CALENDAR -A MOBILE APPLICATION
BASED ON REACT NATIVE FRAMEWORK

Approved: Dr.Songqing Yue Date: 05/01/2017
Paper Advisor

Suggested content descriptor keywords:

React Native

Android/iOS

Calendar

CALENDAR –A MOBILE APPLICATION BASED ON REACT NATIVE FRAMEWORK

A Seminar Paper

Presented to

The Graduate Faculty

University of Wisconsin-Platteville

In Partial Fulfillment

Of the Requirement for the Degree

Master of Science in Computer Science

Department of Computer Science and Software Engineering

College of Engineering, Mathematics and Science

By

Surya Pulithitta Ravindran Nair

2018

ACKNOWLEDGEMENTS

I am indebted to the Almighty for His blessings for the successful completion of this project. I owe my deepest gratitude to my guide Prof. Songqing Yue for his support, patience, guidance and motivation. His immense knowledge and guidance helped me during my project work and writing this final report. I would like to thank all my professors and my family for their continuous and unconditional support during this entire course.

TABLE OF CONTENTS

CHAPTER I INTRODUCTION

As the mobile applications are in the matured state now, there are many frameworks which are being proposed to develop these applications. One such framework is react-native. React Native is a JavaScript framework for writing real, natively rendering iOS and Android applications. It's based on React, Facebook's JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms. React Native is like React, but it uses native components instead of web components as building blocks. React Native gracefully handles multiple platforms. The vast majority of the React Native APIs are cross-platform, so you just need to write one React Native component, and it will work seamlessly on both iOS and Android.

This report documents the architecture of a mobile based calendar application and how it leverages various released packages and support from react-native framework.

CHAPTER II IMPLEMENTATION

i. Implementation Review

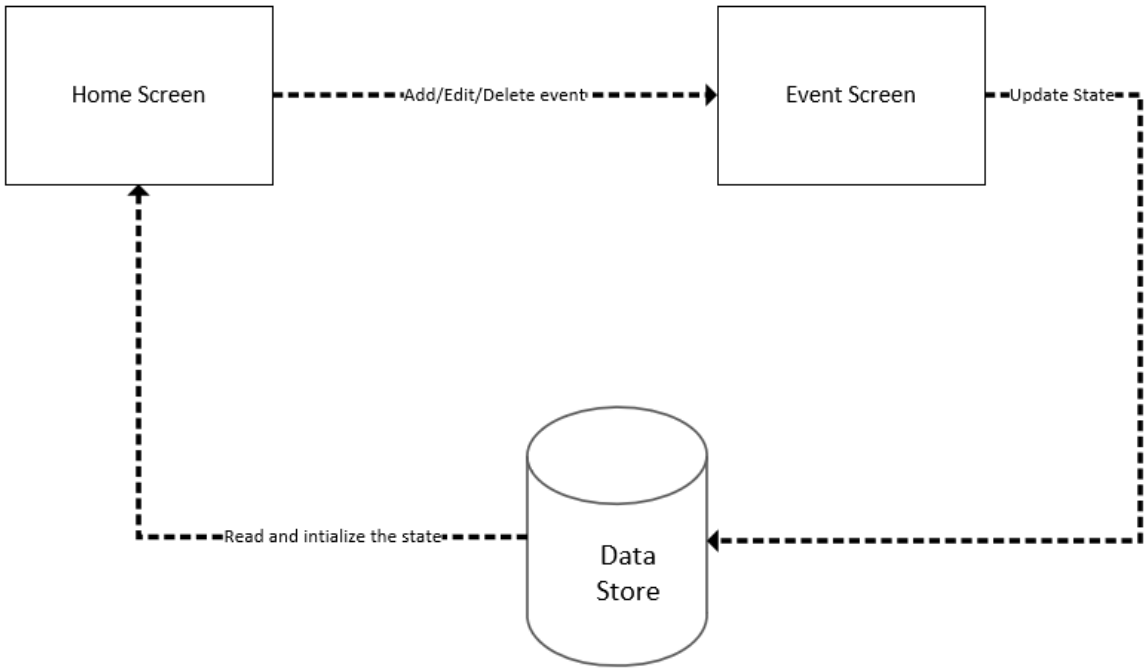
Two screens are used in this project. One is the 'Home Screen' where calendar list is displayed, other is 'Event Screen', where we can add or edit event.

For adding a new event or editing an event that is already saved, we need to select a date and click on the action-button. By pressing action-button we can navigate to Add/Edit Event Page or 'Event Screen'. In this page there is option to enter values of name ,date, time from and time to of the corresponding event and save the event by clicking 'save' button. This information about the event is converted into a string and stored in JSON file by assigning it a unique key. For this react-native-simple-store wrapper is used and it in turn uses React Native's AsyncStorage component.

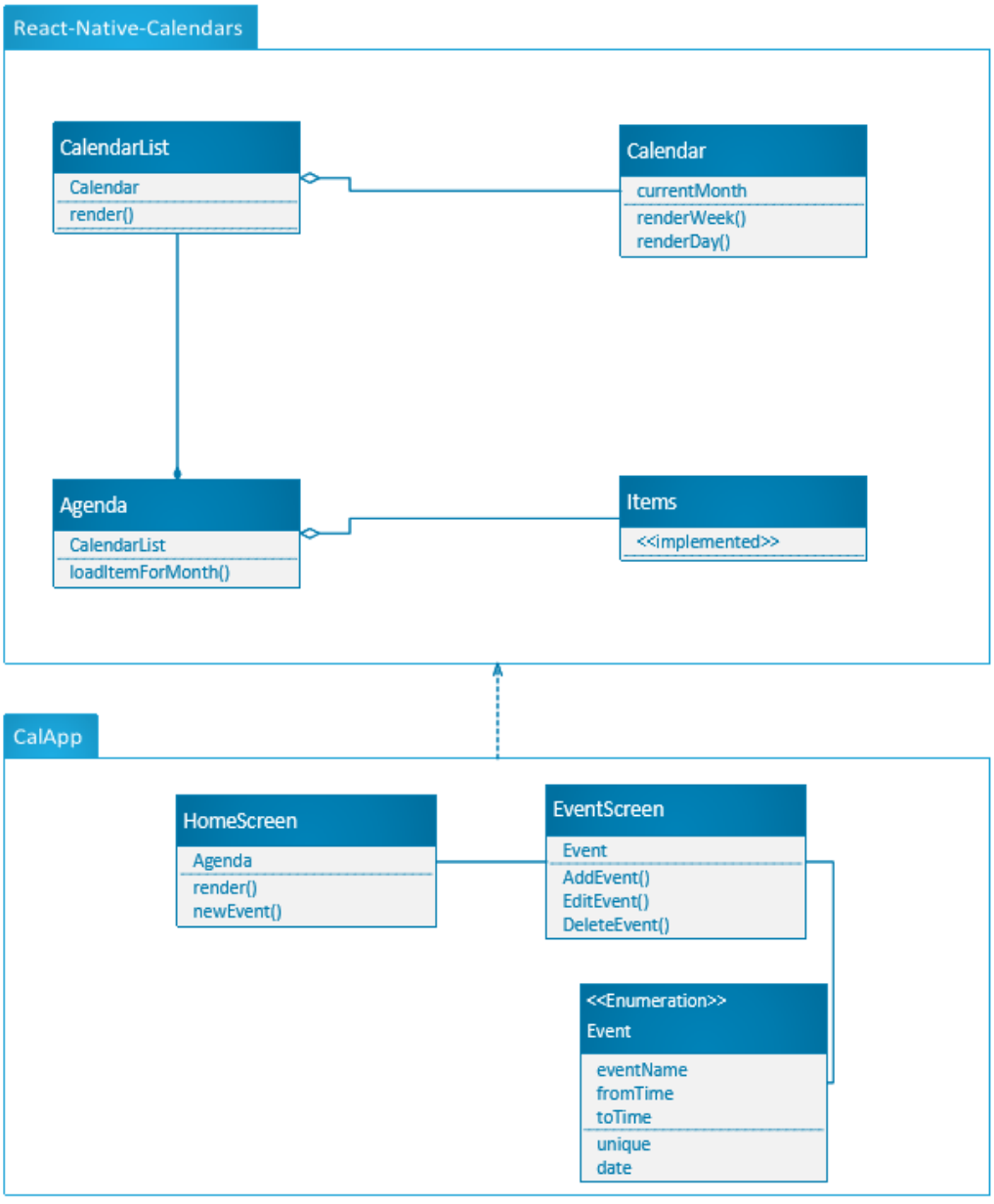
For viewing an event stored on a particular date, just click on the corresponding date from 'Home Screen' and it will navigate to a page in the Agenda Component. Agenda is a component used in react- native-calendars interface. Here we list all the events on that day after retrieving the data from JSON file. In the JSON events are stored as items in the following format,
Items = {'2017-05-22': [{ event string 1},{event string2}] }.

For deleting an event, navigates to Agenda page and click on the event that need to be deleted. This will navigate to Add/Edit page along with event string details and we can delete it by pressing 'Delete' button. This is done by retrieving data stored in the JSON file and comparing it with event string passed and if matches corresponding entry is deleted.

ii. DFD


Home Screen
Add/Edit/Delete event
Event Screen
Update State
Read and intialize the state
Data
Store

iii. Class Diagram



iv. Methodology

a. Basic React Native Components

View:
The most fundamental component for building a UI, View is a container that supports layout with flexbox, style, some touch handling, and accessibility controls. View maps directly to the native view equivalent on whatever platform React Native is running on, whether that is a UIView, , android.view, etc. View is designed to be nested inside other views and can have 0 to many children of any type.

Image:
A React component for displaying different types of images, including network images, static resources, temporary local images, and images from local disk, such as the camera roll.

Image component is used with prop “source”

```
<Image
  style={{width: 50, height: 50}}
  source={{uri: 'https://facebook.github.io/react-native/docs/assets/favicon.png'}}
  />
```

Props:
Most components can be customized when they are created, with different parameters. These creation parameters are called props. For example, one basic React Native component is the Image. When you create an image, you can use a prop named source to control what image it shows.

State:
There are two types of data that control a component: props and state. props are set by the parent and they are fixed throughout the lifetime of a component. For data that is going to change, we have to use state. In general, you should initialize state in the constructor, and then call setState() when you want to change it.

Text :
A react component for displaying text. Text supports nesting, styling, and touch handling.

Text Input:
It a basic component that allows the user to enter text. It has an onChangeText prop that takes a function to be called every time the text changed, and an onSubmitEditing prop that takes a function to be called when the text is submitted.

Button:
Button-basic button component-This will render a blue label on iOS, and a blue rounded rectangle with white text on Android. Pressing the button will call the "_onPressButton"

function, which in this case displays an alert popup. If you like, you can specify a "color" prop to change the color of your button.

Flexbox:
Flexbox-component designed to provide a consistent layout on different screen sizes
-flexDirection
- alignItems
- justifyContent

```
<View style={{flex: 1, flexDirection: 'row'}}>
</View>
```

ScrollView:
ScrollView Component for scrolling both vertically and horizontally on the screen

List Views:
- FlatList- displays a scrolling list of changing, but similarly structured, data.
-SectionList-Provide an array of data and a render Item function

Touchable Components:
The "Touchable" components provide the capability to capture tapping gestures, and can display feedback when a gesture is recognized.
Touchable Components
- **TouchableHighlight**-background will be darkened when the user presses down on the button

-**TouchableOpacity**- background to be seen through while the user is pressing down

-**TouchableWithoutFeedback**-If you need to handle a tap gesture but you don't want any feedback to be displayed

-**TouchableNativeFeedback**- surface reaction ripples that respond to the user's touch

```
<TouchableNativeFeedback onPress={this._onPressButton}
background={TouchableNativeFeedback.Ripple('blue')}
delayPressIn={1}>
```

-long presses can be handled by passing a function to the “onLongPress “ props of any of the "Touchable" components.
ActivityIndicator:
Displays a circular loading indicator

b. APIs Used

AppRegistry:
AppRegistry is the JS entry point to running all React Native apps. App root components should register themselves with AppRegistry.registerComponent, then the native system can load the bundle for the app and then actually run the app when it's ready by invoking AppRegistry.runApplication.

AppRegistry.registerComponent('AwesomeProject', () => AwesomeProject);

AsyncStorage:
AsyncStorage is a simple, unencrypted, asynchronous, persistent, key-value storage system that is global to the app. It should be used instead of LocalStorage.
Methods:
getItem()
Fetches an item for a key and invokes a callback upon completion,returns an object.
setItem()
Sets the value for a key and invokes a callback upon completion.
removeItem()
Removes an item for a key and invokes a callback upon completion.

DatePickerAndroid:
Opens the standard Android date picker dialog.
Methods:
open()
static open(options)
Opens the standard Android date picker dialog.

The available keys for the options object are:

date (Date object or timestamp in milliseconds) - date to show by default
minDate (Date or timestamp in milliseconds) - minimum date that can be selected
maxDate (Date object or timestamp in milliseconds) - maximum date that can be selected
mode (enum('calendar', 'spinner', 'default')) - To set the date-picker mode to calendar/spinner/default
'calendar': Show a date picker in calendar mode

ListViewDataSource:
Provides efficient data processing and access to the ListView component. A ListViewDataSource is created with functions for extracting data from the input blob, and comparing elements (with default implementations for convenience). The input blob can be as simple as an array of strings, or an object with rows nested inside section objects. To update the data in the datasource, use cloneWithRows().

Methods:
constructor(params)
You can provide custom extraction and hasChanged functions for section headers and rows. If absent, data will be extracted with the defaultGetRowData and defaultGetSectionHeaderData functions.

cloneWithRows()
cloneWithRows(dataBlob, rowIdentities);
Clones this ListViewDataSource with the specified dataBlob and rowIdentities. The dataBlob

is just an arbitrary blob of data. The rowIdentities is a 2D array of identifiers for rows. ie. [['a1', 'a2'], ['b1', 'b2', 'b3'], ...].

getRowCount();
Returns the total number of rows in the data source.

rowShouldUpdate(sectionIndex, rowIndex);
Returns if the row is dirtied and needs to be rerendered getRowData

getSectionLengths();
Returns an array containing the number of rows in each section

sectionHeaderShouldUpdate(sectionIndex);
Returns if the section header is dirtied and needs to be rerendered

getSectionHeaderData(sectionIndex);
Gets the data required to render the section header

StyleSheet:

A StyleSheet is an abstraction similar to CSS StyleSheets.
Create a new StyleSheet:

```
const styles = StyleSheet.create({
   container: {
      borderRadius: 4,
      borderWidth: 0.5,
      borderColor: '#d6d7da',
   },
   title: {
      fontSize: 19,
      fontWeight: 'bold',
   },
activeTitle: {
color: 'red',
  },
});
```

Use a StyleSheet:

```
<View style={styles.container}>
   <Text style={styles.title} />
</View>
```

c. <u>NPM Packages Used(NodeJS Package Manager)</u>

react-native-datepicker:

install DatePicker component:
npm install react-native-datepicker –save

Add in your program:
import DatePicker from 'react-native-datepicker';

<DatePicker style={{width: 200}}
date={this.state.date}
mode="date"
placeholder="placeholder"
format="YYYY-MM-DD"
minDate="2017-01-01"
maxDate="2017-12-31"
confirmBtnText="Confirm"
cancelBtnText="Cancel"
onDateChange={(date) => {this.setState({date: date});}}
/>

react-native-vector-icons:

install component:
npm install react-native-vector-icons –save
Add in your program:
import Icon from 'react-native-vector-icons/Entypo';

react-native-simple-store:

A minimalistic wrapper around React Native's **AsyncStorage** and stores events in **JSON** file.

AsyncStorage: is a simple, unencrypted, asynchronous, persistent, key-value storage system that is global to the app. It should be used instead of local storage.

It is recommended that you use an abstraction on top of AsyncStorage instead of AsyncStorage directly for anything more than light usage since it operates globally.

Methods used:

1. setItem():
   Sets the value for a key and invokes a callback upon completion
   AsyncStorage.setItem('@MySuperStore:key', 'I like to save it.')
2. getItem():
   Fetches an item for a key and invokes a callback upon completion

AsyncStorage.getItem('@MySuperStore:key')

JSON.parse():

Parse the data with JSON.parse(), and the data becomes a JavaScript object

react-native-simple-store:

install component:

npm install react-native-simple-store

Add in your program:

import store from 'react-native-simple-store';

Methods used:

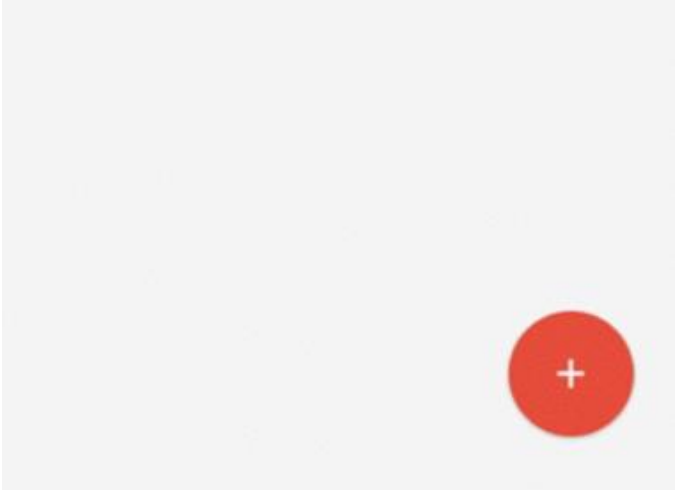
store.save(key,value)

store.get(key)

store.delete(key)

react-native-action-button:

customizable multi-action-button component for react-native.



Install component:

npm i react-native-action-button –save

Add in your program:

import ActionButton from 'react-native-action-button';

ActionButton component is the main component which wraps everything and provides a couple of props.

```
<ActionButton
        buttonColor="rgba(231,76,60,1)"
onPress={this.addEvent.bind(this)}
/>
```

react-navigation:

for navigating between different screens
Link:
Install component:
npm install --save react-navigation
Add in your program:

```
import { StackNavigator } from 'react-navigation';

const SimpleApp = StackNavigator({
Home: { screen: HomeScreen },

Chat: { screen: ChatScreen },
});
```

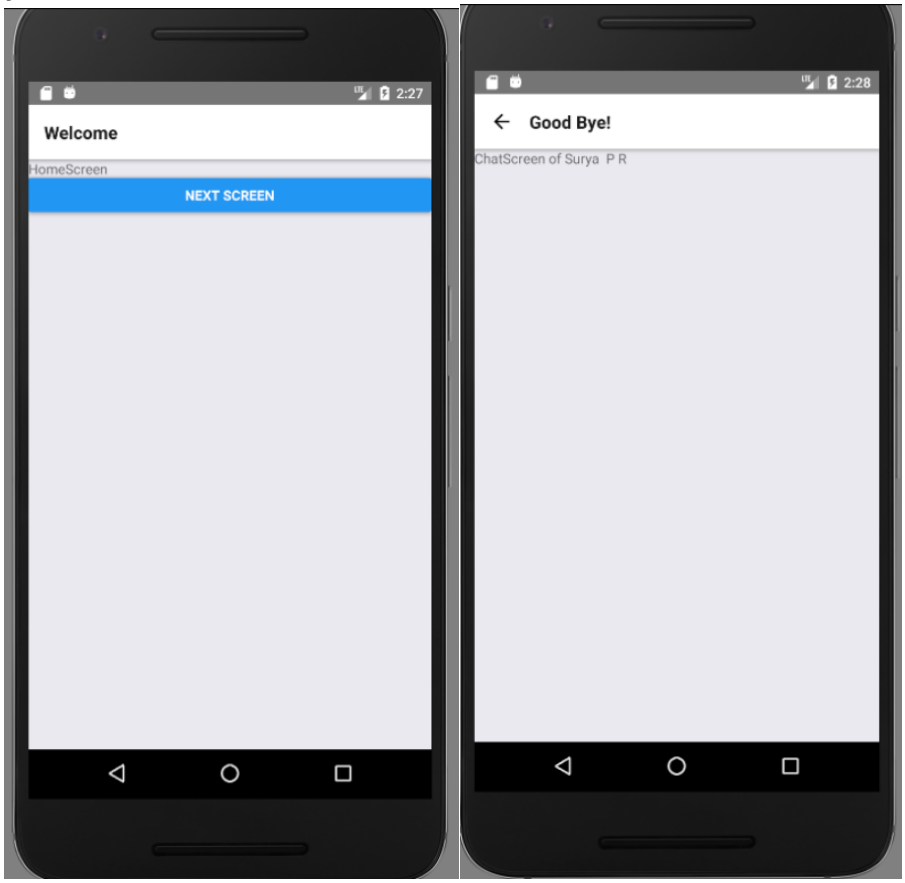
HomeScreen

```
class HomeScreen extends React.Component
{
static navigationOptions =
{
title: 'Welcome',
};
render()
{
const { navigate } = this.props.navigation;
return (
<View>
<Text>HomeScreen</Text>
<Button
onPress={() => navigate('Chat',{user:'Surya',lname: ' P R'})}
title="next screen"
/>
</View>
);
}
}
```

ChatScreen

```
class ChatScreen extends React.Component
{
static navigationOptions = {
title: 'Good Bye!',
};
render() {
// The screen's current route is passed in to `props.navigation.state`:
```

```
const { params } = this.props.navigation.state;
return (
<View>
<Text>ChatScreen of {params.user} {params.lname} </Text>
</View>
);
}
}
```



StackNavigator:

Each new screen is put on the top of the stack and going back removes a screen from the top of the stack

Screen Navigation Prop:

Each screen in your app will receive a navigation prop which contain the following method

Navigate()

Call this to link to another screen in your app

navigate(routeName, params)

react-native-calendars:

This npm package is compatible for both Android and iOS. This module includes various customizable react native calendar components.

Install the package using following command

npm install --save react-native-calendars

Calendar objects are defined as follows
{
day:1, // day of month (1-31)
month:1, // month of year (1-12)
year:2017, // year
timestamp, // UTC timestamp representing 00:00 AM of this date
dateString: '2016-05-13' // date formatted as 'YYYY-MM-DD' string
}

For using the package import the components as follows
import { Calendar, CalendarList, Agenda } from 'react-native-calendars';

• componentWillReceiveProps()
This method is called when props are passed to the Component instance.
Default terms used in Project

export { default as Calendar } from './calendar';
export { default as CalendarList } from './calendar-list';
export { default as Agenda } from './agenda';

d. JS Methods and Concepts Used

Date():

Creating Date Objects:
Eg:
var d = new Date();
document.getElementById("demo").innerHTML = d;
// Thu Jul 13 2017 14:47:47 GMT-0500 (Central Daylight Time)

getFullYear():
JS method returns the year (four digits for dates between year 1000 and 9999) of the
var d = new Date();
var n = d.getFullYear(); //2017

getMonth() :
JS method returns the month (from 0 to 11) for the specified date, according to local time. January is 0, February is 1, and so on.
var d = new Date();
var n = d.getMonth(); //6

PropTypes:

By defining PropTypes, we can create self-documenting components that tell other developers (or future versions of ourselves) what props we expect a component to have

and of what type like Is this a string?, Is this an object?,Is this required?
PropTypes also protect our interfaces from breaking down due to incorrect usage. If a component doesn't validate the props being passed to it, we run the risk of accidentally passing in the wrong type of data causing glitches in our interface for users.
React.PropTypes has moved to package prop-types from React version 15.5.0
Eg:

```
<Field data={data} />
propTypes: {
   data: React.PropTypes.object
}
```

-to validate the values inside data ie. , data.id, data.title

```
import PropTypes from 'prop-types';
propTypes: {
      data: PropTypes.shape({
         id: PropTypes.number.isRequired,
         title: PropTypes.string
      })}
```

- the value of **this** keyword is determined by how a function is called. It can't be set by assignment during execution, and it may be different each time the function is called. ES5 introduced the bind method to set the value of a function's this regardless of how it's called, and ES2015 introduced arrow functions who's this is lexically scoped (it is set to this value of the enclosing execution context)
- An **arrow function expression** has a shorter syntax than a function expression and does not bind its own this, arguments, super, or new. Target. These function expressions are best suited for non-method functions, and they cannot be used as constructors.
  (param1, param2, …, paramN) => {statements}
  (param1, param2, …, paramN) => expression
  () =>{statements} //no arguments for function
  ()=>expression
- **bind()** is helpful in cases where you want to create a shortcut to a function which requires a specific this value. Syntax: fun.bind(thisArg[, arg1[, arg2[, ...]]])
- **slice()-**method returns a shallow copy of a portion of an array into a new array object selected from begin to end (end not included).
  Array.prototype.slice, also used for converting an array-like object to a real array
- **map()** -method creates a new array with the results of calling a provided function on every element in the calling array
  Eg:

```
var numbers = [1, 5, 10, 15];
var doubles = numbers.map(function(x) {      return x * 2;});
// doubles is now [2, 10, 20, 30]
```

- **concat()** - method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

- **push()** -method adds one or more elements to the end of an array and returns the new length of the array.
  Eg:
  var numbers = [1, 2, 3];
  numbers.push(4);
  console.log(numbers); // [1, 2, 3, 4]
  numbers.push(5, 6, 7);
  console.log(numbers); // [1, 2, 3, 4, 5, 6, 7]
- **splice()** -method changes the contents of an array by removing existing elements and/or adding new elements.
  Eg:
  var myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];
  myFish.splice(2, 0, 'drum'); // insert 'drum' at 2-index position
  // myFish is ["angel", "clown", "drum", "mandarin", "sturgeon"]

  myFish.splice(2, 1); // remove 1 item at 2-index position (that is, "drum")
  // myFish is ["angel", "clown", "mandarin", "sturgeon"]
- The **extends** keyword is used in class declarations or class expressions to create a class which is a child of another class.
- The **super** keyword is used to call functions on an object's parent
  super([arguments]); // calls the parent constructor.
  super.functionOnParent([arguments]);
- **Comments in React JSX**
  {/* A JSX comment */}
  {/*
  Multi
  line
  comment
  */}

XDate - A Modern JavaScript Date Library

XDate is a thin wrapper around JavaScript's native Date object that provides enhanced functionality for parsing, formatting, and manipulating dates.

- XDate()
- formatting

  MM : month number, 2-digits
  MMMM : month name, full (like "January")
  yy : year, 2-digits
  yyyy : year, 4-digits
  XDate(d.toString('yyyy-MM-dd'), true)
- getFullYear()
- getMonth()
- getDate()
- addMonths(months, preventOverflow)

this.props.addMonths(1)
this.props.addMonths(-1)

- getTime()
- Diffing

  The following methods return the amount of time that must be added to the XDate to arrive at otherDate.
  .diffYears(otherDate) .diffMonths(otherDate) .diffWeeks(otherDate)
  .diffDays(otherDate) .diffHours(otherDate) .diffMinutes(otherDate)
  .diffSeconds(otherDate) .diffMilliseconds(otherDate)
  Eg:
  var thanksgiving = new XDate(2011, 10, 24);
  var christmas = new XDate(2011, 11, 25);
  thanksgiving.diffDays(christmas); // 31
  christmas.diffDays(thanksgiving); // -31
- .clone()
  returns a copy of the XDate
- You can add new locales by adding a new object to the `XDate.locales` .
- You can change the default locale by changing `XDate.defaultLocale`.
- require('xdate') is built into Node.js to load modules

- You can add new locales by adding a new object to the `**XDate.locales**` .You can change the default locale by changing `**XDate.defaultLocale**
  XDate.locales['fr'] = {
  monthNames: […….],
  monthNamesShort: […],
  dayNames: […….],
  dayNamesShort: […….]
  };
  new XDate().toString('d MMMM, yyyy', 'fr');
  // or, you set the default locale...
  XDate.defaultLocale = 'fr';
  new XDate().toString('d MMMM, yyyy');
- **JavaScript Output**
  Writing into an HTML element, using innerHTML.
  Writing into the HTML output using document.write().
  Writing into an alert box, using window.alert().
  Writing into the browser console, using console.log().
- **dateObj.toISOString()**-method returns a string in simplified extended ISO format (ISO 8601), which is always 24 or 27 characters long (YYYY-MM-DDTHH:mm:ss.sssZ or ±YYYYYY-MM-DDTHH:mm:ss.sssZ, respectively). The timezone is always zero UTC offset, as denoted by the suffix "Z". Eg: "2016-09-08T12:01:11.832Z".

- **let** :allows you to declare block-level variables. The declared variable is available from the block it is enclosed in
- **const** : allows you to declare variables whose values are never intended to change. The variable is available from the block it is declared in
- **var** is the most common declarative keyword. It does not have the restrictions that the other two keywords have. This is because it was traditionally the only way to declare a variable in JavaScript. A variable declared with the var keyword is available from the function it is declared in
- **for...of**

  for (let value of array) {
  // do something with value
  }
- **for...in**

  for (let property in object) {
  // do something with object property
  }
- **create an empty object**

  var obj = new Object();
  var obj = {};
  Object literal syntax can be used to initialize an object in its entirety
  var obj = {
  name: 'Carrot',
  details: {
  color: 'orange',
  size: 12
  }
  }
  obj.details.color; // orange
- **array**

var a = new Array();
a[0] = 'dog';
a[1] = 'cat';

a[2] = 'hen';
a.length; // 3
var a = ['dog', 'cat', 'hen'];
a.length; // 3
a.push(item);// If you want to append an item to an array

- **a.toString()**
  Returns a string with the toString() of each element separated by commas.
- **a.toLocaleString()**
  Returns a string with the toLocaleString() of each element separated by commas.

e. Running the project Using Expo XDE

Expo XDE

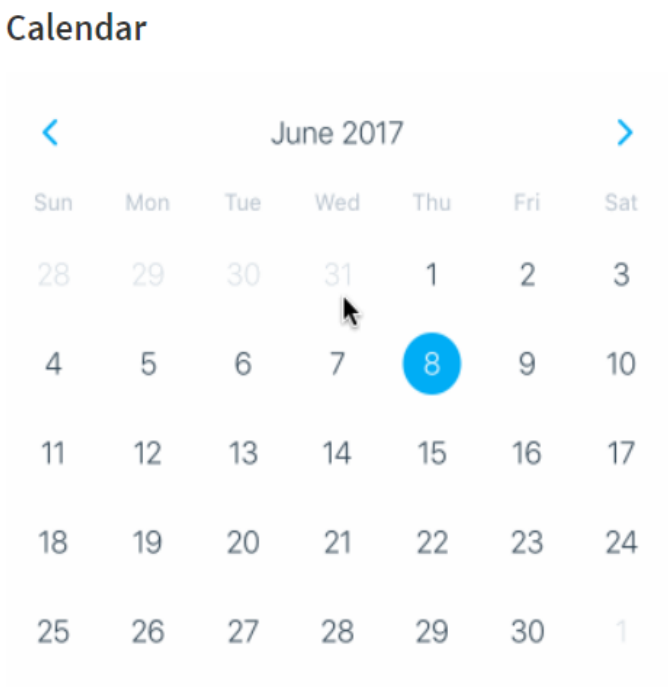
Free open-source project for building native IOS and android app in Java Script
Built at top of react-native, a popular library that lets you controls the native UI components from IOS and android using Java Script.

1. Install Expo from Google play store on android mobile.
2. Register as a new user
3. Create project:
   create-react-native-app AwesomeProject
4. Go to project folder and start server:
   cd AwesomeProject
   npm start
5. Scan QR code from command prompt to your downloaded Expo app. Then it will display output of your project.

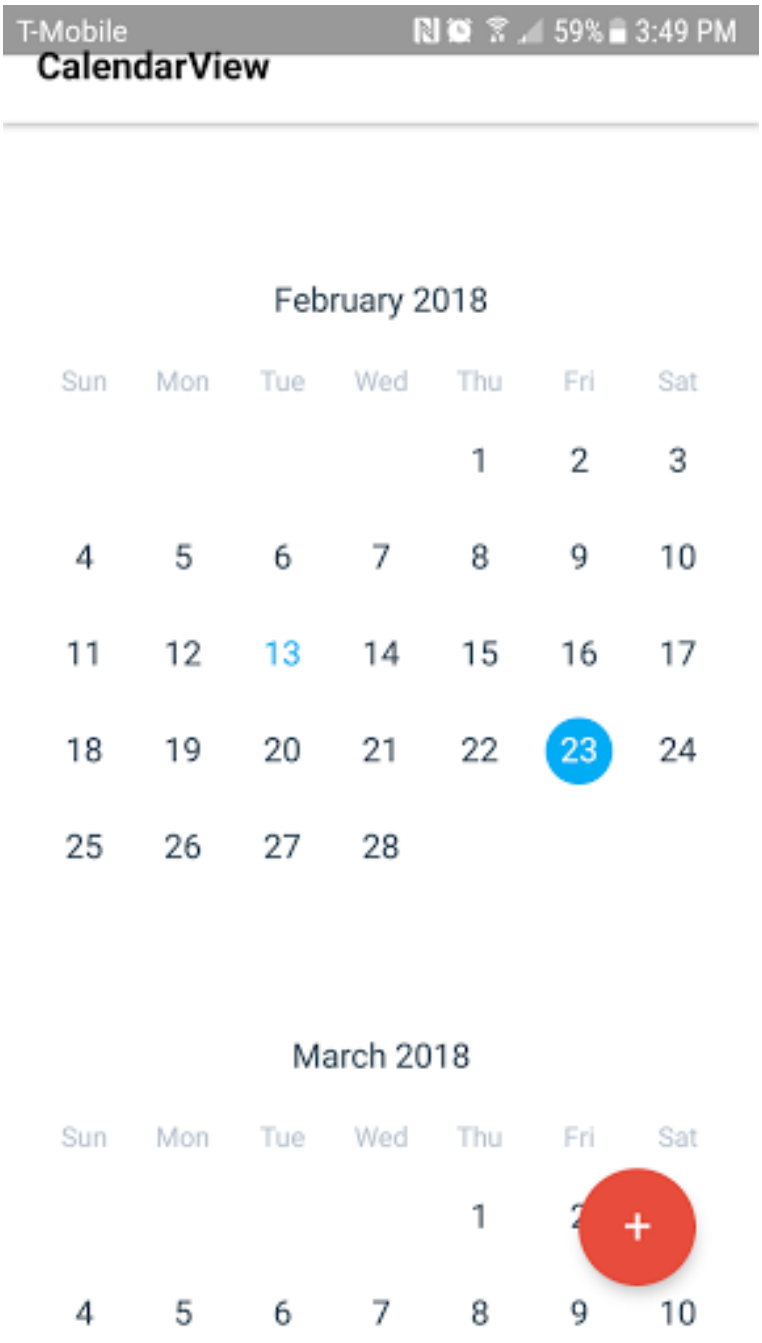
• **npm start**
When you issue the command npm start from the root directory of your nodejs project, node will look for a scripts object in your package.json file. If found, it will look for a script with the key start and run the command specified as its value.If your package.json does not have any scripts object or if the scripts object does not have a start key, it will run the command node server.js instead.

vi. Screenshots of Output

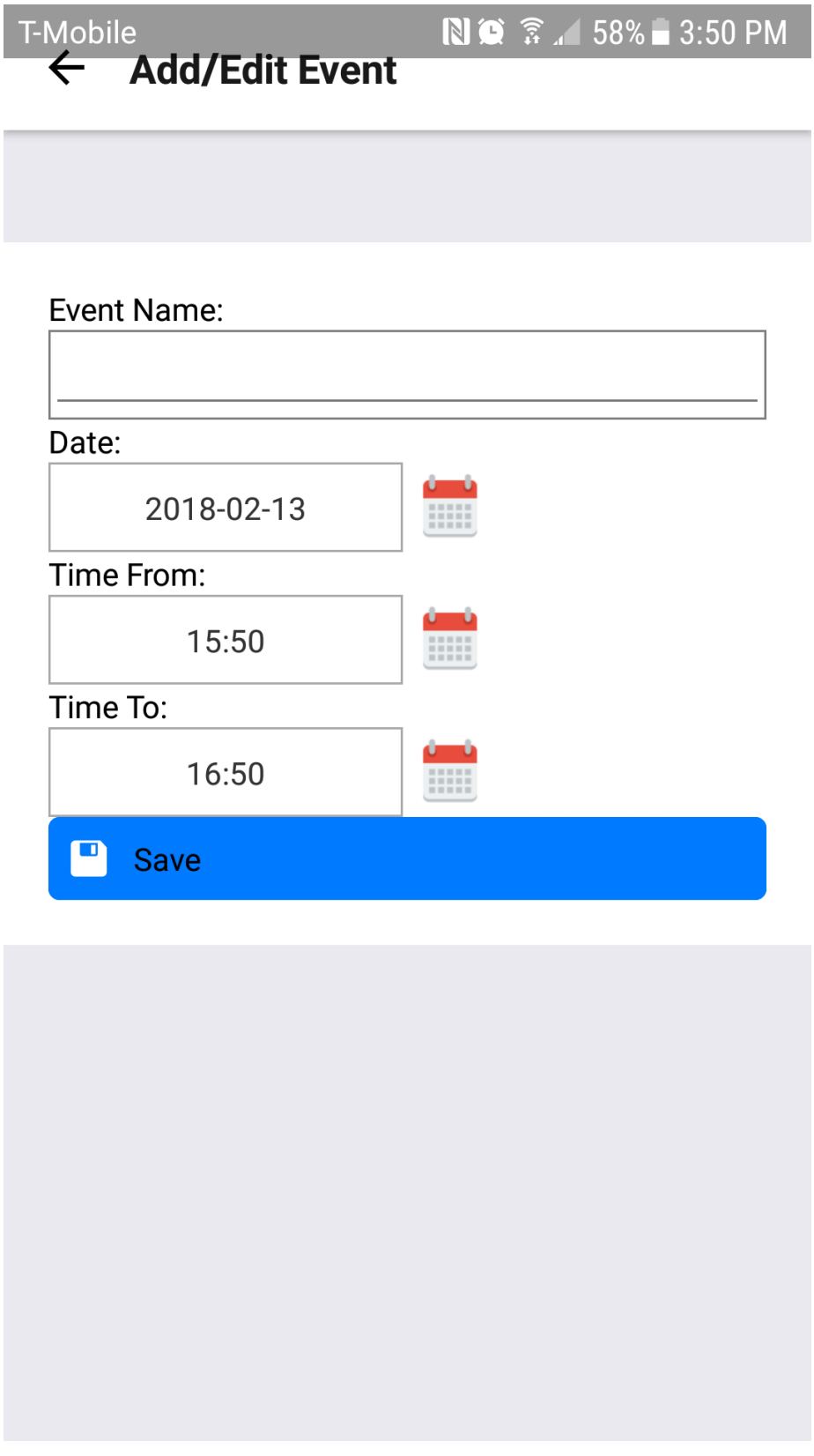
Calendar



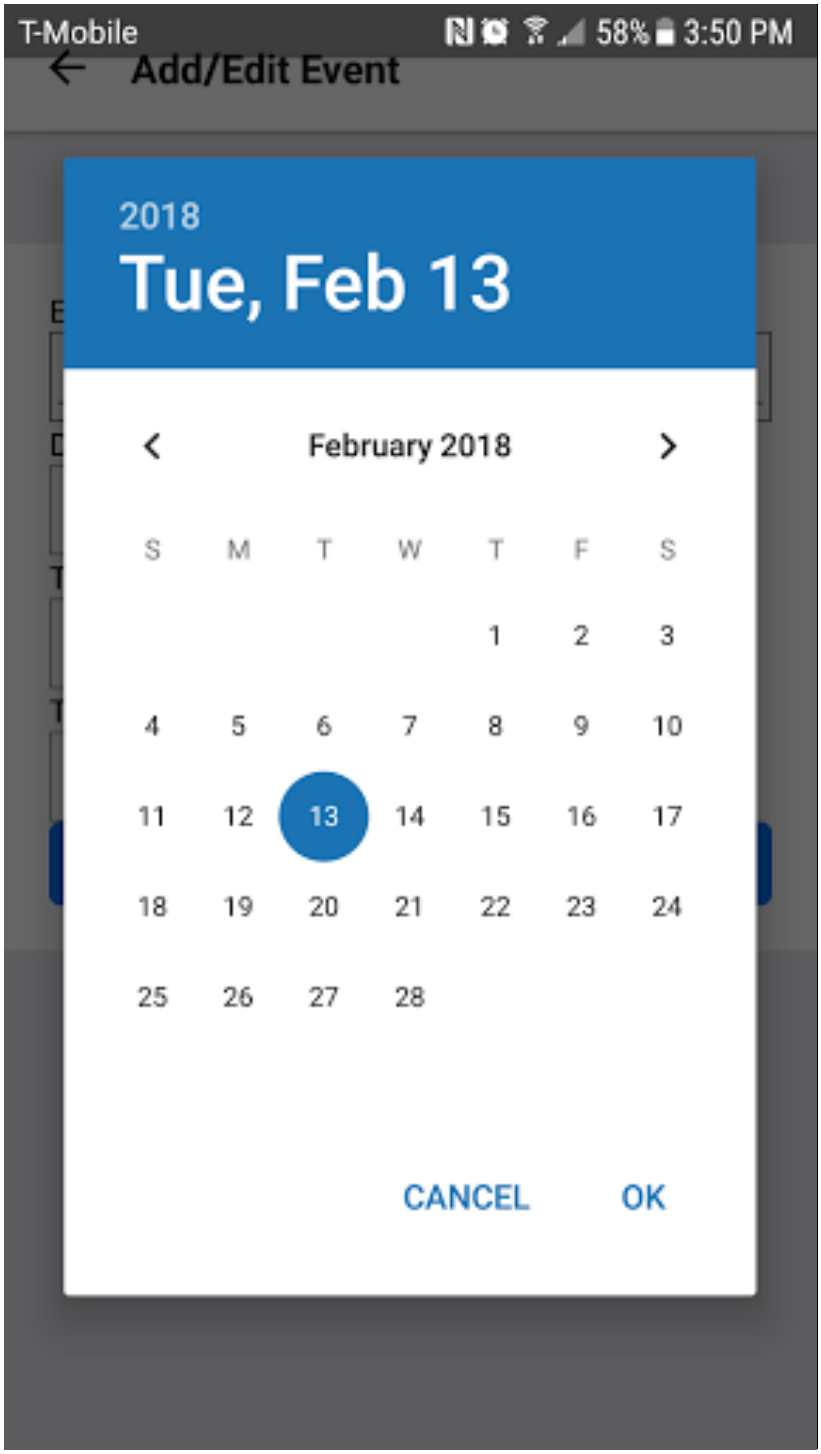
CalendarList

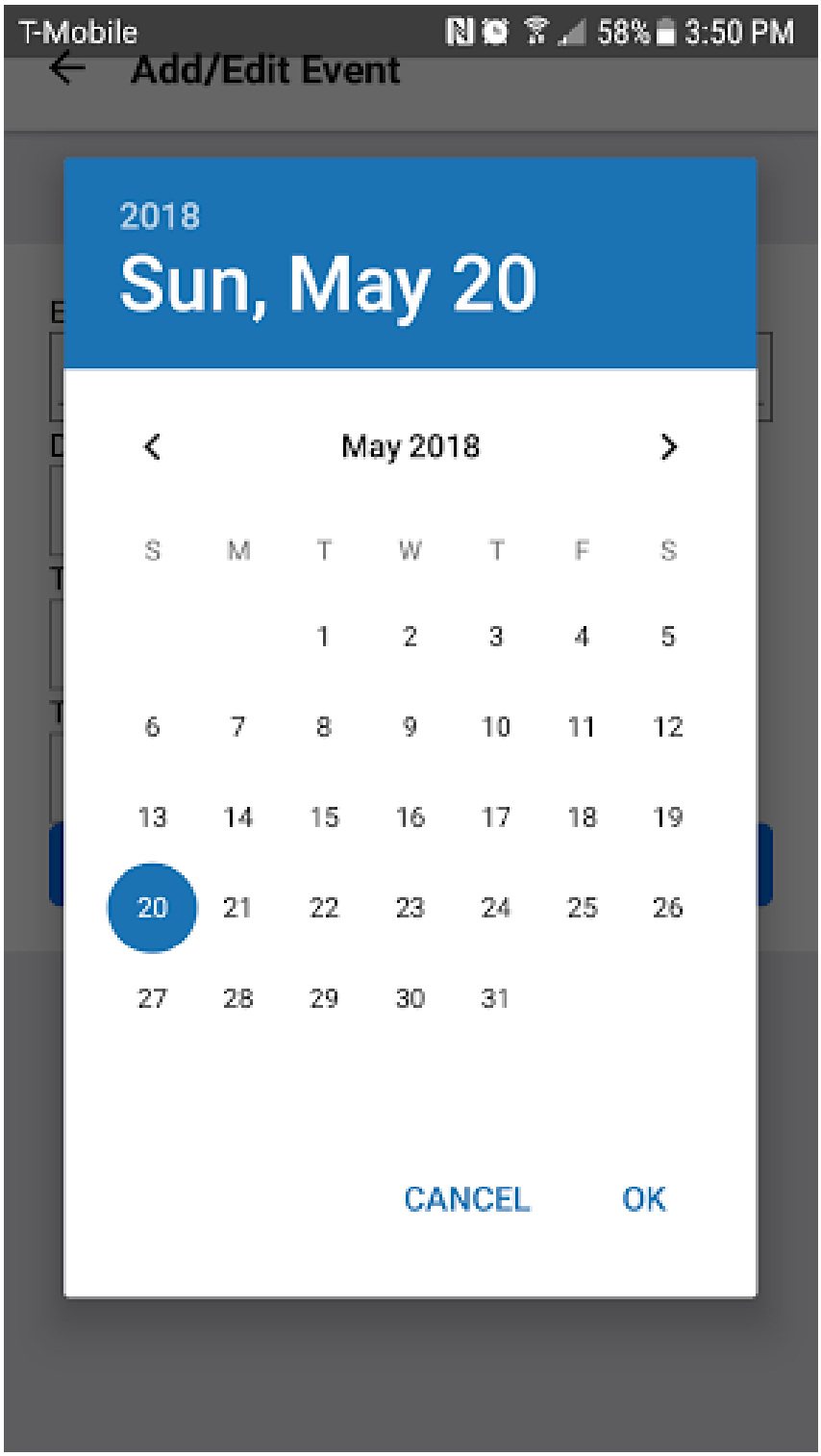


Event Page

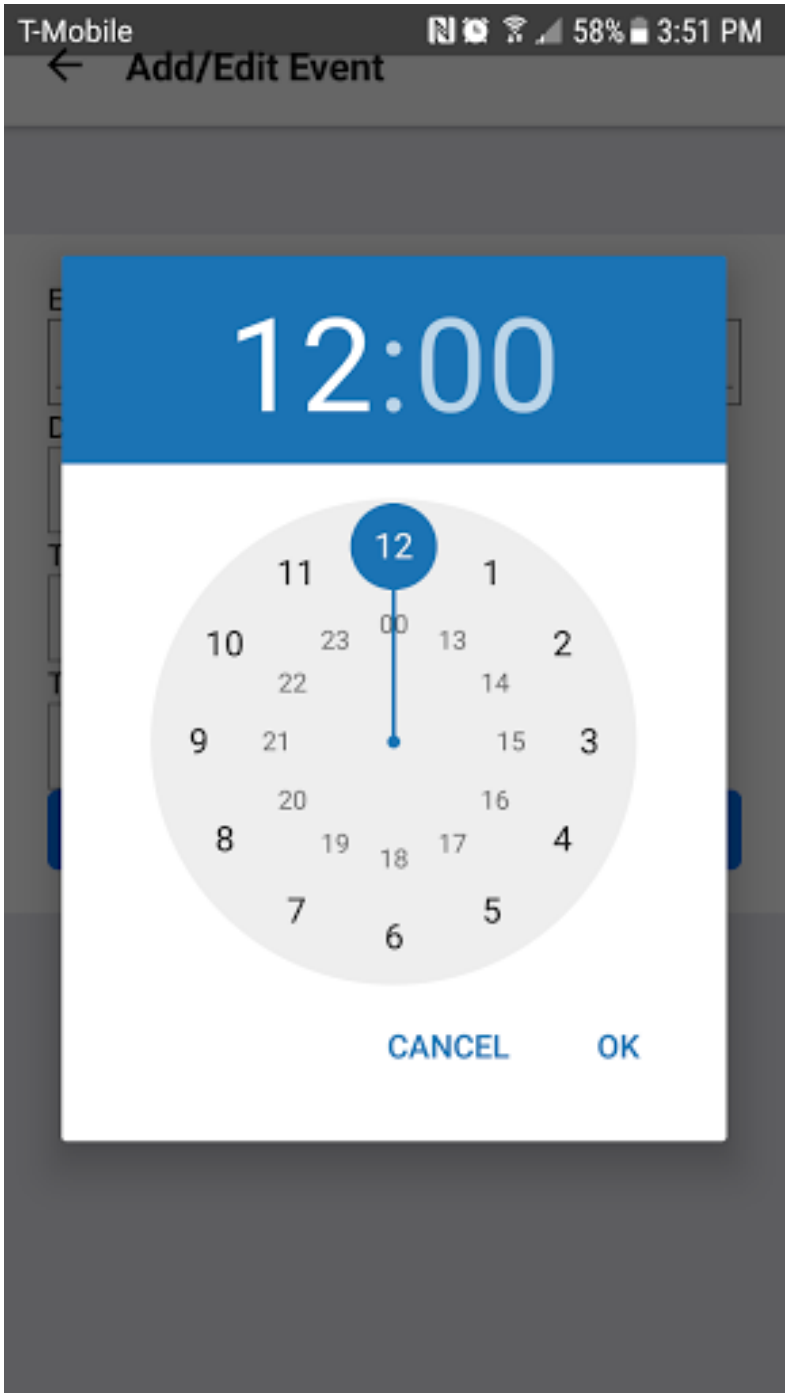

T-Mobile
58%
3:50 PM
Add/Edit Event
Event Name:
Date:
2018-02-13
Time From:
15:50
Time To:
16:50
Save

On Clicking Date icon in Event page

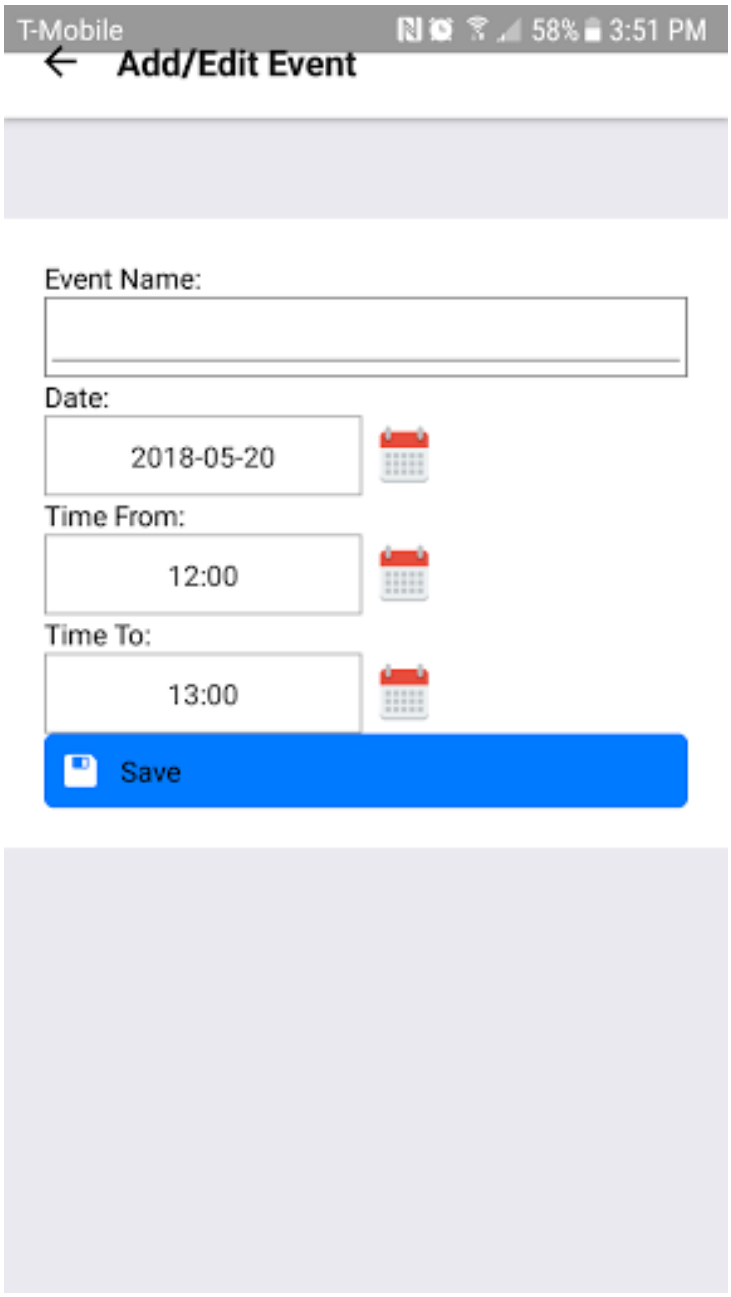



T-Mobile
58% 3:50 PM
Add/Edit Event
2018
Sun, May 20
May 2018
S M T W T F S
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
CANCEL
OK

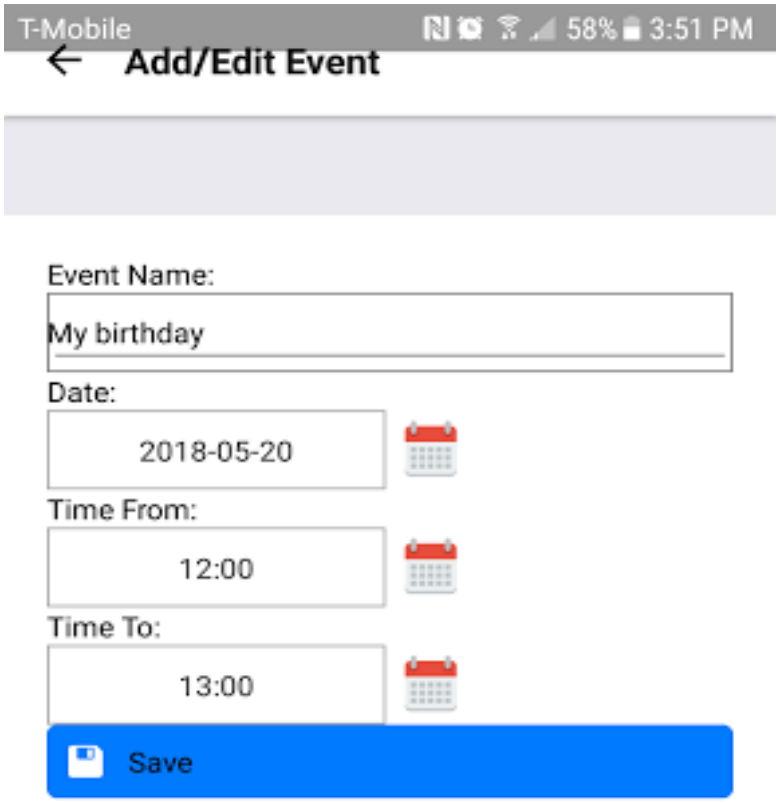
On Clicking Time icon on Event page



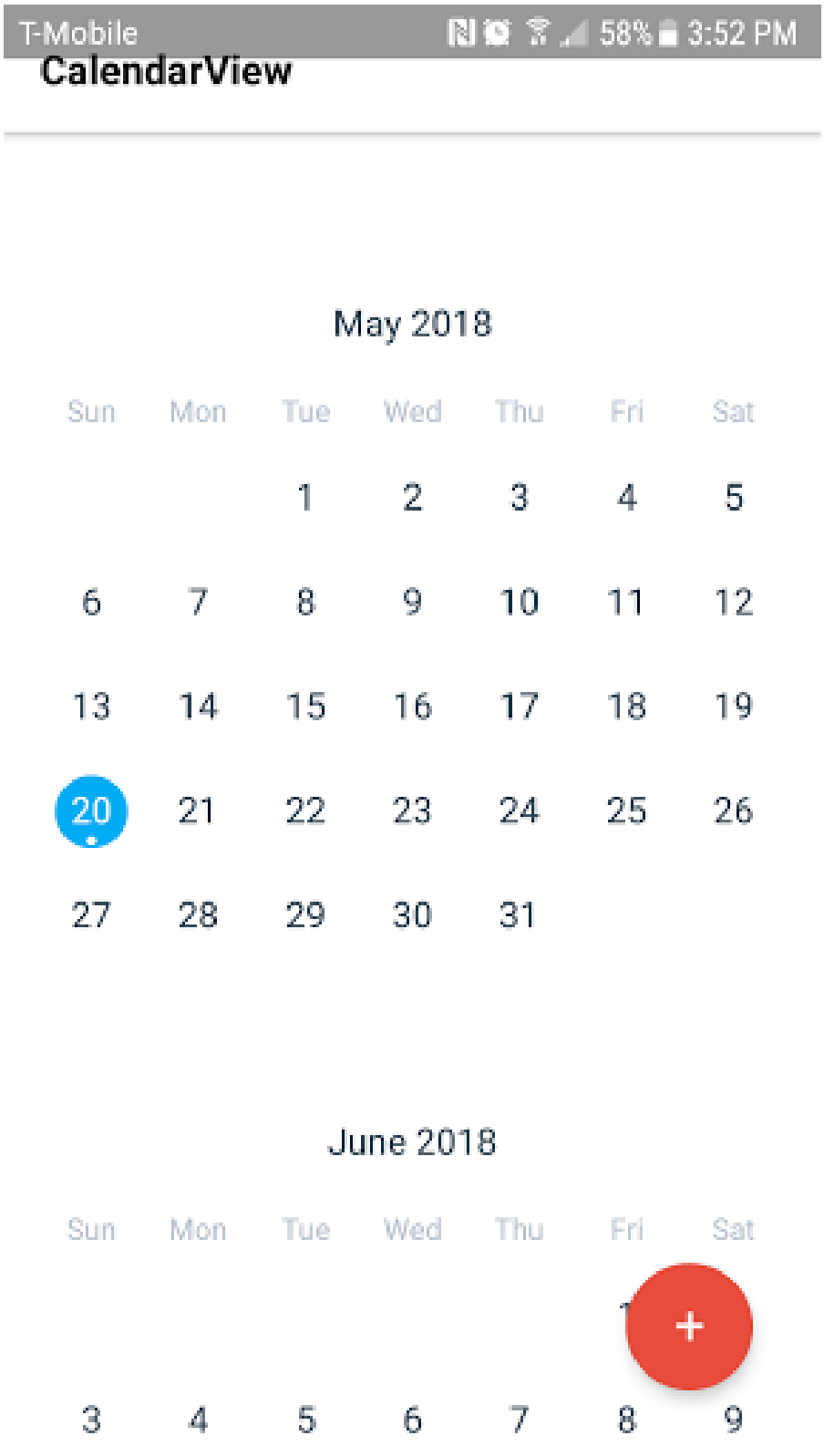
Event Page After selecting date and time



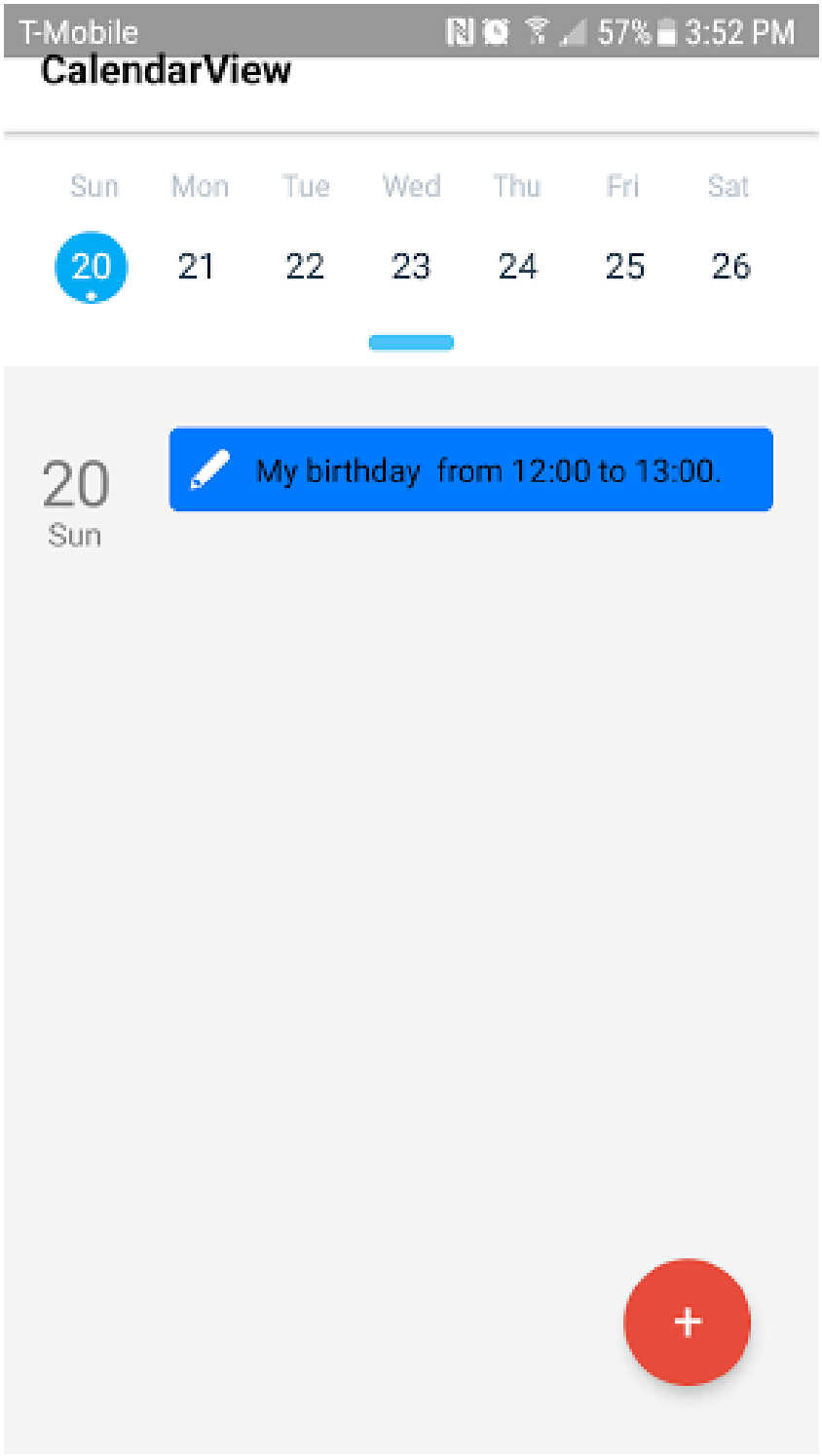
Entering Event name in text box



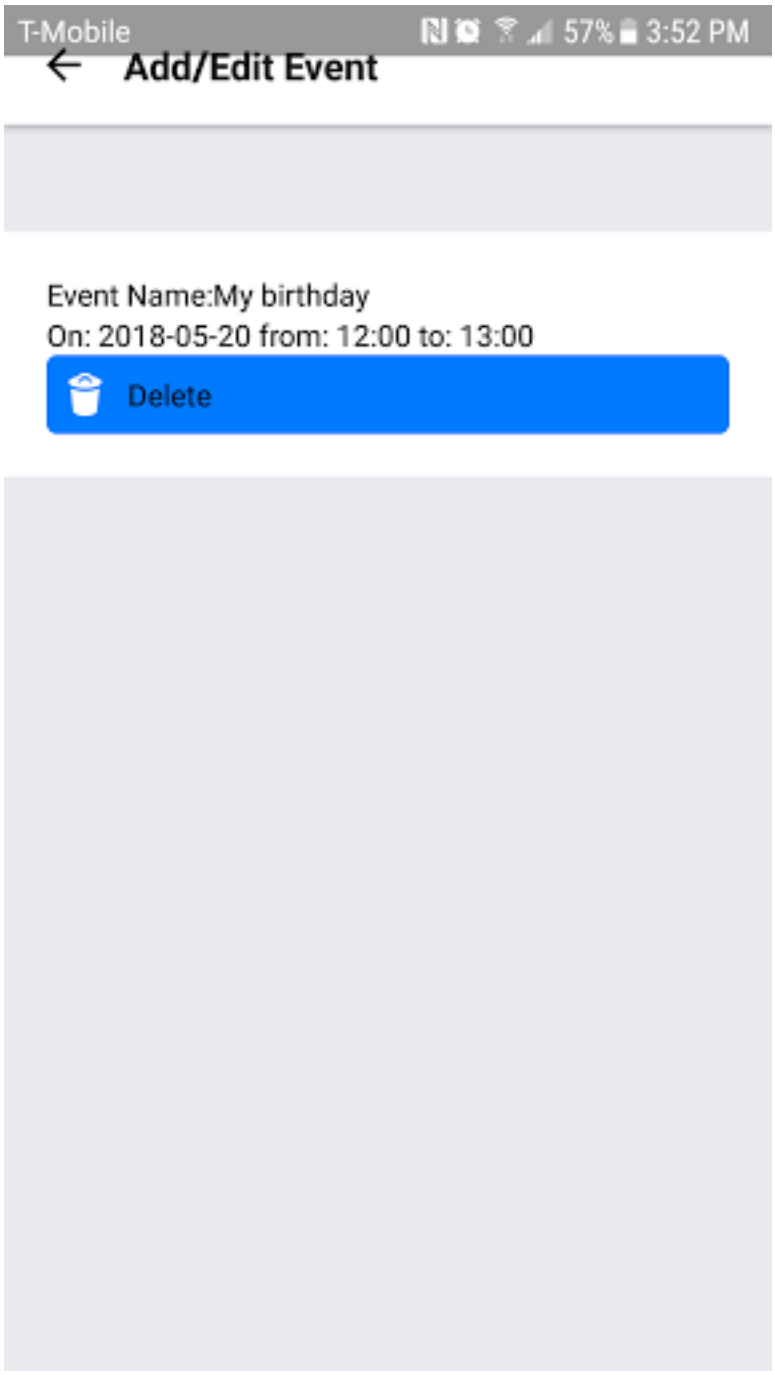
Displaying month after saving event



Viewing the event on a particular date



Deleting event after clicking on edit button



CHAPTER III CONCLUSION

Developing this calendar application revealed flexibility of react-native framework and tools that it provides, both to develop and deploy this application in various mobile platforms. This also gives a greater understanding of object-oriented design and helps me understand the power of modular designs and reusable component approach that object-oriented design advocates.

CHAPTER IV REFERENCES

React Native · A Framework For Building Native Apps Using React
https://facebook.github.io/react-native/

React-native-calendars
https://www.npmjs.com/package/react-native-calendars

Introduction
DevelopmentArc - https://developmentarc.gitbooks.io/react-indepth/