Basic Information and Introduction

Code Sharing Information and High level plan

- Rendering code can be shared, but require a significant amount of refactoring and consolidation of both front-ends into a single repo.
    - React is a javascript library and renders with DOM
    - React Native is a framework and renders with native API
- Logic code can be shared more easily
    - React.js is the heart of React Native
    - Since syntax is shared, most logic code can be shared
    - Potential shared files:
        - fetch.js
        - Config.js

Suggestions and future planning to allow better code sharing:
This will be separated in to 2 main sections, hard code sharing and soft code sharing.
In the hard code sharing section, the suggestions will be focused around creating an environment where the exact same code can be shared directly between the two projects without any modifications. Code that is sharable in this manner will mostly be API-related functions, useful hooks that can be used in both apps, global configuration such as addresses and names, and other forms of logic code that do not directly interact with rendering.

The soft code sharing section, on the other hand, will focus on the sharing of code that follow similar structures or serve similar purposes, but cannot be shared as-is between the two projects. This will mostly be rendering related code.

1. Hard Code Sharing

As mentioned earlier, some major project refactoring will be needed in order for the web app and react native app to directly share codebases.

One of the most important changes would be to combine the react native and react apps into a single root directory/repository. This should preferably be done in a way that keeps both apps independent, but with a pool of shared code that can be accessed by both apps.

Below are some suggestions for modifying the project to create a more efficient environment for sharing logic, rendering, and other code:

Project Directory Structure

The structure of the project is an important part of efficient code reuse. It is beneficial to separate files in a way that reflects where they are used, and shared code needs to be accessible by both systems.

For example:

Indras_net
|
|_ **Controllers**
| |_ **Axios**
| |_ **Hooks**
| |_ SharedConfig.js
| |_ …
|
|_ **webapp**
| |_ **Components**
| |_ app.js
| |_ …
|
|_ **ReactNative**
| |_ **Components**
| |_ app.js
| |_ …
|
|_ …

In this example, the controllers directory would contain the sharable logic and react components that are used in both the webapp and the react native app. Both projects will likely have to be modified to have much less hard-coded logic code and rely more on using hooks and functions from the shared folder.

For example, code like this appear in menu components that interact with the back-end API:

```
try{
      const properties = await axios.get(`${apiServer}${menuID});
}catch(e){
```

```
        //...
}
```

A good start would be to create a more general API-related functions script inside controllers that is agnostic to which platform is calling it, so that it can be transformed into something more like:

```
const properties = await fetch.ApiGet(`${menuID}`);
```

State properties can be changed to be platform agnostic as well, but it is a far more involved process that involves the usage of react hooks. A good example of something that can be a hook would be interactions with the back-end model API, allowing the logic of the interface for model interaction to be platform agnostic as well.
The documentation for hooks can be found here:

- https://reactjs.org/docs/hooks-overview.html
- https://reactjs.org/docs/hooks-custom.html

2. Soft Code Sharing (sharing rendering code)
On the other hand is the more practical and simple code sharing that can happen between the web and react native apps. The most important difference is that rendering code cannot technically be shared between the two platforms, however, styling and structure of components can be similar enough that some minor tweaking would allow efficient retooling of components from one platform to the other.

The main problem in sharing rendering code is, naturally, that react components render using html tags, while react-native use a similar markup language, but not HTML tags directly.
There are two ways to go about sharing rendering code between the two platforms:

With minimal changes to current code:
The way to “share” code between the two platforms having to significantly refactor the current codebase would be to manually make changes to scripts/components. While this may barely seems like sharing code at all, having the current structure and platform-specific styling should allow for the HTML tags to be almost directly translated to react native components without having to make significant changes to the rendering structure of each component.
With code-sharing design in mind, a large portion of components should be able to be shared with great efficiency:
For a simple example, a simple component like this can be translated easily since props and other logic-related code can be shared directly:

```
<div>
        <p>Hello! This is a React Component</p>
        <p>{this.state.testProp}</p>

</div>
```

to:

```
<>
        <Text>Hello! This is a React Component</Text>
        <Text>{this.state.testProp}</Text>

</>
```

If major code refactoring and structural changes to the projects is an option, a lot more can be done in sharing component rendering, as long as the top level components are platform agnostic, then any imported components that use lower-level tags can be loaded on a by-platform basis.
For example, a top-level component called Menu.js might import a Button component or a MenuList component for rendering the list of available models. To do this, the top-level Menu.js component must not contain any HTML tags, and instead rely entirely on lower level

components. These lower level components would have multiple versions that react can import on a by-platform basis.
For example, the components directory might look like this:

components
|_ **platform**
| |_ button.js
| |_ button.native.js
| |_ menuList.js
| |_ menuList.native.js
| |_ …
|
|_ menu.js
|_ …

In the higher-level component, the lower-level component can be imported normally, and react should import the corresponding component based on platform:

```
import { Button } from "./platform/button"
```