Rohit Jindal

Follow 3 Followers About

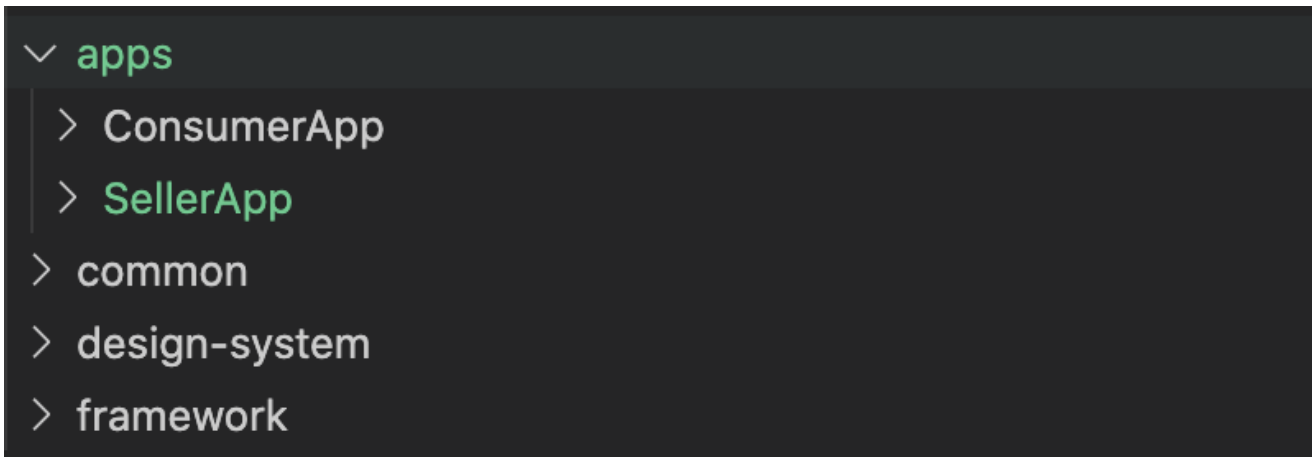
App based file resolution in React Native — MultiApp Framework

 Rohit Jindal Mar 10 · 2 min read

Recently we had a requirement of creating a common multi app framework in our company, where in common codebase can be used to serve multiple apps from same code base.

We decided to go ahead with React native as our preferred choice for Native App development since most of our team members were from React background. When we started the project we had a use case for 2 client facing Apps i.e ConsumerApp and Seller App. Our multi-app framework handled navigation , data management , networking tasks for each Apps.

This is how our code file structure looked like



So we had common framework code in “framework” folder and common code like Navigation root, Networking root file in “common” folder.

Soon we had a use case of providing individual Apps power to override files in common folder (This is very similar to what Android build system provides, wherein you can

in Consumer App / navigation folder it should be picked from the App instead of the common folder.

Here comes "Metro server" to the rescue. We were using metro server for bundling our JS bundle in the native App. For setting up metro, you need to define a config file which has namely two major attributes

> *transform*
>
> *resolve*

resolve provides a custom function resolveRequest which is called on every require statement in the codebase

```
resolveRequest: (context, realModuleName, platform, moduleName) => {
  // Resolve file path logic...

  return {
    filePath: "path/to/file",
    type: 'sourceFile',
  };
}
```

moduleName is the name of the module which is being required. Now it was upto us to define the complete business logic of changing the moduleName based on if the same file exists in the App folder in the same path, once the same file existed in the path, we just had to call our custom resolve with the new path

> *const Resolver = require('metro-resolver');*
>
> *const clearContext = { ...context, resolveRequest: undefined };*
>
> *return Resolver.resolve(clearContext, newPath, platform);*

NOTE: Remember to pass clearContext in resolve, informing Metro that you have handled the resolution logic.

Hence we were able to achieve the Android like file resolution functionality in our JS build system for React native courtesy Metro.