CHAPTER 20

Introducing JHipster

JHipster is a Yeoman-based generator that creates Spring Boot-based web applications. JHipster configures a wide variety of tools and frameworks commonly used in Spring Boot applications, improving developer productivity.

This chapter covers how to install JHipster and create a monolithic application. It also explores the generated application features and looks at how to create entities using the sub-generator and JDL Studio.

Introducing JHipster

Technology is evolving rapidly and new tools, frameworks, and libraries are created daily. In recent years, there has been a lot of innovation in the JavaScript ecosystem and many high-quality, modern web development tools have been born. There are build tools like Webpack and Gulp. There are single page application (SPA) frameworks like React, Angular, and VueJS. And there are many JavaScript testing libraries like Mocha, Jasmine, and Jest. Integrating all of these tools manually is tedious and repetitive.

Yeoman (`http://yeoman.io/`) is a scaffolding tool that generates web projects following best practices. Yeoman provides various generators to scaffold web projects using various technologies. For example, if you want to create a React-based project, you can use `generator-react`, which will generate a ReactJS project with the Gulp build tool, with karma-based testing support.

JHipster (`www.jhipster.tech/`) is a Yeoman-based generator that generates Spring Boot-based web projects with a wide variety of options for building tools, front-end frameworks, relational databases, NoSQL databases, Spring security strategies, caching options, and more.

K. S. P. Reddy and S. Upadhyayula, *Beginning Spring Boot 3*, https://doi.org/10.1007/978-1-4842-8792-7_20

With JHipster, you can generate Spring Boot applications with most configurations appropriately configured and then start implementing the business use cases. JHipster also provides sub-generators to generate JPA entities and a scaffolding UI for typical CRUD operations, making development faster.

Installing JHipster

JHipster is a Yeoman-based generator that depends on the NPM (Node Package Manager). The following section covers the prerequisites for using JHipster.

Prerequisites

Follow these steps to install JHipster:

1. Install JDK 17.
2. Install Git from `https://git-scm.com/`.
3. Install Node.js from `https://nodejs.org/`.
4. Run `npm install -g generator-jhipster.`

You should be able to run `jhipster --help` and see the various commands that JHipster supports.

Creating a JHipster Application

Creating a JHipster application is easy; you simply run the `jhipster` command and answer the questions based on your technology's preferences and application's needs. JHipster can generate a monolithic application or a microservices-based application.

In this chapter, you will create a simple monolithic blog application and then use the relational database `H2` for development and `MySQL` for production.

```
> mkdir jhipster-blog
> cd jhipster-blog
> jhipster
```

The jhipster command will ask you a series of questions. Select the options shown here:

```
? Which *type* of application would you like to create? Monolithic
application (recommended for simple projects)
? What is the base name of your application? jhipsterblog
? Do you want to make it reactive with Spring Webflux? No
? What is your default Java package name? com.apress.jhblog
? Which *type* of authentication would you like to use? JWT authentication
(stateless, with a token)
? Which *type* of database would you like to use? SQL (H2, PostgreSQL,
MySQL, MariaDB, Oracle, MSSQL)
? Which *production* database would you like to use? MySQL
? Which *development* database would you like to use? H2 with in-memory
persistence
? Which cache do you want to use? (Spring cache abstraction) Ehcache (local
cache, for a single node)
? Do you want to use Hibernate 2nd level cache? Yes
? Would you like to use Maven or Gradle for building the backend? Maven
? Do you want to use the JHipster Registry to configure, monitor and
scale your
application? No
? Which other technologies would you like to use?
? Which *Framework* would you like to use for the client? Angular
? Do you want to generate the admin UI? No
? Would you like to use a Bootswatch theme (https://bootswatch.com/)?
Journal
? Choose a Bootswatch variant navbar theme (https://bootswatch.com/)?
Primary
? Would you like to enable internationalization support? No
? Please choose the native language of the application English
? Besides JUnit and Jest, which testing frameworks would you like to use?
? Would you like to install other generators from the JHipster
Marketplace? No
```

Based on the options selected here, JHipster will generate a Spring Boot application with the following features:

- Angular-based front end with Webpack configuration
- `H2` in-memory database used in development and `MySQL` used for production
- Liquibase migration support for database migrations
- Spring Data JPA configured for database interaction
- Caching support configured using EHCache
- Spring Security JWT token-based authentication
- An administration dashboard showing application metrics using the Spring Boot Actuator
- Ability to change log levels at runtime through UI
- Open API-based Rest API documentation
- User accounts out of the box with login, change password, and new user registration functionality

You can run the application by running `./mvnw` on Linux/MacOS or `mvnw.cmd` on Windows. This command will start the application `dev` profile and is accessible at `http://localhost:8080/`. Next, you'll explore the generated application.

As shown on the home page, you can log in with `admin/admin`, which has both the `ROLE_USER` and `ROLE_ADMIN` roles, or with `user/user`, which has only the `ROLE_USER` role.

Log in as the `admin` user using `admin/admin`. After a successful login, you will be redirected to the home page. The top navigation bar includes the Entities, Administration, and Account menus. As you haven't created any entities yet, there won't be any entities listed in the Entities menu.

For gateway and microservice-based applications, JHipster provides a default monitoring UI to view the application metrics, such as memory consumption, thread states, garbage collection details, and HTTP request statistics, which are provided using Micrometer (Figure 20-1).

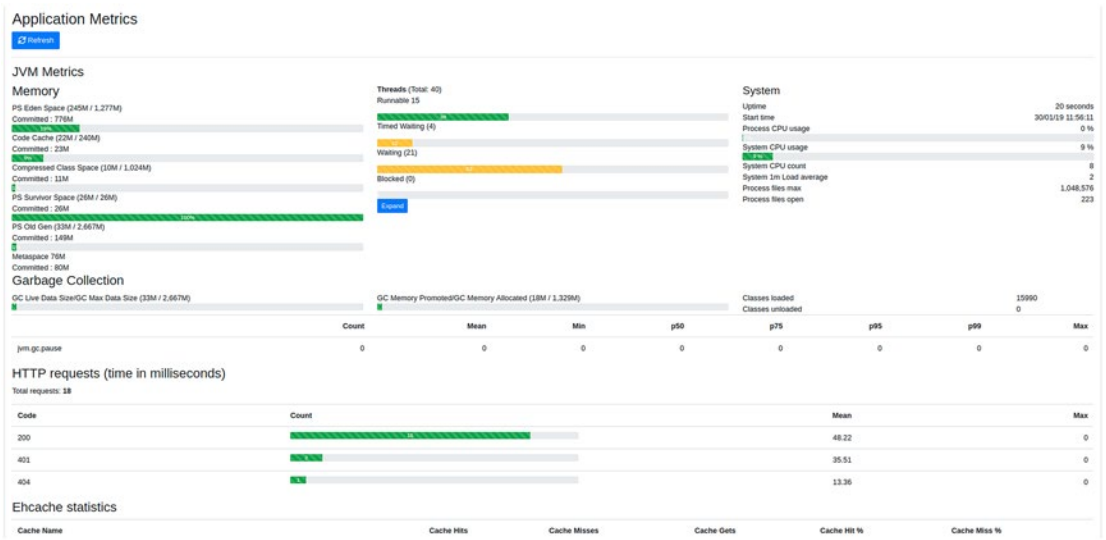

Figure 20-1. *JHipster metrics dashboard*

Click Administration ➤ Database to open the H2 in-memory database console, where you can explore the current state of the database.

You can also manage the application's users by clicking Administration ➤ User Management, where you can perform CRUD operations on users. See Figure 20-2.

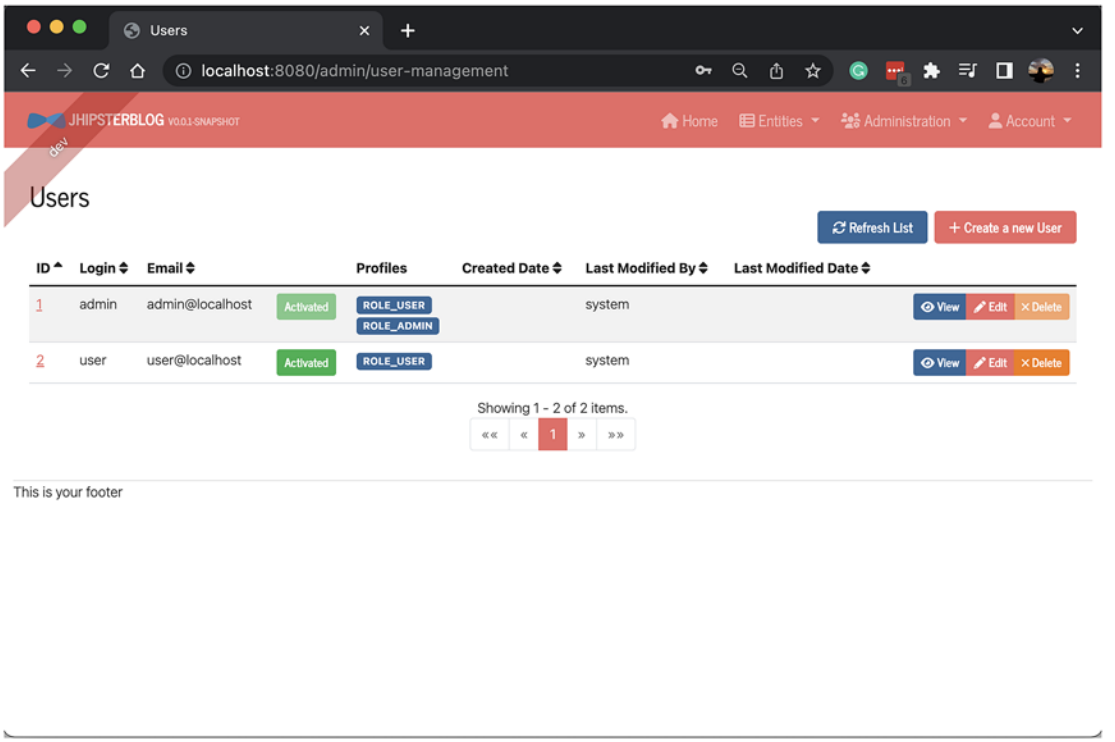

Figure 20-2. *JHipster user management*

You can view the Swagger documentation for the REST API by choosing Administration ➤ API. You can also view the Request and Response formats for each endpoint and trigger REST API calls by providing inputs if needed.

Creating Entities

Once the application is created, you may want to create entities and a scaffold UI for that entity to perform CRUD operations on it. JHipster provides ways to create entities and perform the following tasks:

- Create a JPA entity
- Create a database table based on the property information provided
- Create a Liquibase changeset for database migration
- Create a Spring Data JPA repository for the entity

- Create a Spring MVC REST controller with basic CRUD operations
- Create an Angular router, component, and service
- Create HTML views
- Generate integration and performance tests

You can generate entities using the `jhipster entity` sub-generator, the JHipster Domain Language (JDL) Studio, or the JHipster UML (www.jhipster.tech/jhipster-uml/). The following section uses the JHipster entity sub-generator and JDL Studio to generate entities.

Using the JHipster Entity Sub-Generator

You can generate entities using the `jhipster entity` sub-generator by providing a table name and column details, as follows:

```
jhipster entity Post --table-name posts
```

This command will ask whether you want to add a field to your entity. In this example, you'll add three fields named `title`, `content`, and `createdOn` and specify type and validation rules as follows:

- Name: `title`, Type: `String`, Validation: Required
- Name: `content`, Type: `String`, Validation: Required
- Name: `createdOn`, Type: `LocalDate`

Next, it will ask if you want to add a relationship to another entity. Answer no. You will learn how to manage relationships in the next section.

The following questions will be asked; answer them as follows:

```
? Do you want to use separate service class for your business logic?
? Is this entity read-only? No
? Do you want pagination and sorting on your entity? No
```

You can choose to create data transfer objects (DTOs), which will be used to create a response for the REST API, but for now, you can choose to return entities. You can also choose to create a service layer to perform any business logic, but you are choosing to directly use the Spring Data JPA repositories because there is no business logic involved. Lastly, you can choose whether you need pagination support or not.

After successfully running the JHipster entity sub-generator command, you can run the application and see the `Post` menu item in the Entities menu. You can perform the CRUD operations on the `Post` entity.

Instead of running the entity sub-generator and answering all these questions, you can use JDL Studio to create entities.

Using JDL Studio

JDL Studio is an online utility that creates entities and configures relationships among entities using the JHipster Domain Language (JDL). You can read about JDL at `www.jhipster.tech/jdl/intro`.

If you go to `https://start.jhipster.tech/jdl-studio/`, a sample domain model is configured with various entity definitions and relationships among those entities. Remove all of that and add the `Post` entity definition as follows:

```
entity Post {
        title String required
    content String required
    createdOn LocalDate
}
```

Click the "Download Text File of This JDL" link in the top-right corner. The jhipster-jdl.jdl file will download. Now you can run the `jhipster import-jdl` command to create the entities from the JDL file.

```
> jhipster import-jdl jhipster-jdl.jdl
```

After running this command, the `Post` entity, Spring Data JPA repository, Spring MVC controller, Angular front-end components, and more will be generated. If you already have a `Post` entity, it will update the entity.

Managing Relationships

You can use the JHipster `entity` command not only for creating entities but also to specify relationships among entities. For example, you can create a `Comment` entity, then establish a *one-to-many* relationship from `Post` to `Comment` and a *many-to-one* entity from `Comment` to `Post`.

While creating the Post entity using the jhipster entity sub-generator, you can specify the relationship to the Comment entity as follows:

```
> jhipster entity Post --table-name posts
```

As you already have a Post entity, it will display options to regenerate, add, and remove fields and relationships. Choose the following:

Yes, add more fields and relationships.

```
? Do you want to add a field to your entity? No
? Do you want to add a relationship to another entity? Yes
? What is the name of the other entity? Comment
? What is the name of the relationship? comments
? What is the type of the relationship? one-to-many
? What is the name of this relationship in the other entity? post
```

Now you can generate a Comment entity with the name, email, content, and createdOn fields. When prompted to add a relationship to another entity, you can add a many-to-one relationship from Comment to the Post entity as follows:

```
? What is the name of the other entity? Post
? What is the name of the relationship? post
? What is the type of the relationship? many-to-one
? When you display this relationship with Angular, which field from 'Post'
do you want to use? id
? Do you want to add any validation rules to this relationship? No
```

You can use JDL Studio to create entities and specify relationships among them as well. Configure the Post and Comment entities and the OneToMany relationship as follows:

```
entity Post {
        title String required
    content String required
    createdOn LocalDate
}
entity Comment {
        name String required
    email String required
```

```
    content String required
    createdOn LocalDate
}
relationship OneToMany {
  Post{comments} to Comment{post}
}
```

You can download this JDL file and import it as you did earlier. This will create JPA entities and establish JPA relationship as follows:

```
@Entity
@Table(name = "post")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Post implements Serializable {
    ...
    ...
    @OneToMany(mappedBy = "post")
    @JsonIgnore
    @Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
    private Set<Comment> comments = new HashSet<>();
    ...
    ...
}
@Entity
@Table(name = "comment")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Comment implements Serializable {
    ...
    ...
    @ManyToOne
    private Post post;
}
```

JHipster scaffolding will generate a dropdown with `posts` to select while creating a comment.

You can read more about managing relationships at www.jhipster.tech/managing-relationships/.

By default, when you run `./mvnw`, the application will start in development mode using the `dev` profile. If you want to run the application in production mode, you can run it using the `prod` profile, as in `./mvnw -Pprod`.

You can also generate a runnable WAR file using the `./mvnw -Pprod package` command and run the application as follows:

```
java -jar jhipsterblog-0.0.1-SNAPSHOT.war
```

Various optimizations will be performed when you run the application in the production profile. For example, static assets like HTML, JS, and CSS files will be optimized and GZip compression will be configured.

Note You can also use JHipster to generate Spring Boot-based microservices. To learn how to create microservices using JHipster, refer to `www.jhipster.tech/microservices-architecture/`.

Summary

In this chapter, you learned how to use JHipster to generate Spring Boot-based web applications with the Angular front end. In the next chapter, you will look at how to run production applications and deploy a Spring Boot application on the Heroku cloud platform.