Every application will most likely run in more than one environment. Two obvious candidates for these environments are the development environment and the production environment. Out of these two, the development environment is the one we are running the application right now. Different environments usually have different dependencies, for example, the server we are developing locally might use a local database whereas the server that is deployed to the production environment uses the production database. To make the code environment independent we need to parametrize these dependencies. At the moment we are using one very environment dependant hardcoded value in our application: the URL of the server.

We have previously learned that we can provide running programs with environment variables. These variables can be defined in the command line or using environment configuration files such as *.env* files and third-party libraries such as *Dotenv*. Unfortunately, React Native doesn't have direct support for environment variables. However, we can access the Expo configuration defined in the *app.json* file at runtime from our JavaScript code. This configuration can be used to define and access environment dependant variables.

The configuration can be accessed by importing the Constants constant from the *expo-constants* module as we have done a few times before. Once imported, the Constants.manifest property will contain the configuration. Let's try this by logging Constants.manifest in the App component:

import { NativeRouter } from 'react-router-native';

import { ApolloProvider } from '@apollo/client';

import Constants from 'expo-constants';

import Main from './src/components/Main';

import createApolloClient from './src/utils/apolloClient';

const apolloClient = createApolloClient();

const App = () => {

console.log(Constants.manifest);

return (

<NativeRouter>

<ApolloProvider client={apolloClient}>

<Main />

</ApolloProvider>

</NativeRouter>

);

};

export default App;

You should now see the configuration in the logs.

The next step is to use the configuration to define environment dependant variables in our application. Let's get started by renaming the *app.json* file to *app.config.js*. Once the file is renamed, we can use JavaScript inside the configuration file. Change the file contents so that the previous object:

{

"expo": {

"name": "rate-repository-app",

// rest of the configuration...

}

}

Is turned into an export, which contains the contents of the expo property:

export default {

name: 'rate-repository-app',

// rest of the configuration...

};

Expo has reserved an [extra](https://docs.expo.dev/guides/environment-variables/#using-app-manifest--extra) property in the configuration for any application-specific configuration. To see how this works, let's add an env variable into our application's configuration:

export default {

name: 'rate-repository-app',

// rest of the configuration...

extra: { env: 'development' },};

Restart Expo development tools to apply the changes and you should see that the value of Constants.manifest property has changed and now includes the extra property containing our application-specific configuration. Now the value of the env variable is accessible through the Constants.manifest.extra.env property.

Because using hard coded configuration is a bit silly, let's use an environment variable instead:

export default {

name: 'rate-repository-app',

// rest of the configuration...

extra: { env: process.env.ENV, },};

As we have learned, we can set the value of an environment variable through the command line by defining the variable's name and value before the actual command. As an example, start Expo development tools and set the environment variable ENV as test like this:

ENV=test npm start

If you take a look at the logs, you should see that the Constants.manifest.extra.env property has changed.

We can also load environment variables from an .env file as we have learned in the previous parts. First, we need to install the [Dotenv](https://www.npmjs.com/package/dotenv) library:

npm install dotenv

Next, add a .env file in the root directory of our project with the following content:

ENV=development

Finally, import the library in the *app.config.js* file:

import 'dotenv/config';

export default {

name: 'rate-repository-app',

// rest of the configuration...

extra: {

env: process.env.ENV,

},

};

You need to restart Expo development tools to apply the changes you have made to the *.env* file.

Note that it is *never* a good idea to put sensitive data into the application's configuration. The reason for this is that once a user has downloaded your application, they can, at least in theory, reverse engineer your application and figure out the sensitive data you have stored into the code.