Building User Interfaces

React Native 1

Introductory Concepts

Professor Yuhang Zhao

Midterm

— Main session: October 28th 7:15 pm CST, Van Vleck B102

— Alternative session: October 29th 7:15 pm CST, Psych 103

— 90 minutes; please come to the exam location 15 minutes ahead

— Topic covered: Web Module, including both build (javascript, React) and design - everything before this week.

— Build (44 points), Design (40 points), Intergrate (16 points)

Monday

— 1:00-3:00 PM: Brandon

— 4:30-6:30 PM: Yuren

Tuesday

— 1:30-3:30 PM: Ilkyu

— 3:30-5:30 PM: Salman

Wednesday

— 10:00-12:00 PM: Sujitha

— 2:00-4:00 PM: Brandon

— 5:30-7:30 PM: Ilkyu

Thursday:

— 2:30-4:30 PM: Salman

— 5:00-6:00 PM: Prof. Yuhang Zhao

Friday:

— 10:00-12:00 PM: Sujitha

— 3:00-5:00 PM: Yuren

What we will learn today?

— What is React Native?

— Starting a React Native project

— Differences between ReactJS and React Native

— Communicating with Server APIs

What is React Native?

What is React Native?

Definition: A JS *framework* for building native, cross-platform mobile applications using React, developed by Facebook in 2015.

Unlike ReactJS, which was a library, React Native is a framework that includes everything that we will need to build mobile applications.

React Native supports **iOS** and **Android** development.

Enables web developers using ReactJS to easily develop mobile applications.

How does React Native work?[1]

As in ReactJS, React Native combines JS, JSX.

Under the hood, React Native runs the program in the JS engine of the host platform (iOS, Android, etc.), which renders and interacts with native UI components, instead of web pages, through a "bridge."
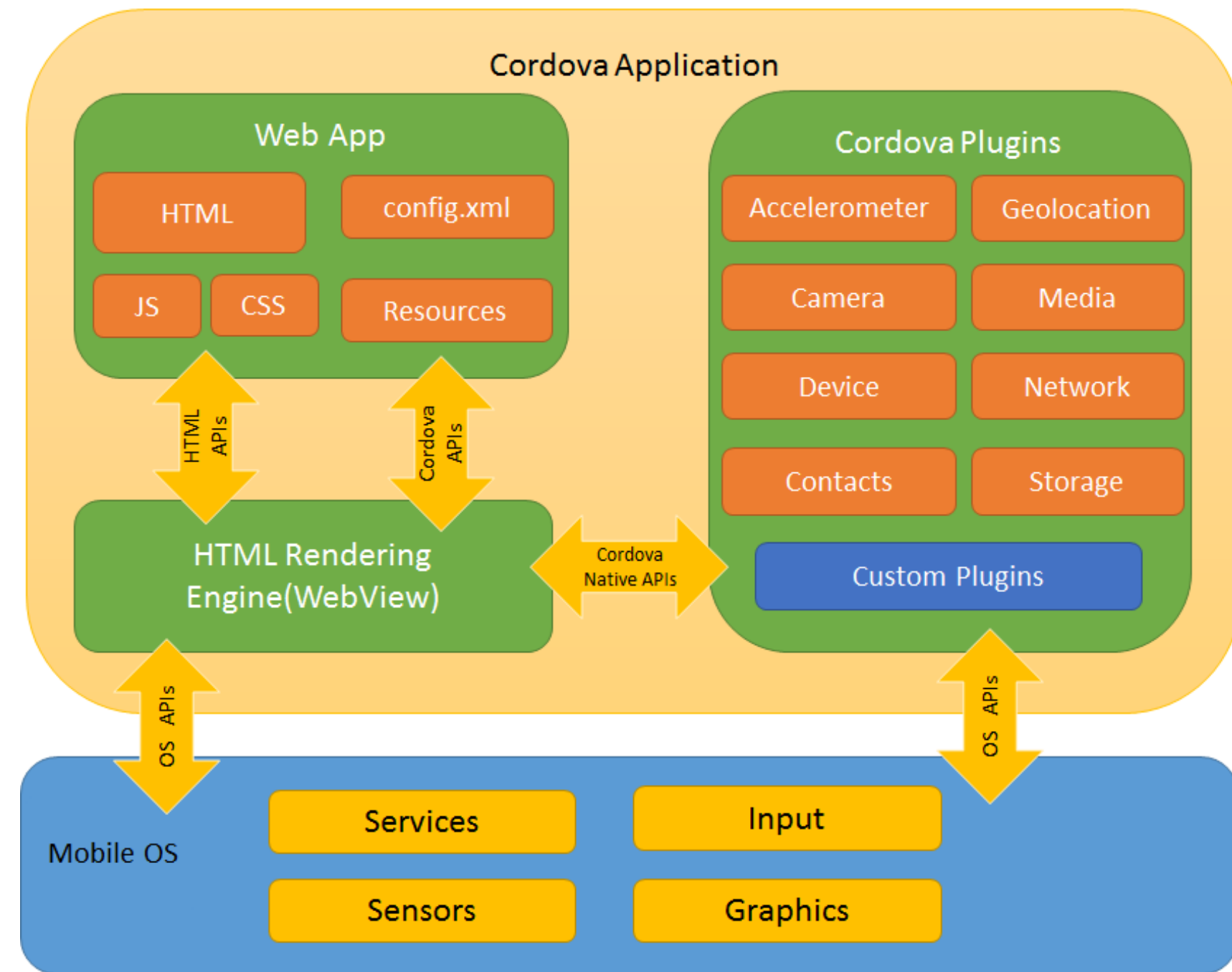
[1] Image source
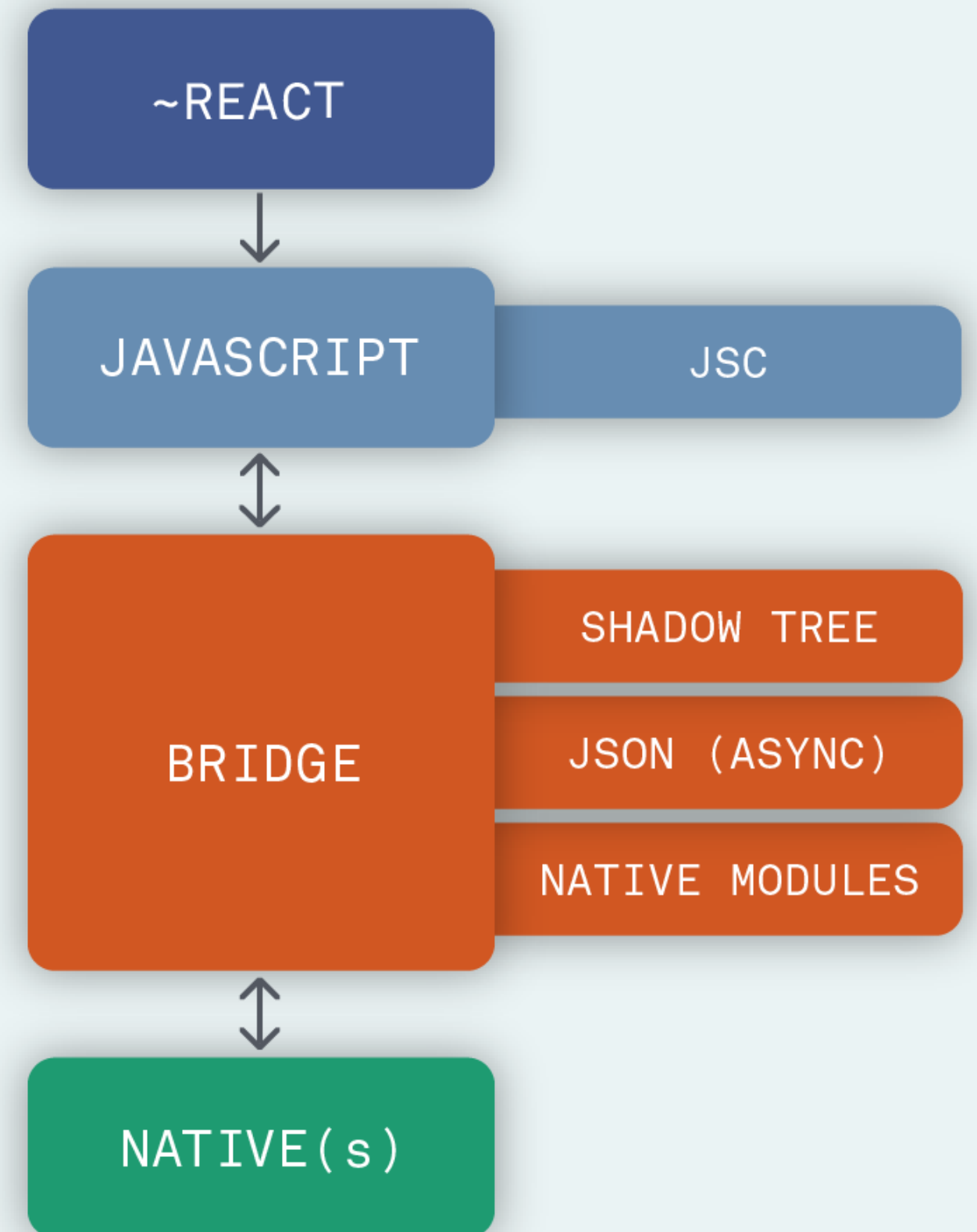
Alternatives to React Native[2]

All cross-platform alternatives to React Native will wrap HTML + CSS + JS within a web view and simulate mobile UI behavior.

Examples:

— <u>Ionic</u>

— <u>Cordova</u>

— <u>Titanium</u>

Starting a React Native project

Two Methods Using Expo[17]

Node.js

To install Expo:

```
npm install expo-cli --global
```

Create and run a new project:

```
expo init my-new-project
cd my-new-project
expo start
```
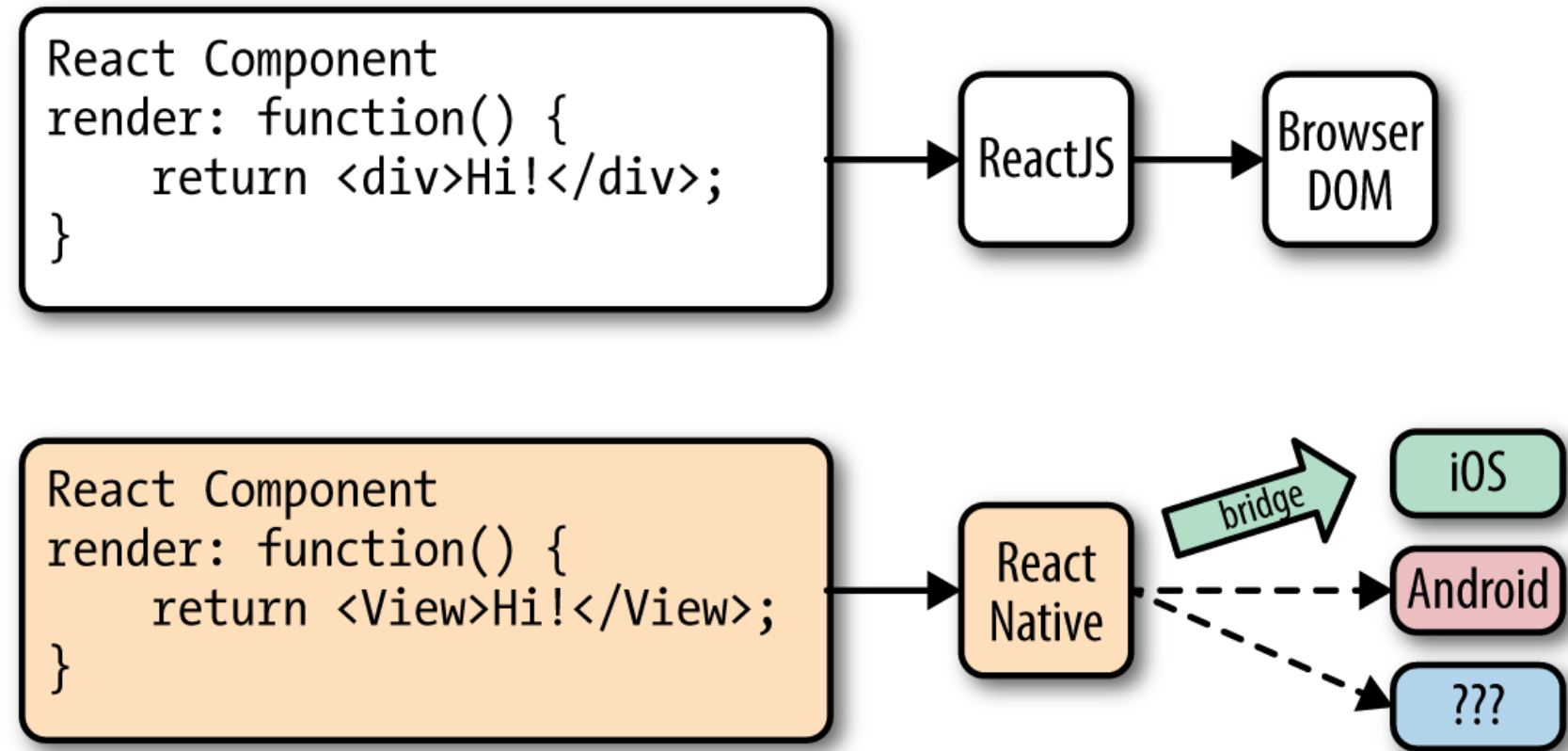
Snacks

Use the Expo web editor.

---

[17] Get started with Expo

How is React Native different from ReactJS?

React Native-ReactJS differences[3]

Good news: They are very, very similar.

Minor differences stem from how React Native interacts with the native mobile platform.

```
React Component
render: function() {
    return <div>Hi!</div>;
}
```
ReactJS → Browser DOM

```
React Component
render: function() {
    return <View>Hi!</View>;
}
```
React Native → bridge → iOS, Android, ???

[3] Image source

Instead of the HTML DOM, React Native interacts with native components through its Bridge.

Instead of React elements that will become DOM elements, React Native uses elements that are similar but better correspond with native components.

Key difference #1: Core Components[4]

Instead of `div`, React Native uses `View`.

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

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

[4] See code in Snack

Similarly, instead of `p`, RN uses `Text`; instead of `img`, it uses `Image`.[5]

```
export default class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Image style={styles.logo} source={require('assets/uw-logo-centered-web.png')} />
        <Text style={styles.paragraph}>
          On Wisconsin!
        </Text>
      </View>
    );
  }
}
```

[5] See code in Snack

Additionally, the button clicks trigger an `onPress` event, instead of an `onClick` event.[6]

In ReactJS:

```
<Button onClick={this.updateCounter}>Press me</Button>
```

In React Native:

```
<Button title="Press me" onPress={this.handlePress} />
```

[6] See code in Snack

Key difference #2: Styling[7]

Because RN does not use web elements, we can't use CSS styles. We instead create stylesheets in JS.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 40,
  },
    ...
});
```

[7] See code in Snack

Pro Tip: Style definitions can be done in stylesheets or in props when components are instantiated. You can also combine both methods.[8]

```
<View>
    <Text style={styles.label}>First label</Text>
    <Text style={{fontSize: 28, color:'tomato'}}>Second label</Text>
    <Text style={[styles.label, {fontSize: 20, color:'gray'}]}>Third label</Text>
</View>
```

[8] See code in Snack

Flexbox is Everything[9] [10]

Flexbox is the only standard way of creating layouts in RN, so we have to master it.

```
<View style={{flex: 1, flexDirection: 'column'}}>
    <View style={{flex: 1, backgroundColor: 'whitesmoke'}}/>
    <View style={{flex: 1, backgroundColor: 'gainsboro'}}/>
    <View style={{flex: 1, backgroundColor: 'silver'}}/>
</View>
```

[9] Visual Flexbox Cheatsheet

[10] React Native Guide to Layouts with Flexbox

Below are the commonly used properties:[11]

`flex: 1` will express how much of the container to fill.

`flexDirection` — `row, column, row-reverse, column-reverse`

`alignItems` — `stretch, flex-start, flex-end, center, baseline`

`justifyContent` — `flex-start, flex-end, center, space-between, space-around, space-evenly`

[11] See code in Snack

Getting Screen Dimension[12]

Mobile devices vary significantly in screen size, and we often need to obtain screen dimensions of the device using the `Dimensions` class in `react-native`.

```
getScreenSize = () => {
    const screenWidth = Math.round(Dimensions.get('window').width);
    const screenHeight = Math.round(Dimensions.get('window').height);
    this.setState({ screenWidth: screenWidth, screenHeight: screenHeight })
}
```

[12] See code in Snack

Key difference #3: Platform-specific Components

RN provides a number of components that utilize platform capabilities that may not be available in other platforms, thus for cross-platform development, we need to utilize multiple platform-specific components.

E.g., `TouchableNativeFeedback` only work on Android, and the same effect can be achieved using `TouchableHighlight`.

First method: Selectively render the component based on the current platform.[13]

```
render() {
    if (Platform.OS === 'android') {
        return (
            <TouchableNativeFeedback> ... </TouchableNativeFeedback>
        )
    } else {
        return (
            <TouchableHighlight> ... </TouchableHighlight>
        )
    }
}
```

[13] See code in Snack

Second method: We create two versions of the component, e.g.,
`MyButton.ios.js` and `MyButton.android.js`.

```
import MyButton from '/components/MyButton';
render() {
    <MyButton />
}
```

Key difference #4: Animation & Gestures

Because of the resource-constrained nature of mobile platforms, applications heavily rely on animation and gestures. In RN, CSS animations are not available, but there are several powerful packages:

— Animated API (for animating components)

— LayoutAnimation (for layout animation)

— PanResponder (for gestures)

More on this next class!

Key difference #5: Navigation

Most mobile applications include several *screens*. Using `react-navigation`, we can create several screens and define navigation.

```javascript
import {createAppContainer} from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack';

const MainNavigator = createStackNavigator({
  Home: {screen: HomeScreen},
  Profile: {screen: ProfileScreen},
});

const App = createAppContainer(MainNavigator);
export default App;
```

More on this next class!

Communicating with Server APIs

Almost all mobile applications offer personalized information and thus have to know who the user is through *authentication.*

Definition: Authentication identifying a user in the process of providing access to system data or services based on the user's identity.

Common Authentication Methods

— Basic HTTP authentication
Requires username/password for every request

— Session-based authentication
Client receives a session ID after authentication, stores it in a cookie, and attaches it to every subsequent request

— JWT-based (JSON Web Tokens) authentication
Client sends encrypted user information and receives a token, which is included in every request and decrypted by the server

In this module, we will use *JWT-based authentication.*

JWT-based Authentication[14]

The client authenticates with a *username* and a *password* once, receives a token, and only sends the *token* for subsequent requests, until the token times out — works like an all inclusive resort!

[14] Image source

Definition: REpresentational State Transfer (REST) is an architectural style for distributed hypermedia systems.

Any information, e.g., an image, can be a *resource*, the key abstraction of REST.

REST uses *resource methods*, e.g., HTTP methods, to facilitate client-server interaction.

[15] More on RESTful APIs

RESTful API Methods

In this module, we will focus on four methods:

— GET

— POST

— PUT

— DELETE

Definition: HTTP method to retrieve information from the server in a way that does not change the server (a *safe* method).

GET should be *idempotent*, returning the same information every time it is called, until another request (POST, PUT) changes the resource.

The server will return either 200 (OK) along with the data (e.g., JSON) or 404 (NOT FOUND).

HTTP GET http://<our-domain>/users

Definition: HTTP method that creates new subordinate resources, e.g., a new user in a collection of users. `POST` is *not* safe or idempotent.

The server will usually return either `201 (Created)` along with information on the new resource and a location header. The server can also return `200 (OK)` or `204 (No Content)`.

```
HTTP POST http://<our-domain>/users
```

Definition: HTTP method to update existing information on the server.

The server will return `200 (OK)` or `204 (No Content)`. If the information does not exist, the API may create the resource, as done in a `POST` request, and return `201 (Created)`.

```
HTTP PUT http://<our-domain>/users/<username>
```

DELETE

Definition: HTTP method that deletes resources from the server.

`DELETE` is *idempotent*, as calling `DELETE` several times does not change the outcome.

The server will return `code 200 (OK)` if the response includes an entity with status, `202 (Accepted)` if the request is queued, or `204 (No Content)` if it is performed but an entity is not included.

`HTTP DELETE http://<our-domain>/users/<username>`

Resource Methods in React Native

For all methods, we can use `fetch()`. React Native `GET` example:

```
fetch('<our-domain>/endpoint')
    .then(function(response) {
        return response.json()
})

fetch('<our-domain>/endpoint', {
  method: 'GET',
})
```

React Native `POST` example:

```
fetch('<our-domain>/endpoint', {
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    userName: '<user-name>',
    password: '<password>',
  })
})
```

React Native Get example:

```
fetch('<our-domain>/endpoint', {
    method: 'GET',
    headers: {
      'Authorization': 'Basic ' + base64.encode(this.state.username + ":" + this.state.password)
    }
  .then(res => res.json())
  .then(res => {
      if (res.token) {
        this.props.login(this.state.username, res.token);
      } else {
        alert("Incorrect username or password! Please try again.");
      }
    });
  })
```

When we pass the userName and password and receive a token, we need to encode what we are passing. NPM `base-64` package accomplishes that.

```
import base64 from 'base-64';

base64.encode(username + ":" + password);
```

Passing Authentication Information

Example header to pass user credentials:

```
'Authorization', 'Basic ' + base64.encode(username + ":" + password)
```

Example header to pass token:

```
'x-access-token', result.token
```

React Native PUT example:

```
fetch('<our-domain>/endpoint', {
    method: 'PUT',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json',
    'x-access-token': <my-token>
  },
  body: JSON.stringify({
        location: 'Madison, WI'
  })
});
```

React Native `DELETE` example:

```
fetch('<our-domain>/endpoint', {
    method: 'DELETE',
  headers: {
    'x-access-token': <my-token>
  },
});
```

Endpoints

Different resource methods will use different *endpoints* at the server. For example, in React Native 1 α, we will use the following endpoints for `https://cs571.cs.wisc.edu:5000`:

`/login` — allows `GET` to receive the token

`/users` — allows `POST`

`/users/<user-name>` — allows `GET`, `POST`, `PUT`, `DELETE`

More on React Native 1 α next class!

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`			✓			

Server API Tools[16]

Easy to use tools help setting up server APIs for user authentication, e.g., Postman.

More on how this next class!
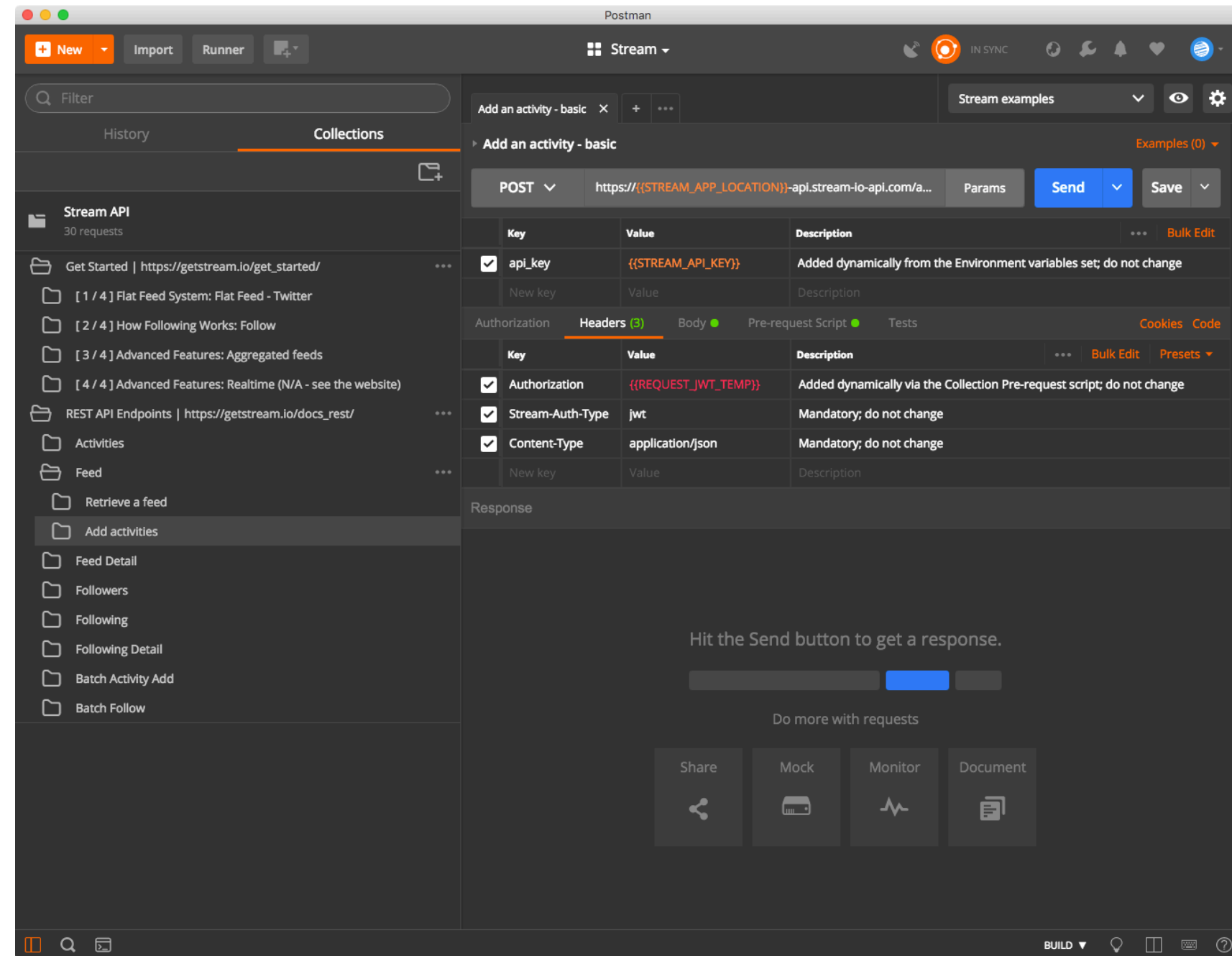
[16] Image source

What did we learn today?

— What is React Native?

— Starting a React Native project

— Differences between ReactJS and React Native

— Communicating with Server APIs