JHipster is a development platform to quickly generate, develop, & deploy modern web applications & microservice architectures.

JHipster CLI

Configuration

In order to generate a JHipster project, the JHipster CLI is used. For this project JHipster has been configured with the following settings (can be found **in yo-rc.json** in the root directory):

- **Application type – monolith**
  A classical, one-size-fits-all application, easy to use and develop – the default.
- **Authentication type – JWT**
  JSON Web Token is used to securely transmit information between parties in a JSON format. Commonly used to authorise access to resources and services.
- **Build tool – Maven**
  Maven is a build automation tool used primarily for Java projects. It makes use of a **pom.xml** file to determine which dependencies to install and resolves them automatically. The file **mvnw / mvnw.cmd** is a maven wrapper which allows developers to run a maven project using the correct version of maven, without having it installed and present on the path.

Commands

jhipster : generate a new JHipster application, answer questions about type of application, authentication method, database, etc.
jhipster ci-cd : generate a ci and cd pipelines JHipster answer questions about type of ci pipeline needed
jhipster entity : Generates a single new entity along with associated files: entity classes, database migration scripts, REST API endpoints, and Angular/React components.
jhipster jdl myapp.jdl : generate all the entities and relationships specified in the file myapp.jdl
./mvnw : Runs the application in development mode on a laptop

Dependency Management and Bundling

NPM

Node.js is a cross-platform, open-source JavaScript runtime environment, which enables developers to execute JavaScript code outside of the web browser. Node.js is often used to build highly-scalable back-end JavaScript APIs. NPM is the package manager that comes preinstalled with Node.js.

Package.json

NPM references the information in package.json to start the project, run scripts, and install dependencies. Some attributes include:

Scripts: keys are script-names which can be run using npm run 'SCRIPT_NAME', with values being the corresponding shell code. This helps automate repetitive terminal tasks. For example:

```
"app:start": "./mvnw"
```

Dependencies: contains the external dependencies, with key=dependency name, value=version. NPM uses this list to fetch the dependencies and their versions when building the system.

devDependencies: contains the external dependencies only to be used during the development of the project, not kept for deployment. For example: **jest-junit**, a JavaScript unit testing framework.

Using NPM to manage dependencies

You can manage dependencies by specifying them in package.json, or by running `npm update` and `npm install`.

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

npm install --save --save-exact leaflet

Webpack

Webpack is a module bundler. It takes code which might be composed of multiple files and formats, and bundles it into one or more output files. This way, we can reduce the number of requests, improve loading speed, and manage dependencies.

File Structure

Root

In the project root, JHipster generates configuration files for tools like git, prettier, eslint, husky, and others. Here are some important examples:

.yo-rc.json: contains the settings used to generate JHipster
.jhipster/*.json: JHipster entity configuration files
npmw: JHipster installs Node and npm locally using the build tool by default. This wrapper makes sure npm is installed locally and uses it avoiding some differences different versions can cause.
gitlab-ci.yml: a ci pipeline using docker
src/main/docker/app.yml : the docker compose script to start the app and postgres on a the server in development mode.
src/main/docker/prd.yml : the docker compose script to start the app and postgres on a the server in development mode.

Frontend (Angular)

src/main/webapp/app: This directory contains the Angular application code.
src/main/webapp/app/shared: Shared components, services, and utilities used across the application.
src/main/webapp/app/entities: Subdirectories for each entity in the application, with components, services, and models specific to that entity.
src/main/webapp/app/layouts: Layout components such as headers, footers, and navigation menus.
src/main/webapp/app/admin: Administrative components and modules.
src/main/webapp/app/core: Core services, interceptors, guards, and modules.
src/main/webapp/app/main.ts: The main entry point for the Angular application.
src/main/webapp/app/app.module.ts: The root module where you define application-wide dependencies and configurations.
src/main/webapp/app/router/router.module.ts: The Angular router module where you define application routes.
src/main/webapp/content/css: CSS stylesheets for styling the application.
src/main/webapp/content/images: Image assets used in the application UI.

Backend (Java/Springboot)

src/main/java: the Java source code for the application, including controllers, services, repositories, security configs, and domain models.
src/main/resources: Configuration files, Liquibase database migrations, i18n files, and static resources.