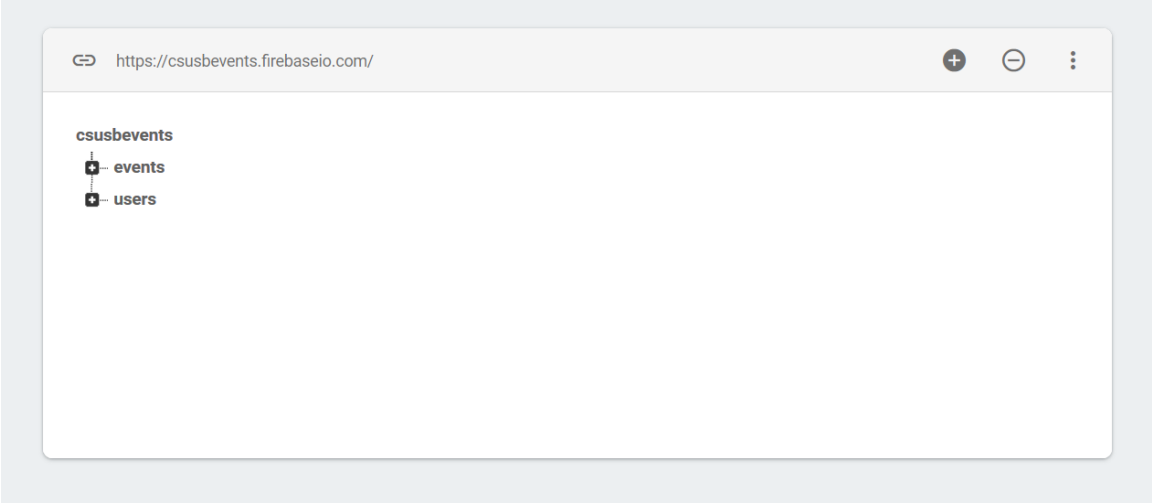


Figure 30. CSUSB Events Database Objects

Users. Users Object store the information on all users. Each user is uniquely identified using their Coyote ID number.

Table 1. User Object

Attribute	Datatype	Description
Name	String	First and last name of the user.
Dob	String (MM-DD-YYYY)	Date of birth of the user.
Major	String	Major of study of the user.
Moderator	Boolean	A flag to identify that the user is a moderator or not. If the value is true the user is a

		moderator and if the value is false he is not a moderator.

Events. Events Object stores the information on all events. Each event is given a unique key generated by firebase.

Table 2. Event Object

Attribute	Datatype	Description
Name	String	Event name.
Code	String	Unique event code.
CreatorId	String	The Id of the event creator.
Desc	String	Description about the event.
Organizer	String	The department organizing the event.
Moderators	Object	Lists all the moderators who can make changes to this event.
When	Object	The date and time of the event.
Where	String	The location where the event takes place.
Attendees	Object	List containing the IDs of all users registered for the event.
Attended	Object	List containing the Ids of all users who attended the event.

Data Store

It is always recommended to store the server communication logic and data requested from the server in one place which can later be accessible by the

entire application. We created a Singleton dbData through which we can communicate with server and make data requests.

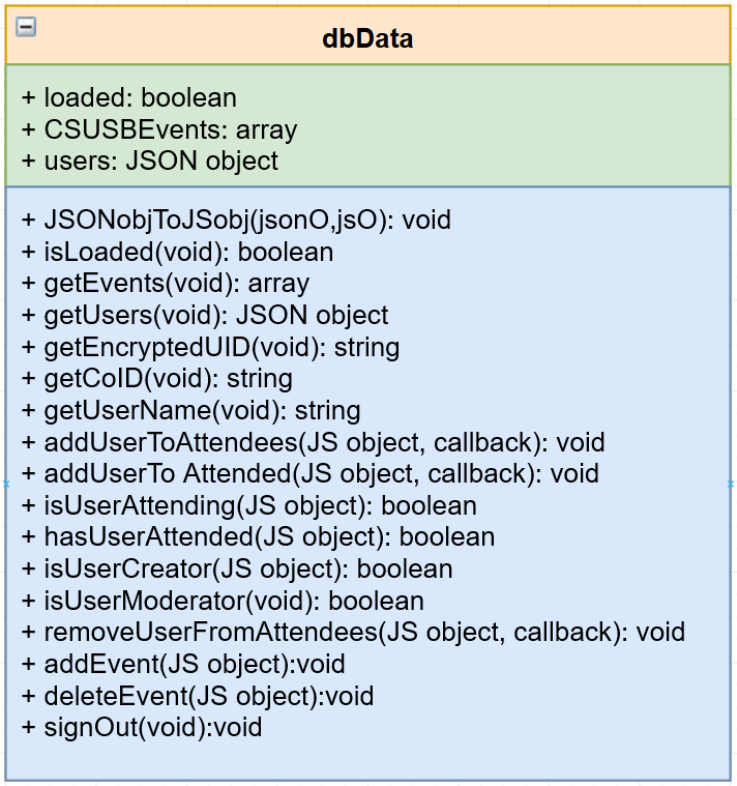


Figure 31. dbData Singleton Class Diagram

CHAPTER FOUR

SYSTEM TESTING

In any software development project testing should be given the same priority as development. Testing ensures bugless, reliable and quality application is produced. A well tested application ensures low maintenance costs in the long run. Testing should be done in all phases of the application development and the methods of testing used in this project are described below.

Unit Testing

Unit testing is the process of testing the smallest unit in the application to determine whether the individual units give the expected output for a specific input. Unit testing the application is done manually and using the Jest Java Script Testing Framework. Jest uses snapshots to test individual units in the application. A snapshot is an expected look of the component when it is rendered. When the test is run a screenshot of the rendered page is taken and compared with the snapshot. This kind of tests ensures that the user interface does not change unexpectedly.

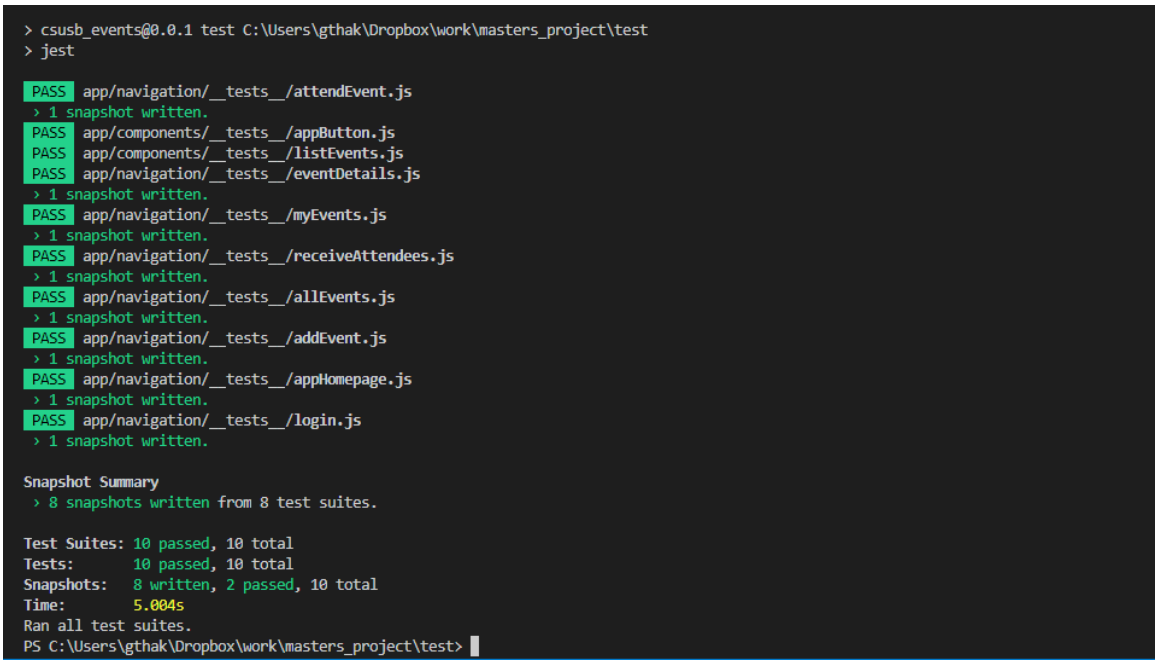


Figure 32. Creating Snapshots

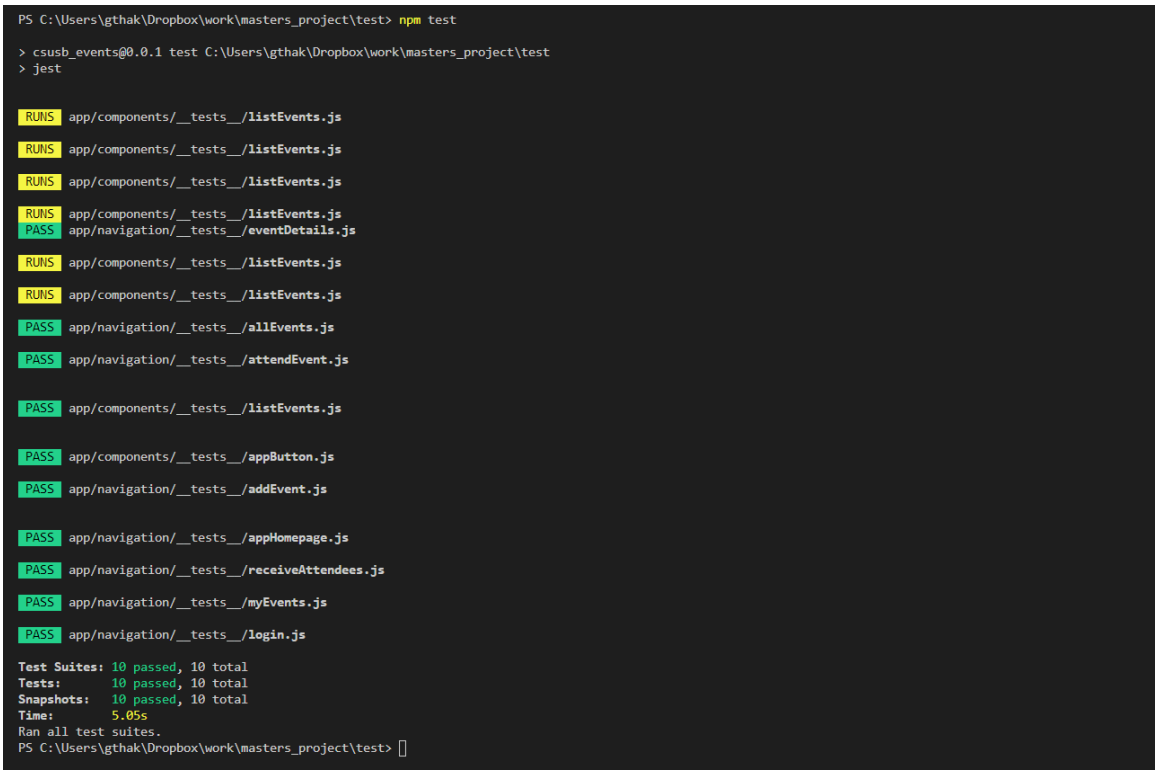


Figure 33. Snapshots Comparison

Integration Testing

Integration Testing tests the application for unexpected behavior when the tested individual units are integrated. Integration tests in the application are done manually. If any unexpected behavior is observed, the bugs are documented and fixed and again tested to ensure the bug fixes will not create new bugs when interacting with other components.

User Acceptance Testing

The developed application will not be released until the user acceptance testing is successful. In this phase of testing some users are randomly selected

and are briefed about the application. An alpha build of the application is installed on few devices for some selected users to test. The users use the application as a normal user and report any unexpected behavior or problems faced during the test usage. The feedback from the users are taken, analyzed and any confirmed bugs or issues they will be resolved and again sent for user testing until the user does not find any issues with the app.

Table 3. User Acceptance Testing – Users and Feedback

User Name	**Admission Status**	**Role**	**User Comment /Feedback**	**Notes**
Harika Alluwala	Female; Conditionally Admitted: Fall 2018	User	**Problem**: Event Description Page won't scroll down when the event description is too long.	**Fixed**: Updated EventDetails component to use ScrollView component instead of View component.
Charitha Chanamolu	Female; Conditionally Admitted: Fall 2016	Moderator	**Problem**: Application won't go beyond the loading page.	**Fixed**: Application was struck because it lost connection with the server as the user's phone lost internet connectivity. This issue was resolved by updating the text on the loading screen.

Testing the application is done with utmost care and ensure that all identified bugs and issues are resolved, and the production build of the application works as intended.

CHAPTER FIVE

FUTURE ENHANCEMENTS

The ideas for extending this project which can make the CSUSB Events application better are listed below.

Locate Event Location on Maps

When creating an event, the moderator can mark the location of the event on the map and the user attending the event can get directions from location of origin to the event location. This make a user new to CSUSB find an event location easily.

Better Method like Near Field Communication to Check-In to the Event

When checking in to the event, with features like Near Field Communication a user can check in to the event with just a tap on the receiver with their phone. This improves the check-in time of the guests to the event because unlike scanning using camera which needs the device camera to be angled parallel pointing to the QR Code which requires adjustment, Near Field Communication is fast because in which angle the devices may be check-in takes place with only one tap.

Notifications and Reminders for New and Registered Events

Notifications for new events keeps a user aware of all events going on campus. Whenever a new event is posted, the user gets a pop-up notification on his phone. Adding reminders for the registered events to remind on the day of the event will ensure that the user does not miss the event.

Share Events with Friends

Event share feature by which the users can share the events to which they are attending with their friends through which the users need not search for the event in the app instead just clicks on a link to see the event information in the app.

CHAPTER SIX

CONCLUSION

This project created software that eliminates unnecessary data input to save time for both event attendees and organizers. Events organized have an easier way of registering and attending. The developed application is tested and free from errors and works with a wide variety of scenarios. This project has the potential of expanding to include features such as sending the event attendance list to event organizers, generating analytics to see which type of events attract which type of users and what times and days of the week are best for these events. The system can also be expanded to provide event suggestions based on user interests.

APPENDIX A

APPLICATION CODE

index.js

```
import React from 'react';
import { AppRegistry } from 'react-native';
import App from './App';

export default AppRegistry.registerComponent('test', () => App);
```

App.js

```
import React, { Component } from 'react';
import { RootSwitch } from './app/navigation/rootSwitch';
import NavigationService from './app/navigation//NavigationService';

export default class App extends Component<{}> {
  render() {
    return (
      <RootSwitch ref={navigatorRef => {
        NavigationService.setTopLevelNavigator(navigatorRef);
      }}/>
    );
  }
}
```

rootSwitch.js

```
import { createSwitchNavigator } from 'react-navigation'

// import the different screens
import LoadingScreen from './loadingScreen'
import Login from './login'
import RootStack from './appNavigator'

// create our app's navigation
export const RootSwitch = createSwitchNavigator(
  {
    LoadingScreen,
    Login,
    rootStack: RootStack,
  },
  {
    initialRouteName: 'LoadingScreen'
  }
);
```

appNavigator.js

```
import { createStackNavigator } from 'react-navigation';
import AppHomepage from './appHomepage'
import AllEvents from './allEvents'
import MyEvents from './myEvents'
import EventDetails from './eventDetails'
import AttendEvent from './attendEvent'
import ReceiveAttendees from './receiveAttendees'
import AddEvent from './addEvent.js'
import React from 'react';
import { Image, TouchableOpacity } from 'react-native';
import dbData from '../store/dataStore'
import NavigationService from './NavigationService';

logOut = function()
{
  dbData.signOut();
  NavigationService.navigate('Login');
}

const RootStack = createStackNavigator(
  {
    appHome: AppHomepage,
    allEvents: AllEvents,
    myEvents: MyEvents,
    eventDetails: EventDetails,
    attendEvent: AttendEvent,
    receiveAttendees: ReceiveAttendees,
    addEvent: AddEvent,
  },
  {
    initialRouteName: 'appHome',
    navigationOptions: {
      headerStyle: {
        backgroundColor: '#005eb8',
      },
      headerRight: (
        <TouchableOpacity activeOpacity={0.5} onPress={logOut}
          style={{
            marginRight: 10,
            width: 32,
            height: 32,
            alignItems: 'center',
            justifyContent: 'center',
          }}>
            <Image source={require('../assets/images/logout.png')}
                  style={{

                    resizeMode: 'contain',
                    width: 32,
```

```
                    height: 32,
                  }} />
        </TouchableOpacity>
      ),
      headerTintColor: '#fff',
      headerTitleStyle: {
        fontWeight: 'bold',
      },
    },
  }
);

export default RootStack;
```

appHomepage.js

```
import React, { Component } from 'react';
import { View, Text, ActivityIndicator,StyleSheet} from 'react-
native';
import AppButton from '../components/appButton';
import dbData from '../store/dataStore'
import firebase from '../firebase.config';
import { observer } from 'mobx-react/native';

@observer
class AppHomepage extends Component {
  state = { currentUser: null }
  componentDidMount() {
      const { currentUser } = firebase.auth();
      this.setState({ currentUser });
  }
  static navigationOptions = {
    title: 'CSUSB Events',
  };

  render() {
    const { currentUser } = this.state;
    this.CSUSBEvents = dbData.getEvents();
    return (dbData.isLoaded() ?
      <View style={{ flex: 1, alignItems: 'center', justifyContent:
'center' }}>
        <Text style = {{
          fontSize: 32,
          fontWeight: 'bold',
        }}>
            Welcome {dbData.getUserName()}!
        </Text>
        <AppButton
          title="All Events"
```

```
          onPress={() => {
            this.props.navigation.navigate('allEvents', {
              CSUSBEvents: this.CSUSBEvents,
            });
          }}
        />
        <AppButton
          title="My Events"
          onPress={() => {
            this.props.navigation.navigate('myEvents', {
              CSUSBEvents: this.CSUSBEvents,
            });
          }}
        />

      </View>:
      <View style={styles.container}>
        <Text>Loading...</Text>
        <Text style={styles.txt}>Check your internt connectivity and
          relaunch application if this takes more than 10 seconds.
        </Text>
        <ActivityIndicator size="large" />
      </View>
    );
  }
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  txt: {
    padding: 15,
    textAlign: 'center',
  }
})
export default AppHomepage;
//export default testable('AppHomepage.View.Text')(AppHomepage);
```

allEvents.js

```
import React, { Component } from 'react';
import ListEvents from '../components/listEvents';
import {observer} from 'mobx-react/native';
import dbData from '../store/dataStore'

@observer
class AllEvents extends Component {
```

```
  static navigationOptions = {
    title: 'All Events',
  };

  render() {
    const { navigation } = this.props;
    CSUSBEvents = navigation.getParam('CSUSBEvents', {});
    console.log(dbData.isLoaded());
    return (
        <ListEvents eventList = {CSUSBEvents} navigation = {
navigation }/>
    );
  }
}

export default AllEvents;
```

myEvents.js

```
import React, { Component } from 'react';import { StyleSheet, View,
Image, TouchableOpacity, Alert, Text } from 'react-native';
import ListEvents from '../components/listEvents';
import {observer} from 'mobx-react/native';
import firebase from '../firebase.config';
import dbData from '../store/dataStore'

@observer
class MyEvents extends Component {
  static navigationOptions = {
    title: 'My Events',
  };
  navigateTo=()=>{
    this.props.navigation.navigate('addEvent');
  }
  myEvent=(e)=>{
    if(dbData.isUserAttending(e)!==null || dbData.hasUserAttended(e)
|| dbData.isUserCreator(e))
      return true;
    return false;
  }
  render() {
    const { navigation } = this.props;
    CSUSBEvents = navigation.getParam('CSUSBEvents', {});

    return (
      <View style={styles.MainContainer}>
        <ListEvents eventList = {CSUSBEvents.filter((event) =>
this.myEvent(event))} navigation = { navigation } />
        {dbData.isUserModerator() ?
```

```
          <TouchableOpacity activeOpacity={0.5}
onPress={this.navigateTo} style={styles.TouchableOpacityStyle} >
            <Image
source={require('../assets/images/floatingActionButton.png')}
                  style={styles.FloatingButtonStyle} />
          </TouchableOpacity> : null
        }
      </View>
    );
  }
}

const styles = StyleSheet.create({

  MainContainer: {
    flex: 1,
    backgroundColor : '#F5F5F5'
  },

  TouchableOpacityStyle:{

    position: 'absolute',
    width: 64,
    height: 64,
    alignItems: 'center',
    justifyContent: 'center',
    right: 30,
    bottom: 30,
  },

  FloatingButtonStyle: {

    resizeMode: 'contain',
    width: 64,
    height: 64,
  }
});

export default MyEvents;
```

addEvent.js

```
import React, { Component } from 'react';
import { ScrollView, StyleSheet, Button } from 'react-native';
import moment from 'moment';
import t from 'tcomb-form-native'; // 0.6.9
import dbData from '../store/dataStore';

const Form = t.form.Form;
```

```
const User = t.struct({
  name: t.String,
  organizer: t.String,
  location: t.String,
  description: t.maybe(t.String),
  date: t.Date,
  fromTime: t.Date,
  toTime: t.Date
});
```

eventDetails.js

```
import React, { Component } from 'react';
import { StyleSheet, Button, Text, TextInput, ScrollView, View }
from 'react-native';
import AppButton from '../components/appButton';
import dbData from '../store/dataStore'
import Spinner from 'react-native-loading-spinner-overlay';

class EventDetails extends Component{
    constructor(props) {
        super(props);
        this.event = this.props.navigation.getParam('event', {});
        this.state = {
            visible : false
        };
    }
    static navigationOptions = {
        title: 'Event Details',
    };
    stopLoading = ()=>{
        this.setState(pState => {
            return {visible: false}
        });
    }
    render() {
        let eventStatus = dbData.isUserAttending(this.event)!==null;
        return(
            <ScrollView style={{ flex: 1 }}>
                <Spinner visible={this.state.visible}
textContent={"Loading..."} textStyle={{color: '#FFF'}} />
                <View style={styles.container}>
                    <Text
style={styles.eventName}>{this.event.name}</Text>
                    <Text
style={styles.item}>{this.event.desc}</Text>
                    <Text>Organizer: {this.event.organizer}</Text>
                    <Text>Location: {this.event.where}</Text>
```

```
                    <Text>Date: {this.event.when.date}</Text>
                    <Text>Time: {this.event.when.fromTime} to
{this.event.when.toTime}</Text>
                </View>
                {dbData.isUserCreator(this.event) ||
dbData.hasUserAttended(this.event) ? null :
                    <View>
                        <AppButton
                            title={  eventStatus ? "UnRegister" :
"Register"}
                            onPress={() => { eventStatus ?
dbData.removeUserFromAttendees(this.event,this.stopLoading)
                                                    :
dbData.addUserToAttendees(this.event,this.stopLoading);
                                this.setState(pState => { return
{visible: true} });
                            }}
                        />
                        {eventStatus ?
                            <AppButton //active only if user
registerd
                                title="Attend"
                                onPress={() =>
{this.props.navigation.navigate('attendEvent',{event:
this.event});}}
                            /> : null
                        }
                    </View>
                }
                {dbData.isUserCreator(this.event) ?
                    <View>
                        <AppButton //active only on day of event and
user is creator of event
                            title="Receive Attendees"
                            onPress={() =>
{this.props.navigation.navigate('receiveAttendees', { barcode:
this.event.code,});}}
                        />
                        <AppButton
                            title="Delete Event"
                            onPress={() =>
{dbData.deleteEvent(this.event);
this.props.navigation.goBack(null);}}
                        />
                    </View> : null
                }
            </ScrollView>
        );
    }
}
const styles = StyleSheet.create({
```

```
    container: {
        paddingTop: 5,
        margin: 5,
        padding: 10,
        backgroundColor:'lightgray',
        borderRadius:10,
        borderWidth: 1,
        borderColor: 'gray'
    },
    item: {
        paddingTop: 5,
        margin: 5,
        padding: 10,
        backgroundColor:'white',
        borderRadius:10,
        borderWidth: 1,
        borderColor: 'gray'
    },
    eventName: {
      fontSize: 32,
      fontWeight: 'bold',
    },
    eventDate: {
      fontSize: 12,
    },
})
export default EventDetails;
```

receiveAttendees.js

```
import React, { Component } from 'react';
import QRCode from 'react-native-qrcode';
import {Text} from 'react-native';

import {
    StyleSheet,
    View,
    TextInput
} from 'react-native';

export default class ReceiveAttendees extends Component{
    static navigationOptions = {
        title: 'Event Code',
      };
    state = {
        text: 'csusb-events-app',
    };

    render() {
```

```
        return (
            <View style={styles.container}>
            <Text style={{padding:10,fontSize: 18,}}>Scan this code
to Attend</Text>
            <QRCode
                value={this.state.text}
                size={300}
                bgColor='black'
                fgColor='white'/>
            </View>
        );
    };
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: 'white',
        alignItems: 'center',
        justifyContent: 'center'
    },
});
```

appButton.js

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

export default class AppButton extends Component{
    render() {
        return (
            <View style={{ marginTop: 5, margin:10 }}>
                <Button
                title={this.props.title}
                onPress={this.props.onPress}
                />
            </View>
        );
    }
}
```

listEvents.js

```
import React, { Component } from 'react';
import { FlatList, StyleSheet, View, Text, TouchableOpacity } from
'react-native';
```

```
export default class ListEvents extends Component {
    render() {
      return (
        <View style={styles.container}>
            <FlatList
                data={this.props.eventList.map((item, i) =>
Object.assign({key:i}, item))}
                renderItem={({item}) =>
                  <TouchableOpacity style={styles.item}
                    onPress={() =>
{this.props.navigation.navigate('eventDetails', { event: item,});}}>
                    <Text
style={styles.eventName}>{item.name.substring(0,32)}{item.name.lengt
h<32 ? "" : "..."}</Text>
                    <Text
style={styles.eventOrganizer}>{item.organizer}</Text>
                    <Text
style={styles.eventDate}>{item.when.date}</Text>

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

const styles = StyleSheet.create({
    container: {
     flex: 1,
     paddingTop: 22
    },
    item: {
      margin: 5,
      padding: 10,
      height: 86,
      backgroundColor: 'lightgray',
    },
    eventName: {
      fontSize: 18,
    },
    eventDate: {
      alignSelf: 'flex-end',
      fontSize: 12,
    },
})
```

dataStore.js

```
import firebase from '../firebase.config';
import {observable} from 'mobx';
import moment from 'moment';

class dbData{
    @observable loaded = false;
    @observable CSUSBEvents = [];
    users = {};
    eventSnap = {};
    JSONobjToJSobj(jsonO,jsO)
    {
        for (let objKey in jsonO){
            jsO.push(jsonO[objKey]);
        }
    }
    constructor(props){
        firebase.database()
          .ref('/')
          .on('value',function(snapshot) {
            this.CSUSBEvents.length=0;
            let snap = snapshot.val();
            this.users = snap['users'];
            this.eventSnap = snap['events'];
            this.JSONobjToJSobj(this.eventSnap,this.CSUSBEvents);
            this.loaded = true;
          }.bind(this));
    }
    isLoaded()
    {
        console.log(this.loaded);
        return this.loaded;
    }
    getEvents() {
        return this.CSUSBEvents;
    }
    getUsers() {
        return this.users;
    }
    getEncryptedUID()
    {
        return firebase.auth().currentUser.uid;
    }
    getCoID()
    {
        return firebase.auth().currentUser.email.split("@")[0];
    }
    getUserName()
    {
        return this.users[parseInt(this.getCoID())].name;
```

```
    }
    addUserToAttendees(e,exec)
    {
        let newAttendeeKey =
firebase.database().ref().child('events/'+e.code+'/attendees').push(
).key;
        firebase.database()
        .ref('events/'+e.code+'/attendees/' + newAttendeeKey)
        .set(this.getCoID()).then(()=>{exec()});
    }

    addUserToAttended(e,exec)
    {
        let newAttendeeKey =
firebase.database().ref().child('events/'+e.code+'/attended').push()
.key;
        firebase.database()
        .ref('events/'+e.code+'/attended/' + newAttendeeKey)
        .set(this.getCoID()).then(()=>{exec()});
    }

    isUserAttending(e)
    {
        e = this.eventSnap[e.code];
        let cid = this.getCoID();
        for (let objKey in e.attendees){
            if(e.attendees[objKey] == cid)
            {
                return objKey;
            }
        }
        return null;
    }

    hasUserAttended(e)
    {
        e = this.eventSnap[e.code];
        let cid = this.getCoID();
        for (let objKey in e.attended){
            if(e.attended[objKey] == cid)
            {
                return true;
            }
        }
        return false;
    }

    isUserCreator(e)
    {
        let cid = this.getCoID();
        if(e.creatorId == cid)
```

```
        {
            return true;
        }
        return false;
    }

    isUserModerator()
    {
        return this.users[parseInt(this.getCoID())].moderator;
    }

    removeUserFromAttendees(e,exec)
    {
        let stat = this.isUserAttending(e);
        if(stat!=null)
        {

firebase.database().ref().child('events/'+e.code+'/attendees/'+stat)
.remove().then(()=>{exec()});
        }
    }

    addEvent(value)
    {
        let userID =
firebase.auth().currentUser.email.split("@")[0];
        let newEventKey =
firebase.database().ref().child('events').push().key;
        let newEvent = {
            name: value.name,
            organizer: value.organizer,
            when: {
                date:moment(value.date).format('MM-DD-YYYY'),
                fromTime:moment(value.fromTime).format('HH:mm'),
                toTime:moment(value.toTime).format('HH:mm')
            },
            where: value.location,
            desc: value.description,
            creatorId: userID,
            moderators: [userID],
            attendees: [userID],
            attended: [userID],
            code: newEventKey,
        }

        firebase.database()
        .ref('/events/' + newEventKey)
        .set(newEvent);
    }
    deleteEvent(e)
    {
```

```
        firebase.database().ref().child('events/'+e.code).remove();
        console.log('events/'+e.code);
    }
    signOut()
    {
        firebase.auth().signOut().then(function() {
            console.log('Signed Out');
          }, function(error) {
            console.error('Sign Out Error', error);
          });
    }
}

export default new dbData;
```

firebase.config.js

```
import RNFirebase from 'react-native-firebase'
const firebase = RNFirebase.initializeApp({
  apiKey: "xxxxxxxxxxxxxxxxxxxxxxxxxxx",// intentionally removed
  authDomain: "csusbevents.firebaseapp.com",
  databaseURL: "https://csusbevents.firebaseio.com",
  projectId: "csusbevents",
  storageBucket: "csusbevents.appspot.com",
  messagingSenderId: "000000000000" // intentionally removed
});

export default firebase;
```

APPENDIX B

APPLICATION ASSETS

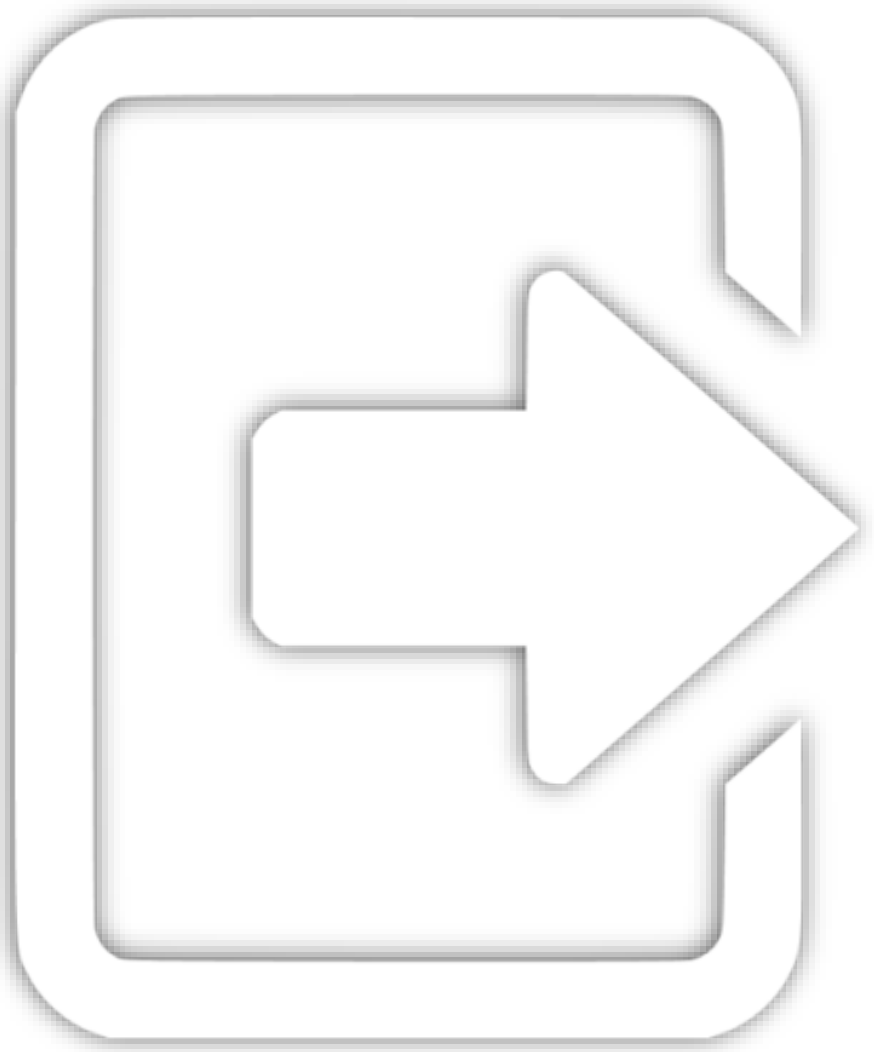
CSUSB Events Application Logo

csusb_LG.png (600x309)

Add Event Button

floatingActionButton.png (512x512)

Logout Button



logout.png (512x512)

REFERENCES

Ahmad, F. A. (2018, April 12). React Native & Firebase: Authentication – React Native Training – Medium. Retrieved from https://medium.com/react-native-training/react-native-firebase-authentication-7652e1d2c8a2

Carli, S. (2017, September 20). Easily Build Forms in React Native – React Native Development – Medium. Retrieved from https://medium.com/react-native-development/easily-build-forms-in-react-native-9006fcd2a73b

Dabit, N. (2016, July 01). React Native with MobX - Getting Started – React Native Training – Medium. Retrieved from https://medium.com/react-native-training/react-native-with-mobx-getting-started-ba7e18d8ff44

Gaare, J. (2017, May 25). Learning to test React Native with Jest - part 1 – React Native Training – Medium. Retrieved from https://medium.com/react-native-training/learning-to-test-react-native-with-jest-part-1-f782c4e30101

JSC build scripts for Android. (2018, June 05). Retrieved from https://github.com/react-community/jsc-android-buildscripts#how-to-use-it-with-my-react-native-app

Jun, K. (2017, August 18). RNCamera: Use the barcode scanner on React Native. Retrieved from https://gist.github.com/goodpic/f1ba553d85f96c76b6b2992faf037d87

Learn the Basics · React Native. (n.d.). Retrieved from https://facebook.github.io/react-native/docs/tutorial.html

Porcello, E. (2018, June 20). Retrieved from https://www.lynda.com/React-js-tutorials/React-js-Essential-Training/496905-2.html

Ten minute introduction to MobX and React. (n.d.). Retrieved from https://mobx.js.org/getting-started.html