**Twelve Factor App Implementation**

**The Twelve-Factor App:** It is a set of guidelines for building cloud-native applications. By cloud-native, we will mean an application that is portable across environments, easy to update, and scalable enough to take advantage of the elastic capabilities of the cloud.

**Sprint Boot**: Developing Spring applications with Spring Boot is a real time saver. It gets you up and running quickly, and continues to make your life easier as you move to production and start releasing incremental updates to your

application.

**React JS**: Create React App does the same thing for your React frontend. It speeds up the initial phase by stitching together everything you need to get your React frontend working with minimal effort on your part, and it shortens your feedback loop by enabling hot reload. That way, you can just save your files, and changes will be immediately visible in the browser.

In this application, I have combined Create React App with Spring Boot, and generated a single jar file to deploy.

**Create a Spring Boot Java Application**

* First, create a spring boot project with https://start.spring.io. Add the `Web`,’Lombok dependency.
* Generate the project and unzip it into your project directory. You probably want to initialize git, add a .gitignore and make your initial commit at this point.
* Run the java application using maven.

You can view the output in the browser

URL: http://localhost:8080/

**Create a React Application**

* Run the following command to check the Node version in the command prompt: $ node -v
* Run the following command to check the NPM version in the command prompt; $ npm -v
* We can install React using npm package manager by using the following command.

**npm install -g create-react-app**

* Once the React installation is successful, we can create a new React project using create-react-app command. Here, I choose "reactproject" name for my project.

**create-react-app frontend**

* Install the required libraries by using npm install react, react-dom, and react-scripts etc
* Change the directory to frontend: $ cd frontend.
* Staring the react app by using the following command: $ npm start
* You will get the following output in the command line

Compiled successfully!

You can now view frontend in the browser.

Local: [**http://localhost:3000/**](http://localhost:3000/)

On Your Network: [**http://172.16.25.84:3000/**](http://172.16.25.84:3000/)

**Implementing the 12 factors**

1. **Codebase**

A Twelve-Factor App requires code to be stored in source control (e.g., Git), and defines the one-to-many relationship between a codebase and deploys (running instances of the app) resulting from it. It’s quite typical to have different versions of the app (from different commits) running in different environments.

1. Initialize the empty git repository: **GIT INIT**
2. Create a java application
3. Create a react application
4. The changes has to be committed and pushed to the remote repository
5. **Dependencies**

Declaring and isolating dependencies helps avoid conflicts between different applications that run on the same host but require different dependencies

For Java Application:

1. Create a pom file
2. Add the required dependencies like spring web, JPA etc
3. When we are running the java application, it fetches dependencies as jar (artifacts) from various repositories

For React Application:

1. After creating react app, a package.json file will be created.
2. To add dependency to the package.json file using the npm install command

Eg: npm install axios

1. **Config**

Configurations are a central part of any application, specifically when there is a need to support multiple environments or clients. Use cases are as follows:

· Database connection properties

· Backing services credentials and connection information

· Application environment specific information such as Host IP, Port, etc.

For Java Application:

1. we have added the databases properties like username, password. Etc to the application.properties file

For React Application

1. We have added the Proxy setting to get the data from spring boot application
2. **Backing Services**

Backing services refer to the infrastructure and other services by which the application communicates over the network. Database, Message Brokers, other API-accessible consumer services such as Authorization Service, Twitter, GitHub etc., are loosely coupled with the application and treat them as resource.

We have added Azure SQL services as our backing services

Eg: spring.datasource.url=jdbc:sqlserver://<**ServerName**>:1433;database=<**DatabaseName**>;encrypt=true;trustServerCertificate=false;hostNameInCertificate=\*.database.windows.net;loginTimeout=30;

1. **Port Binding**

Spring Boot can really assist here so that we can use embedded Tomcat, Jetty or Undertow using the dependency manifest (pom.xml).

In Spring boot application, we can bind our application to particular port

Eg: server.port=9090

1. **Logs**

Twelve-factor apps should not be concerned about routing and storage of its output stream or writing/managing logfiles — the app will write it’s event stream to stdout.

To log the spring boot application, we have used the LOGGER FACTORY and we have stored the logs into separate files.

**Integrating the React application with Spring boot application**

Calling rest services in spring from React Now we have a backend server in Spring Boot running at http://localhost:8080 and a frontend in React running at http://localhost:3000. We'd like to be able to call services in the backend and display the results in the frontend. In order to do this (and not get into trouble with any cross-origin requests (CORS)) we ask the frontend server to proxy any requests from `:3000` to `:8080`.

* you have to add a `proxy` entry to `frontend/package.json`. This will ensure that the web server at :3000 proxies any requests to `http://localhost:3000/api/\*` to `http://localhost:8080/api`.

Frontend/package.json`

"proxy": "http://localhost:8080",

* Make sure you have the backend running and restart the frontend. You should now be able to fetch data from spring boot application.
* Next, let's add a rest call to the frontend:

Open `frontend/src/App.js` and add the rest api calls to the react application using axios.

* We now have a React frontend that talks to our Spring Boot backend.

**Packaging the React app with Spring Boot**

* We'd like to be able to publish \*one\* jar file to production, and that jar file should contain both the backend and the frontend.
* Spring Boot applications can serve static content if you putit into the `classes/public` directory of the application jar file. Create React App can build a static bundle for production by running `npm build` in the frontend directory.

To accomplish this, we have to do the following:

1. create a production build of the frontend

2. copy the production build into `${target/classes/public}`

* We'll use `frontend-maven-plugin`
* in step 1, and `maven-ant run-plugin`
* in step 2. When we're done,
* we can just type `$ mvn clean install` and we'll end up with a single jar file containing both the frontend and the backend.

When you run `mvn clean install`, maven will install npm and node locally and run `npm build`

in the `frontend` directory.

```

$ mvn clean install

```

This results in a production build of the frontend in `frontend/build`:

Include frontend build files in spring boot jar

We now have to copy these files to `target/classes/public` in order to serve them as static resources

from the Spring Boot application. We'll use the ant plugin for this.

Add the following to `pom.xml` under `/build/plugins`:

<copy todir="${project.build.directory}/classes/public">

<fileset dir="${project.basedir}/frontend/build"/>

This will ensure that the frontend build files are copied after they have been generated by `npm build`.

Run maven again, and inspect the contents of the `target/classes` directory:

$ java -jar target/twelvefactorapp-0.0.1-SNAPSHOT.jar

You now have a spring boot application with a React frontend. During development, you can now run the application

using `React-scripts` by running `cd frontend; npm start`, and you'll have all the benefits of rapid application

development with hot reloads and everything you might wish for, while being able to deploy the application to test

and production environments as a single artifact.

Happy Learning!