

Building User Interfaces

React Native 2

Intermediate Concepts

Professor Yuhang Zhao

What we will learn today?

- Mobile Navigation using React Native
- Mobile Input via Gestures using React Native
- Working with Date object in JS
- Assignment Preview

Mobile Navigation using React Native

The Options

There are two main ways of implementing navigation in RN:

1. Using ReactJS navigation, i.e., `react-navigation`
2. Using RN navigation, i.e., `react-native-navigation`

We will be covering `react-navigation` in depth. `react-native-navigation` is for advanced use, as it involves modifying native components, while `react-navigation` is programmed in JS.

Setting up ReactJS

Install react-navigation for RN:

```
npm install @react-navigation/native
```

Install dependencies:

```
npm install react-native-reanimated  
react-native-gesture-handler  
react-native-screens  
react-native-safe-area-context  
@react-native-community/masked-view
```

How does navigation in HTML work?

The History API¹ provides a `Window` object that gives access to a history object, which includes a stack of all the pages that the user has previously visited.

When a new link (`<a>`) is pressed, the current URL is pushed to the history stack. The "back" button calls the following function.

```
window.history.back()
```

¹[More on the History API](#)

When the "forward" button is pressed, it calls the following function.

```
window.history.forward()
```

We can also navigate in the stack and access a particular URL in the history:

```
window.history.go(3);
```

How does navigation in RN work?

RN provides a set of *navigators* that accomplish stack-based and other types of navigation:

1. Switch navigator
2. Stack navigator
3. Tab navigator
4. Drawer navigator

Switch Navigator

Definition: Enables showing one screen at a time and does not involve "back" actions. Used primarily in authentication flows.

Stack Navigator

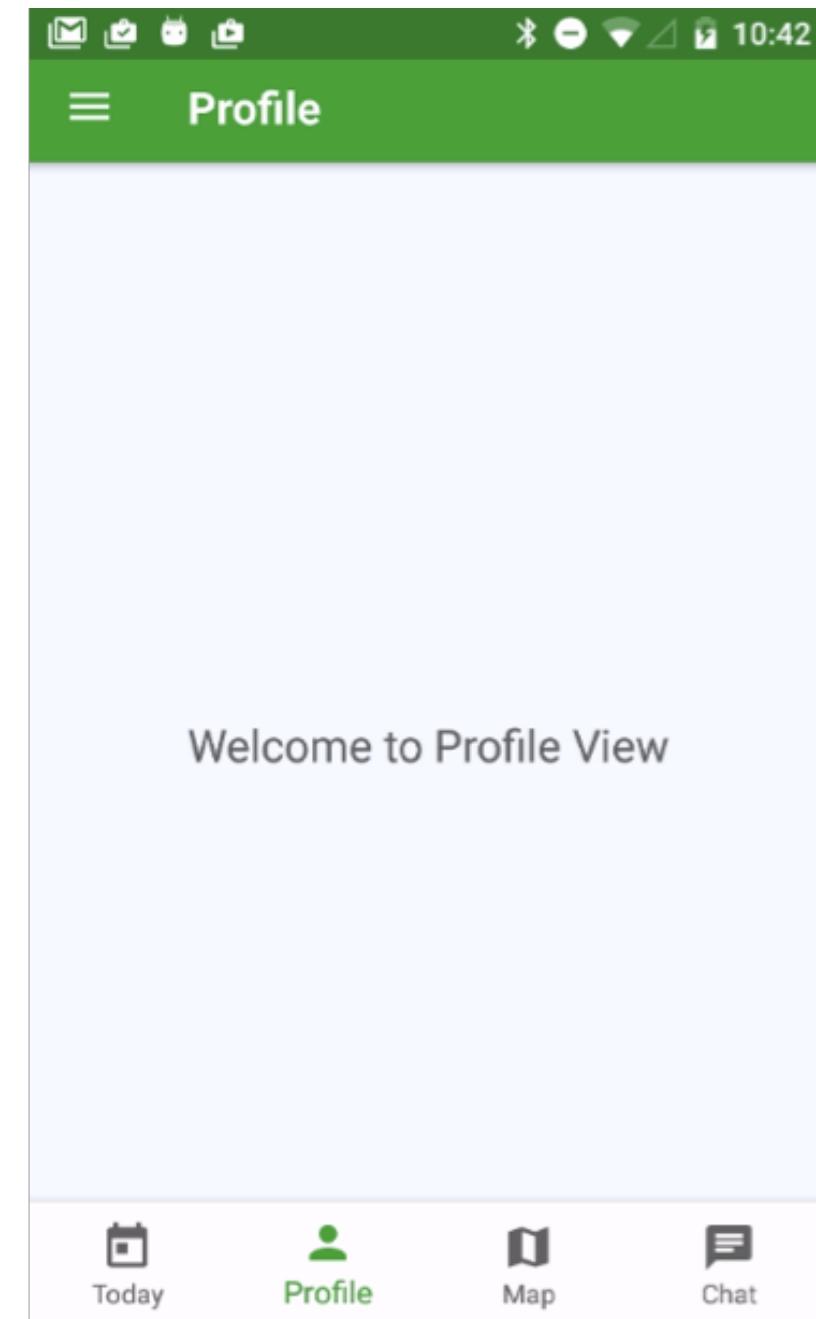
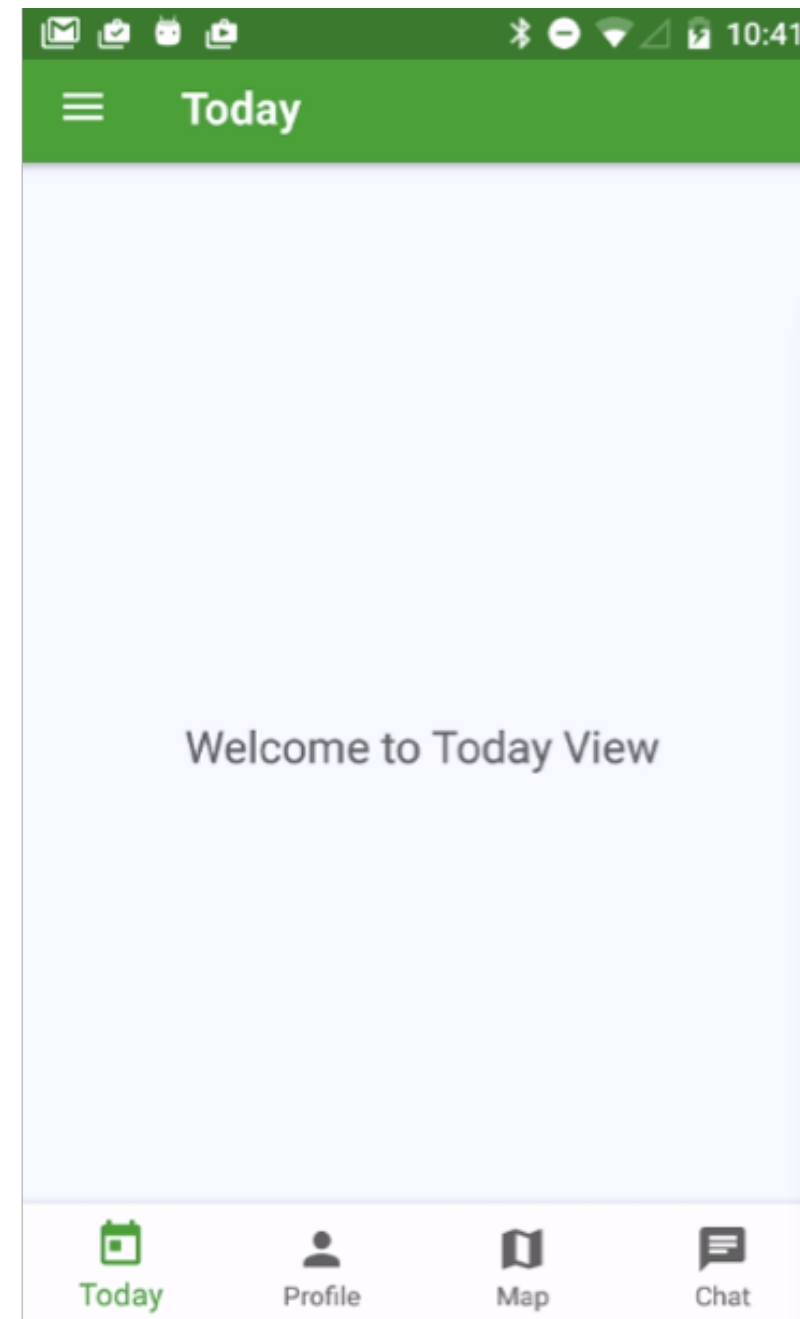
Definition: Enables transition between screens where each screen is placed on a stack, as the History API does. The navigator automatically implements the native transition animations.

Primarily used to go back and forth between list and detail views or to walk the user through a process.

Tab Navigator²

Definition: Implements tabs at the bottom or the top of the screen to enable transitions among them.

Most commonly used navigation to establish a main menu for the different sections/parts of an application.

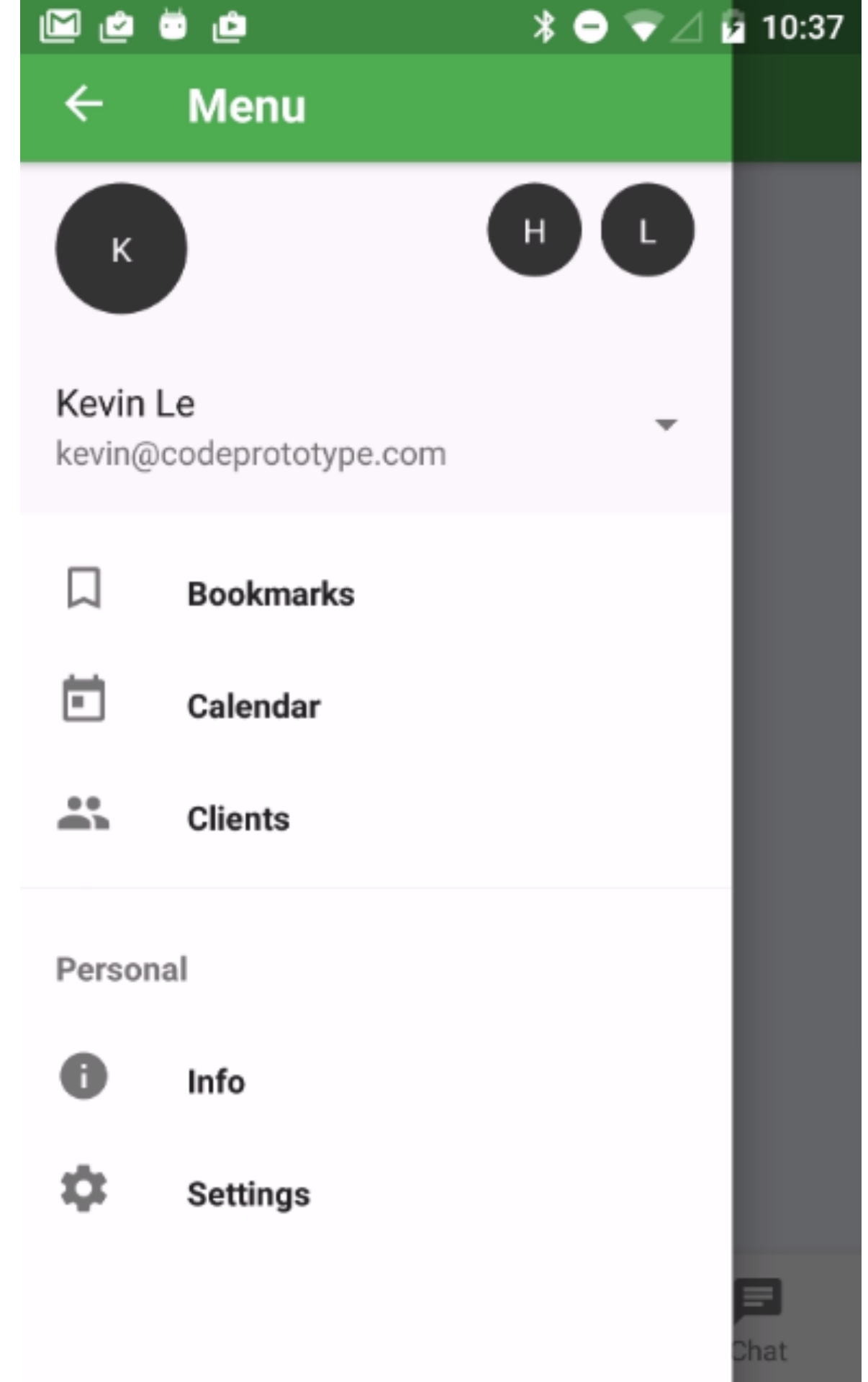


²[Image source](#)

Drawer Navigator³

Definition: Enables tab-like transitions through a hidden drawer that can be exposed and hidden.

Used primarily for options and settings.



³[Image source](#)

The Big Picture

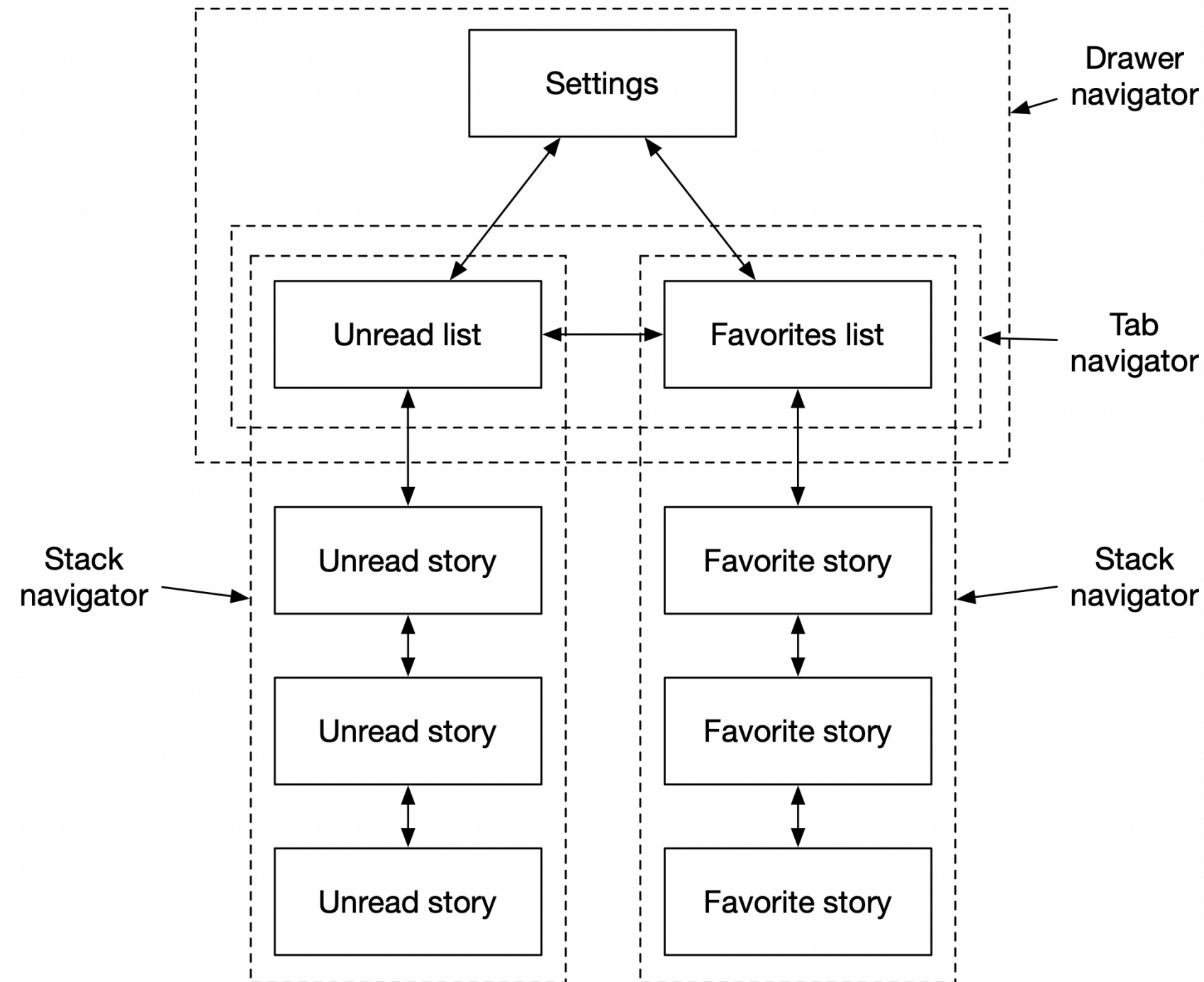
Every RN project will use a combination of these navigators.

As a working example, let's imagine a *news/RSS reader* app with the following specifications:

1. Landing page with *unread* and *favorites* tabs
2. Pages to show unread and favorite stories
3. Settings to change reading mode

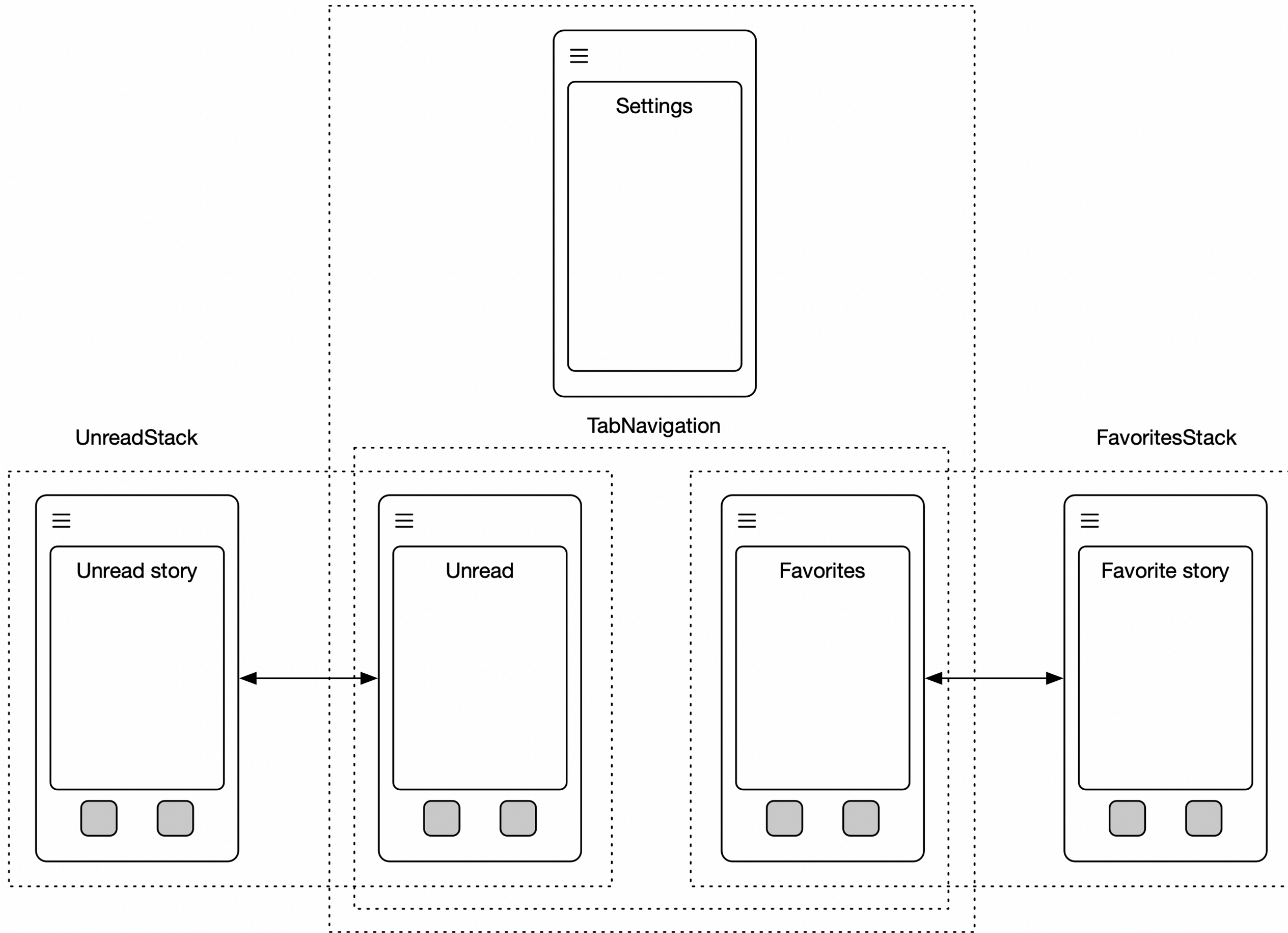
My implementation should include:⁴

1. Tab navigator for the unread and favorites pages
2. Stack navigators for the unread and favorite stories
3. Drawer navigator for the drawer and the tabbed pages



⁴See example on Snack

DrawerNavigation



Screens

Screens can be defined as a React class component with the elements we would like on them.

```
class UnreadScreen extends React.Component {  
  render() {  
    return (  
      <View>  
        <Text>Unread Stories</Text>  
        ...  
      </View>  
    );  
  }  
}
```


Screens can also be defined as functions:

```
function UnreadScreen() {  
  return (  
    <View>  
      <Text>Unread Stories</Text>  
      ...  
    </View>  
  );  
}
```

And placed inside a `NavigationContainer`:

```
const Stack = createStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Unread" component={UnreadScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

UnreadStack & FavoritesStack

To create a stack navigator, we can use `createStackNavigator`:

```
import { createStackNavigator } from '@react-navigation/stack';

const FavoritesStack = createStackNavigator();

function CreateFavoritesStack() {
  return (
    <FavoritesStack.Navigator initialRouteName="Favorites">
      <FavoritesStack.Screen name="Favorites" component={FavoritesScreen}/>
      <FavoritesStack.Screen name="FavoriteStory" component={FavoriteStory}/>
    </FavoritesStack.Navigator>
  );
}
```

Navigators come with a set of screen options:

```
<FavoritesStack.Navigator  
  initialRouteName="Favorites"  
  screenOptions={{  
    gestureEnabled: false,  
    headerTintColor: 'white',  
    headerStyle: { backgroundColor: 'tomato'}  
  }}  
>
```

We can also set options for each screen:

```
<FavoritesStack.Screen
  name="Favorites"
  component={FavoritesScreen}
  options={{
    title: 'Favorite Stories',
  }}
/>
```

TabNavigation

To create a tab navigator, we can use `createBottomTabNavigator`:

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const TabNavigation = createBottomTabNavigator();

function MyTabs() {
  return (
    <TabNavigation.Navigator>
      <TabNavigation.Screen name="Unread" component={CreateUnreadStack} />
      <TabNavigation.Screen name="Favorites" component={CreateFavoritesStack} />
    </TabNavigation.Navigator>
  );
}
```

DrawerNavigation

To create the drawer navigation for settings, we can use `createDrawerNavigator`:

```
import { createDrawerNavigator } from '@react-navigation/drawer';

const DrawerNavigator = createDrawerNavigator();

function CreateDrawerNavigator() {
  return (
    <DrawerNavigator.Navigator initialRouteName="Home">
      <DrawerNavigator.Screen name="Home"
        component={CreateTabNavigationStackNavigator}
        options={{ drawerLabel: 'Home' }} />
      <DrawerNavigator.Screen name="Settings"
        component={CreateSettingsStackNavigator}
        options={{ drawerLabel: 'Settings' }} />
    </DrawerNavigator.Navigator>
  );
}
```

navigation prop⁵

Each screen is automatically provided with a navigation prop (no need to use constructor() for the navigation prop) that provides access to parameters and actions, e.g., navigate, goBack, state.

```
<Button
  style={styles.button}
  color="tomato"
  title="Read"
  onPress={() => this.props.navigation.navigate('FavoriteStory')}
/>
```

⁵[Read more on navigation prop](#)

Navigator actions

Each navigator has a set of specialized actions associated with them that provide low-level access to the navigation behavior of the navigator:

- CommonActions include navigate, reset, goBack, setParams
- StackActions include replace, push, pop, popToTop
- TabActions include jumpTo
- DrawerActions include openDrawer, closeDrawer, toggleDrawer, jumpTo

Mobile Input via Gestures using React Native

Why worry about gesture?

Because of the direct/absolute mapping between input space and the screen space and the touch-sensitive input capabilities, gestures are a resource for mobile development. A number of RN packages provide access to gestures:

- Gesture Responder System
- PanResponder
- React Native Gesture Handler
- React Native Swipe Gestures
- React Native Swipeout

Handling Gestures Using PanResponder

PanResponder uses the core gesture responder system to reconcile several touches into a single gesture that can be used to recognize multi-touch gestures.

To initialize, we create a PanResponder object with event handlers:

```
import { PanResponder } from 'react-native';  
...  
this._panResponder = PanResponder.create({  
  onStartShouldSetPanResponder: (evt, gestureState) => true,  
  onStartShouldSetPanResponderCapture: (evt, gestureState) => true,  
  onMoveShouldSetPanResponder: (evt, gestureState) => true,  
  onMoveShouldSetPanResponderCapture: (evt, gestureState) => true,  
  onPanResponderGrant: (evt, gestureState) => { },  
  onPanResponderMove: (evt, gestureState) => { },  
  onPanResponderTerminationRequest: (evt, gestureState) => true,  
  onPanResponderRelease: (evt, gestureState) => { },  
  onPanResponderTerminate: (evt, gestureState) => { },  
  onShouldBlockNativeResponder: (evt, gestureState) => { return true; },  
});
```

PanResponder Event Handlers

Event handlers utilize `nativeEvent` and `gestureState` objects:

```
onPanResponderMove: (event, gestureState) => {}
```

`nativeEvent` object provides properties such as `locationX` and `locationY` (position of the touch with respect to the element).

`gestureState` object provides properties about the gesture, such as `vx` and `vy` (velocity of the gesture).

`onPanResponderGrant: (evt, gestureState) => { }`

Indicates that the gesture has started. The screen should provide the user with visual feedback on what's happening.

`onPanResponderMove: (evt, gestureState) => { }`

`gestureState` provides access to the most recent move distance (`gestureState.move{x, y}`) and the accumulated gesture distance (`gestureState.d{x, y}`).

`onPanResponderRelease: (evt, gestureState) => { }`

Indicates that the user has released all touches while this view is the responder.

`onPanResponderTerminate: (evt, gestureState) => { }`

Indicates that another component has become the responder, so this gesture should be cancelled.

Associating Gestures with Screens⁶

We provide panHandlers as a prop into the component:

```
<View style={styles.container} {...this._panResponder.panHandlers}>  
  // ...  
</View>
```

⁶ See [example 1](#), [example 2](#) in Snack

Are we done? *No.*

We need to be able to respond to the gestures with appropriate behaviors on the interface, and that's done using animation packages, particularly:

- Animated
- LayoutAnimation

Animated

The Animated library provides the ability to create time-based animation using a number of methods.

```
this.state = { // Create Animated.Value
  fadeValue: new Animated.Value(0) // Connect it to style attributes
};

_start = () => {
  Animated.timing(this.state.fadeValue, { // Animate
    toValue: 1,
    duration: 1000
  }).start();
};
```

The core workflow involves using `Animated.Value`, connecting it to style attributes, and driving it using `Animated.timing()`, but other methods include:⁷

- `Animated.sequence()` allows sequencing several animations.
- `Animated.spring()` animates attributes without a set time in different motion styles, e.g., velocity, bounciness, speed, tension, friction.
- `Animated.interpolate()` maps input ranges to output ranges using linear interpolation.
- Easing functions help in gradual acceleration or deceleration (e.g., easing: `Easing.back()`).

⁷See example in Snack

LayoutAnimation

The `LayoutAnimation` library animates the entire screen when there are changes in the layout, e.g., when an element is removed from the screen.

`LayoutAnimation` is used before `setState()` is called.

`Animated` animates specific components without changing the layout of the screen, while `LayoutAnimation` animates all components on the screen when the layout changes.

```
import { UIManager, LayoutAnimation } from 'react-native';  
...  
<TouchableOpacity  
  onPress={() => {  
    LayoutAnimation.configureNext(LayoutAnimation.Presets.spring);  
    this.setState({expanded: !this.state.expanded});  
  }}>  
  <Text>{this.state.expanded ? 'Expanded text' : 'Collapsed text'}</Text>  
</TouchableOpacity>
```

⁸See example in Snack

Notifications example⁹

⁹See combined example in Snack



Working with Date Objects in JS

Date

The Date object represents a single moment in time in a platform-independent format. We need to use the object in ways that are meaningful both for the server API and for the user.

Users would like to see something like:

Thu Nov 07 2019 11:53:47 GMT-0600 (Central Standard Time)

While the server expects something like:¹⁰

2019-11-07T11:53:47-06:00

¹⁰ ISO 8601 Standard for Date and Time Formats

Good news: We can serialize Date object into the ISO 8601 format.

```
var date = new Date();  
console.log(date); // Thu Nov 07 2019 11:58:58 GMT-0600 (Central Standard Time)
```

```
var json = JSON.stringify(date);  
console.log(json); // "2019-11-07T17:58:58.487Z"
```

Bad news: There is no good method to deserialize back to a date format.

```
var json = "\"2019-11-07T17:58:58.487Z\"";
```

```
var dateStr = JSON.parse(json);  
console.log(dateStr); // 2019-11-07T17:58:58.487Z
```

The trick: We can use the Date constructor for this translation.

```
var json = "\"2019-11-07T17:58:58.487Z\"";  
  
var dateStr = JSON.parse(json);  
console.log(dateStr); // 2019-11-07T17:58:58.487Z  
  
var date = new Date(dateStr);  
console.log(date); // Thu Nov 07 2019 11:58:58 GMT-0600 (Central Standard Time)
```

Assignment Preview

React Native 1 + 2

Design a calorie tracking application, supporting:

- User login/profile
- Exercise and Planning
- Recipes and Foods

React Native 1 α

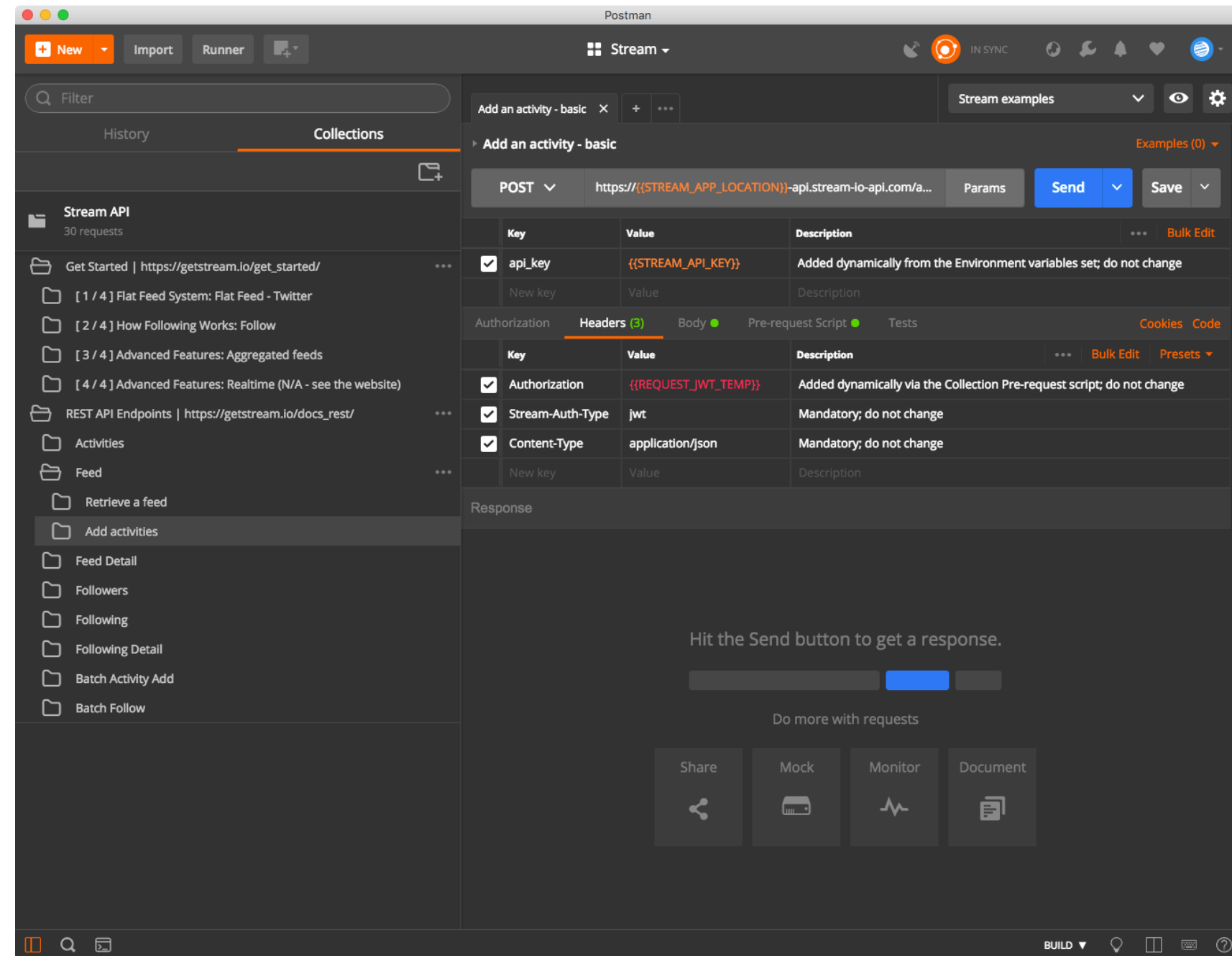
- Create a “Login” view with username and password input fields
- Create a “Create User” view with username and password fields
- Create a profile view that allows the logged-in user to view and edit their name and goals
- Clean and clear code/interface

The following API can be accessed at <https://cs571.cs.wisc.edu:5000>.

Route	Auth Required	Token Required	Get	Post	Put	Delete
/login	✓		✓			
/users				✓		
/users/ <username>		✓	✓	✓	✓	✓
/meals		✓	✓	✓		
/meals/ <meal_id>		✓	✓		✓	✓
/meals/ <meal_id> /foods		✓	✓	✓		
/meals/ <meal_id> /foods/ <food_id>		✓	✓		✓	✓
/activities		✓	✓	✓		
/activities/ <activity_id>		✓	✓		✓	✓
/foods			✓			
/foods/ food_id			✓			

A Few Tips

- The passwords may not be secure, so do not use a password you use for other accounts.
- Do not create too many accounts.
- Postman demo...



What did we learn today?

- Mobile Navigation using React Native
- Mobile Input via Gestures using React Native
- Working with Date object in JS
- Assignment Preview