MANUAL DO DESENVOLVEDOR DO SISTEMA

visão

Visão

This application was generated using JHipster 5.1.0, you can find documentation and help at https://www.jhipster.tech/documentation-archive/v5.1.0.

1. Development

Before you can build this project, you must install and configure the following dependencies on your machine:

1. Node.js: We use Node to run a development web server and build the project. Depending on your system, you can install Node either from source or as a pre-packaged bundle.
2. Yarn: We use Yarn to manage Node dependencies. Depending on your system, you can install Yarn either from source or as a pre-packaged bundle.

After installing Node, you should be able to run the following command to install development tools. You will only need to run this command when dependencies change in package.json.

yarn install

We use yarn scripts and Webpack as our build system.

Run the following commands in two separate terminals to create a blissful development experience where your browser auto-refreshes when files change on your hard drive.

./gradlew
yarn start

Yarn is also used to manage CSS and JavaScript dependencies used in this application. You can upgrade dependencies by specifying a newer version in package.json. You can also run yarn update and yarn install to manage dependencies. Add the help flag on any command to see how you can use it. For example, yarn help update.

The yarn run command will list all of the scripts available to run for this project.

Service workers

Service workers are commented by default, to enable them please uncomment the following code.

- The service worker registering script in index.html

```
<script>
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker
    .register('./service-worker.js')
    .then(function() { console.log('Service Worker Registered'); });
  }
</script>
```

Note: workbox creates the respective service worker and dynamically generate the service-worker.js

2. Managing dependencies

For example, to add Leaflet library as a runtime dependency of your application, you would run following command:

yarn add --exact leaflet

To benefit from TypeScript type definitions from DefinitelyTyped repository in development, you would run following command:

yarn add --dev --exact @types/leaflet

Then you would import the JS and CSS files specified in library's installation instructions so that Webpack knows about them: Edit src/main/webapp/app/vendor.ts file:

import 'leaflet/dist/leaflet.js';

Edit src/main/webapp/content/css/vendor.css file:

@import '~leaflet/dist/leaflet.css';

Note: there are still few other things remaining to do for Leaflet that we won't detail here.

For further instructions on how to develop with JHipster, have a look at Using JHipster in development.

Using angular-cli

You can also use Angular CLI to generate some custom client code.

For example, the following command:

ng generate component my-component

will generate few files:

create src/main/webapp/app/my-component/my-component.component.html
create src/main/webapp/app/my-component/my-component.component.ts
update src/main/webapp/app/app.module.ts

3. Building for production

To optimize the visao application for production, run:

./gradlew -Pprod clean bootWar

This will concatenate and minify the client CSS and JavaScript files. It will also modify index.html so it references these new files. To ensure everything worked, run:

java -jar build/libs/*.war

Then navigate to http://localhost:8080 in your browser.

Refer to Using JHipster in production for more details.

4. Testing

To launch your application's tests, run:

./gradlew test

Client tests

Unit tests are run by Jest and written with Jasmine. They're located in src/test/javascript/ and can be run with:

yarn test

For more information, refer to the Running tests page.

visão

Optional

1. Using Docker to simplify development (optional)

You can use Docker to improve your JHipster development experience. A number of docker-compose configuration are available in the src/main/docker folder to launch required third party services.

For example, to start a postgresql database in a docker container, run:

docker-compose -f src/main/docker/postgresql.yml up -d

To stop it and remove the container, run:

docker-compose -f src/main/docker/postgresql.yml down

You can also fully dockerize your application and all the services that it depends on. To achieve this, first build a docker image of your app by running:

./gradlew bootWar -Pprod buildDocker

Then run:

docker-compose -f src/main/docker/app.yml up -d

For more information refer to Using Docker and Docker-Compose, this page also contains information on the docker-compose sub-generator (jhipster docker-compose), which is able to generate docker configurations for one or several JHipster applications.

2.Continuous Integration (optional)

To configure CI for your project, run the ci-cd sub-generator (jhipster ci-cd), this will let you generate configuration files for a number of Continuous Integration systems. Consult the Setting up Continuous Integration page for more information.