Lesson:

App Structure for the Hello World Example

PW SKILLS

In the Hello World example of a React Native project, the app structure consists of several files that serve specific purposes. Understanding the role of each file will help you grasp the overall structure and organization of a React Native project. Let's dive into the purpose of each file in the Hello World example:

1. App.js

App.js is the main entry point of your React Native application. It serves as the starting point of the app and contains the root component that will be rendered on the screen. In the Hello World example, App.js defines the App component, which is a functional component responsible for rendering the "Hello, World!" message. It imports necessary components from the react-native package and applies styles to the components. This file is where you write most of your application's logic and components.

```
// App.js

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

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Hello, World!</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  text: {
    fontSize: 24,
    fontWeight: 'bold',
  },
});

export default App;
```

2. package.json

package.json is a configuration file that contains metadata about your project and its dependencies. It provides information about the project's name, version, dependencies, scripts, and more. It is automatically generated when you initialize a React Native project using tools like the React Native CLI. It also allows you to manage and install project dependencies using npm (Node Package Manager).

```
{
  "name": "HelloWorldApp",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "start": "react-native start",
    "android": "react-native run-android",
    "ios": "react-native run-ios",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "react": "16.13.1",
    "react-native": "0.63.4"
  },
  "devDependencies": {
    "@babel/core": "7.12.9",
    "@babel/runtime": "7.12.5",
    "@react-native-community/eslint-config": "2.0.0",
    "babel-jest": "26.6.3",
    "eslint": "7.14.0",
    "jest": "26.6.3",
    "metro-react-native-babel-preset": "0.63.0",
    "react-test-renderer": "16.13.1"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

3. node_modules/

The `node_modules/` directory contains all the external dependencies and packages that your React Native project requires. When you initialize a React Native project or install new packages using npm, the dependencies are downloaded and stored in this directory. It is recommended not to modify the files in this directory manually, as they are managed by npm.

4. app.json/

In an Expo project, the app.json file is a configuration file that is used to define various settings and metadata for your Expo app. Expo is a framework and platform for building React Native applications, and the app.json file helps you customize and configure your app's behavior, appearance, and other settings.

Here are some of the key uses of the app.json file in an Expo project:

1. **App Configuration:** You can use app.json to configure various aspects of your app, such as its name, description, version number, and icon. This information is used when your app is published to app stores.
2. **Environment Configuration:** You can define environment-specific configuration variables that your app might need. This is particularly useful when you have different settings for development, staging, and production environments.
3. **Permissions and Services:** You can specify which device permissions your app requires and enable various services, such as push notifications, location services, and more.
4. **App Entry Point:** You can specify the main entry point of your app. This is the file that will be executed when your app is launched.
5. **Splash Screen Configuration:** You can configure the appearance and behavior of the splash screen that is displayed when your app starts.
6. **Orientation and Display Configuration:** You can control the orientation and other display-related settings for your app.
7. **Android and iOS Configuration:** You can define platform-specific settings, such as icons, splash screens, and permissions, for both Android and iOS.
8. **Expo Modules Configuration:** You can configure specific Expo modules that your app uses, such as push notification configuration, Google Maps API keys, and other third-party integrations.

Here is an example of a basic app.json file:

```
{
  "expo": {
    "name": "My Expo App",
    "slug": "my-expo-app",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
```

```
    "android": {
      "package": "com.myexpoproject"
    },
    "ios": {
      "bundleIdentifier": "com.myexpoproject"
    }
  }
}
```

Remember that the specific configuration options available in app.json can change over time as new features and improvements are introduced to the Expo framework. Always refer to the official Expo documentation for the most up-to-date information on configuring your Expo app using the app.json file.

5. assets/

In an Expo project, the assets folder is used to store various static assets that your app needs, such as images, fonts, videos, audio files, and other resources. These assets are then accessible and can be used within your app's code and components. The assets folder is an important part of organizing and managing the visual and auditory elements of your app.

Here's why the assets folder is important and how it is used:

1. **Images:** You can store image files (e.g., PNG, JPEG) in the assets folder, and then reference and display these images in your app's UI components. This is useful for icons, illustrations, logos, background images, and any other visual elements your app may need.
2. **Fonts:** If your app uses custom fonts, you can place the font files (e.g., TTF, OTF) in the assets folder and configure your app to use them. This ensures consistent typography across different devices and platforms.
3. **Videos and Audio:** If your app includes videos or audio files, you can store them in the assets folder. This allows you to easily include and play these media files within your app.
4. **Splash Screens:** If you're customizing your app's splash screen (the initial screen that appears when the app is launched), you can place the splash screen image in the assets folder and reference it in your app.json configuration.
5. **Static Files:** Any other static files that your app requires, such as JSON files, XML files, or other data files, can be stored in the assets folder.

Here's an example of the structure you might find within the assets folder of an Expo project:

```
assets/
  ├── images/
  │   ├── logo.png
  │   ├── icon.png
  │   └── background.jpg
  ├── fonts/
  │   ├── OpenSans-Regular.ttf
  │   ├── Roboto-Bold.ttf
  │   └── ...
  ├── videos/
  │   ├── intro.mp4
  │   └── ...
  └── audio/
      ├── bg_music.mp3
      └── ...
```

To use assets from the assets folder in your app, you would typically import them into your components and use them in your code. Expo provides tools and utilities to help you manage and load assets efficiently, ensuring that they are bundled with your app and accessible at runtime.

For example, you might use code like this to load an image from the assets folder and display it in your app:

```jsx
import React from 'react';
import { Image } from 'react-native';

const App = () => {
  return (
    <Image source={require('./assets/images/logo.png')} />
  );
};

export default App;
```

Conclusion

Understanding the purpose of each file in a React Native project, as demonstrated in the Hello World example, gives you a solid foundation for organizing and working with your React Native applications. The App.js file serves as the entry point and contains your app's logic, while index.js bootstraps the app. package.json manages dependencies. With this understanding, you can confidently navigate and explore more complex React Native projects as you progress in your learning journey.