Week 1 Workshop

React Native Course

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes
Introductions & Check-In	10 minutes
Week Recap	40 minutes
Task 1	60 minutes
BREAK	15 minutes
Tasks 2 & 3	90 minutes
Check-Out	15 minutes

Congratulations once again

- To get here, you passed the React course, well done!
- You learned a lot about ES6 JavaScript as well as React.
- That means you've done a lot of the hard work already, and in the React Native course, you will reap the benefits of that, as it will be easier to learn React Native than it was to learn React.

Introductions & Check-In

- Instructor introduction (and Instructor Lead, if present)
- Student introductions – students should know each other already, but please introduce yourself briefly to your instructor if you have not met.
- Check-In:
  - How was this week? Any particular challenges or accomplishments?
  - Did you understand the Exercises and were you able to complete them?
  - You must complete all Exercises before beginning the Workshop Assignment.

Welcome to React Native!

Week 1 Recap: New Concepts This Week

- Native vs Web vs Multiplatform vs Hybrid Mobile Apps
- Using the Expo CLI
- Installing and Using an Android Virtual Device (AVD)
- React Native Components:
  - <View> <Text> <FlatList> <ScrollView>

- React Native Elements Components:
  - <ListItem> <Card>
- React Navigation
  - Stack Navigator, Drawer Navigator
  - Navigation prop

Next slides will review these concepts

Types of Mobile Apps

- Native mobile apps: created with specific programming language for specific OS - Java/Kotlin for Android; Swift/Objective C for iOS. Native apps are standalone and do not run in a browser, can be downloaded and installed directly to mobile device.
- Mobile-first web apps: another approach to developing for mobile, these apps are run through the browser using Web technologies, not standalone. This category includes Progressive Web Apps (PWAs).
- Cross-platform/multiplatform mobile apps: make it possible to use single language & codebase to develop for multiple platforms, e.g. React Native lets you write JavaScript codebase for both iOS and Android.
- Hybrid apps: subset of cross-platform apps that use web technologies combined with standalone mobile apps, e.g. WebView & TypeScript with Ionic/Cordova and JavaScript with React Native/NativeScript.
- React Native is a cross-platform and hybrid approach.

Types of Mobile Apps (cont)

- Native mobile apps: Highest access to device capabilities and typically fastest. Requires separate codebase for each platform.
- Mobile-first web apps: More limited in access to full device capabilities, slower – but improving all the time. Single codebase.
- Cross-platform/multiplatform mobile apps (including hybrid): Typically between native and web apps in terms of access and speed, also improving all the time. Single codebase.

Expo CLI

- Expo CLI is used to scaffold out a starter site for React Native, like **create-react-app** for React
- Third party from React Native but is officially recommended by React Native for developers to use when starting out – you may not need it later at a more advanced stage
- Expo also offers other tools, mainly Expo Snack (RN code playground) and Expo SDK – we will learn more about Expo SDK later in this course

Android Emulator

- Android emulator (Android Virtual Device/AVD) is used in this course to present consistent learning environment, regardless of whether you use MacOS or Windows
- If you were unable for some reason to get your emulator working, then try installing the Expo client on your phone and loading your app through that client
  - You can find instructions for this in the learning portal at the bottom of Exercise: Setting up the Android Emulator
- Please let your instructor know if neither of these methods worked for you!

Built-In React Native Components

- <View>
  - Most fundamental component for building React Native UI
  - Container for other components, analogous to HTML or
- <Text> - a component that simply holds some text
- <FlatList>
  - renders lists from array
  - uses lazy loading – only visible items are loaded

React UI Elements Components

- <ListItem>
  - use to display rows of information inside a container component such as View, FlatList, ScrollView
- <Card>
  - typically used to display information about a single subject

FlatList (React Native) & ListItem (RNE)

```
function Directory(props) {

    const renderDirectoryItem = ({item}) => {
        return (
            <ListItem
                title={item.name}
                subtitle={item.description}
                onPress={() => props.onPress(item.id)}
                leftAvatar={{ source: require('./images/react-lake.jpg')}}
            />
        );
    };

    return (
        <FlatList
            data={props.campsites}
            renderItem={renderDirectoryItem}
            keyExtractor={item => item.id.toString()}
        />
    );
```

Directory component:

- View rendered by returning `<FlatList>`
  - Iterates through `data` array and sends each array item (in this case, a campsite object) to function named in `renderItem` prop (in this case, `renderDirectoryItem`)
  - Uses `keyExtractor` prop to set unique key needed by React for lists generated from arrays, using each object's `id` property.
- `renderDirectoryItem` function is passed each object from the campsites array in turn, returns a `<ListItem>` for that object
- `<ListItem>` also has onPress prop which takes a function to run when ListItem component is pressed

FlatList & ListItem continued

- Discuss together:
  - From what component's state is the `props.campsites` array used to render the FlatList in the Directory component passed?
  - What happens when the `onPress` for a ListItem in the Directory component gets triggered? Can you trace through what happens together, step by step? Talk through it as a class first, then view following slide and make sure you hit all the points (or more!)

ListItem onPress (answer to discussion question)

- When a campsite rendered with the `ListItem` component in the `Directory` component is pressed:
  - The `onPress` prop passed from the `Main` component contains the `onCampsiteSelect` method – this will run, passing the campsite object's `id` property into its parameter list
  - The `onCampsiteSelect` method will update the `Main` component's `this.state.selectedCampsite` value with that id
  - This will change what is rendered by the `CampsiteInfo` component beneath the `Directory` component in the `Main` component – the `CampsiteInfo` component will filter for that id from the campsites array

React Navigation: App Containers

- Always wrap the top-level navigator in an App Container vy using the createAppContainer function, as you did in MainComponent.js this week:

```
const AppNavigator = createAppContainer(DirectoryNavigator);
```

- Then use the App Container that you created to render your app, e.g.:

```
class Main extends Component {
    render() {
        return (
            <View>
                <AppNavigator />
            </View>
        );
    }
}
```

React Navigation - Navigators

The React Navigation library, despite the misleading name, is for React Native navigation and not React.

It supports these common mobile application navigation patterns:

- Drawer Navigator
- Stack Navigator
- Tab Navigator
- Switch Navigator
- Custom Navigator (write your own)

We will be using **Drawer, Stack,** and **Tab** Navigators in this course

Stack Navigators

- Multiple Stack Navigators in our application, each creating its own Stack of screens
- LIFO behavior – Last In, First Out - like a stack of dishes, each new screen you navigate to in a stack is placed on top, then removed if you use Back functionality (hardware or software)
- Link components to a Stack Navigator by listing them as a `screen` in the first argument for `createStackNavigator` function
- Use static property `navigationOptions` to specify `title` within a particular component – this will show in the Stack Navigator header
- Each component that is used as a `screen` is automatically provided with the `navigation` prop, which contains various convenience functions including:
    - `navigate()` function – you used it in `Directory` component to navigate to selected campsite with `onPress`
    - `getParam()` function – you used it in `CampsiteInfo` to access selected campsite ID.

Navigation Prop

- Each component that is used as a `screen` is automatically provided with the `navigation` prop, which contains various convenience functions including:
  - `navigate()` function – you used it in Directory component to navigate to selected campsite with onPress
  - `getParam()` function – you used it in CampsiteInfo to access selected campsite ID, as shown below:

```
render() {
    const campsiteId = this.props.navigation.getParam('campsiteId');
```

  – Discuss together: Where and how was the 'campsiteId' param set? (answer in next slide)

Navigation Prop (cont)

- Answer to discussion question: The 'campsiteId' param was set in the Directory component via the navigation prop's `navigate()` function:

```
onPress={() => navigate('CampsiteInfo', { campsiteId: item.id })}
```

- The `navigate()` function takes an optional second argument of an object with parameters, can have multiple parameters
- It also has two other (advanced) optional arguments of *action* and *key* - consult the documentation for more information (https://reactnavigation.org/docs/4.x/stack-navigator)

Drawer Navigator

- We implemented a single side drawer navigator named `MainNavigator`
- Added two screens, using the Stack Navigator components we created: the `HomeNavigator` (which wraps the Home component) and the `DirectoryNavigator` (which wraps both the Directory and CampsiteInfo components)
- Rendering the `MainNavigator` Drawer Navigator from the `Main` component now lets us access and navigate to all our component screens from the side drawer

ScrollView (React Native component)

- Container component that enables scrolling
- Discuss together:
  - When might you use `<ScrollView>` instead of `<FlatList>`?
  - What are some differences between the two?

ScrollView (React Native component)

- Potential answers:
- FlatList uses lazy loading (renders only the child components that are on the screen or about to be) and ScrollView renders all its child components at once
- FlatList is better for very long lists of items where each item is rendered in a similar way, ScrollView is better for a smaller number of items
- FlatList always renders using an array; with ScrollView, you place what you want to render inside it like you would with a View component
- FlatList provides many out-of-the-box features such as rendering separators between items, multiple columns, etc – ScrollView is simpler

Workshop Assignment

- It's time to start the workshop assignment!
- Sit near your workshop partner.
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - 10-minute rule does *not* apply to talking to your partner, you should consult each other throughout.
- Follow the workshop instructions very closely
  - both the video and written instructions.
- Talk to your instructor if any of the instructions are unclear to you.

Check-Out

- Submit your 3 assignment files in the learning portal:
  - *ContactComponent.js*
  - *MainComponent.js*
  - *AboutComponent.js*
- Wrap up – Retrospective
  - What went well
  - What could improve
  - Action items
- Start Week 2 or work on your Portfolio Project