What is JHipster?

JHipster is one of those open-source projects which combines three very successful frameworks in web development: Bootstrap, Angular, and Spring Boot.

 JHipster is a [Yeoman](http://yeoman.io/) generator. Yeoman is a code generator that you run with a yo command to generate complete applications or useful pieces of an application.

This is an opinionated client-side stack of tools that can help developers quickly build beautiful web applications. It takes care of providing everything needed to get working without the normal pains associated with a manual setup.

**Install JHipster 4**

The Installing JHipster instructions show you all the tools you’ll need to use a released version of JHipster.

1. Install Java 8 from Oracle.
2. Install Git from <https://git-scm.com>.
3. Install Node.js from http://nodejs.org. I used Node 6.9.1 to write this article.
4. Install Yarn using the Yarn installation instructions.
5. Run the following command to install Yeoman.

yarn global add yo

1. Run the following command to install JHipster.

yarn global add generator-jhipster

***Create a Project***

*To create a project,* Navigate into the directory which created for the project example: MyJhipsterApp . and run yo jhipster in cmd. You’ll be asked a number of questions about the type of application you want to create and what features you’d like to include.

1. Example (1/15) Which \*type\* of application would you like to create? (Use arrow keys)

Monolithic application (recommended for simple projects)

Microservice application

Microservice gateway

[BETA] JHipster UAA server (for microservice OAuth2 authentication)

1. ***What is the base name of your application? (MyJhipsterApp)***
2. ***Would you like to install other generators from the JHipster Marketplace? //no***
3. ***What is your default Java package name? // you need to write your package name***
4. ***Which \*type\* of authentication would you like to use? (Use arrow keys)***
5. ***HTTP Session Authentication (stateful, default Spring Security mechanism) //selected***
6. ***JWT authentication (stateless, with a token)***
7. ***OAuth2 Authentication (stateless, with an OAuth2 server implementation)***
8. ***Which \*type\* of database would you like to use? (Use arrow keys)***

***SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL) //selected***

***MongoDB***

***Cassandra***

1. ***(7/15) Which \*production\* database would you like to use? (Use arrow keys)***

***MySQL//selected***

***MariaDB***

***PostgreSQL***

***Oracle (Please follow our documentation to use the Oracle proprietary driver)***

***Microsoft SQL Server***

1. ***(8/15) Which \*development\* database would you like to use? (Use arrow keys)***

***H2 with disk-based persistence***

***H2 with in-memory persistence***

***MySQL//selected***

1. ***(9/15) Do you want to use Hibernate 2nd level cache? (Use arrow keys)***

***No***

***Yes, with ehcache (local cache, for a single node) //selected***

***Yes, with HazelCast (distributed cache, for multiple nodes)***

1. ***(10/15) Would you like to use Maven or Gradle for building the backend? (Use***

***arrow keys)***

***Maven//selected***

***Gradle***

1. ***? (11/15) Which other technologies would you like to use? (Press <space> to sel***

***ect, <a> to toggle all, <i> to inverse selection)***

***( ) Social login (Google, Facebook, Twitter)***

***( ) Search engine using Elasticsearch // selected***

***( ) WebSockets using Spring Websocket***

***( ) [BETA] Asynchronous messages using Apache Kafka***

1. ***Which \*Framework\* would you like to use for the client? (Use arrow keys)***

***AngularJS 1.x***

***[BETA] Angular 2.x//selected***

1. ***? (13/15) Would you like to use the LibSass stylesheet preprocessor for your CS***

***S? (y/N) // yes***

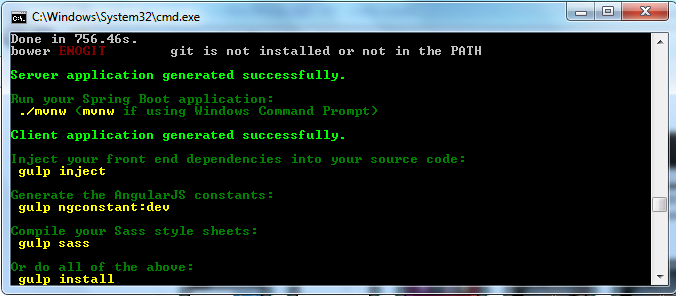
1. ***? (14/15) Would you like to enable internationalization support? (Y/n) // n***
2. ***? (15/15) Besides JUnit and Karma, which testing frameworks would you like to use? (Press <space> to select, <a> to toggle all, <i> to inverse selection)***

***( ) Gatling // not select any thing press enter***

***( ) Cucumber***

***( ) Protractor***

***Output will come like bellow:***



Create Sql database with name as your project name. we can use another database name also :

Then we need to go <project name>\src\main\resources\config:

Open the application-dev and application-prod and need to check and write your username and password of sql . myjhipsterapp is my sql database .

datasource:

type: com.zaxxer.hikari.HikariDataSource

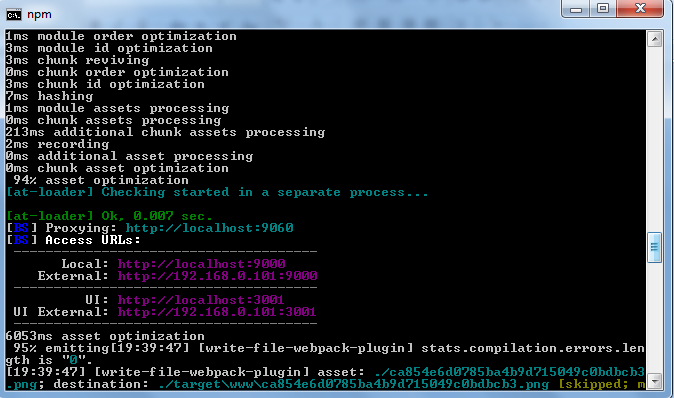
url: jdbc:mysql://localhost:3306/myjhipsterapp?useUnicode=true&characterEncoding=utf8&useSSL=false

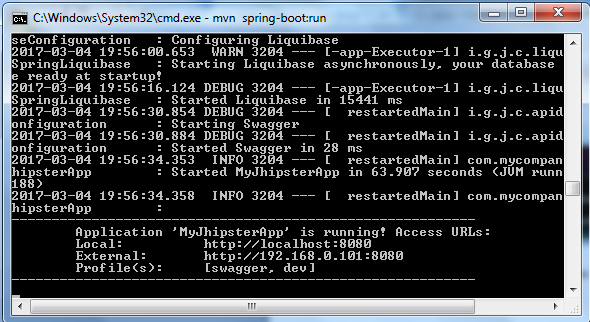
username: root

password: root

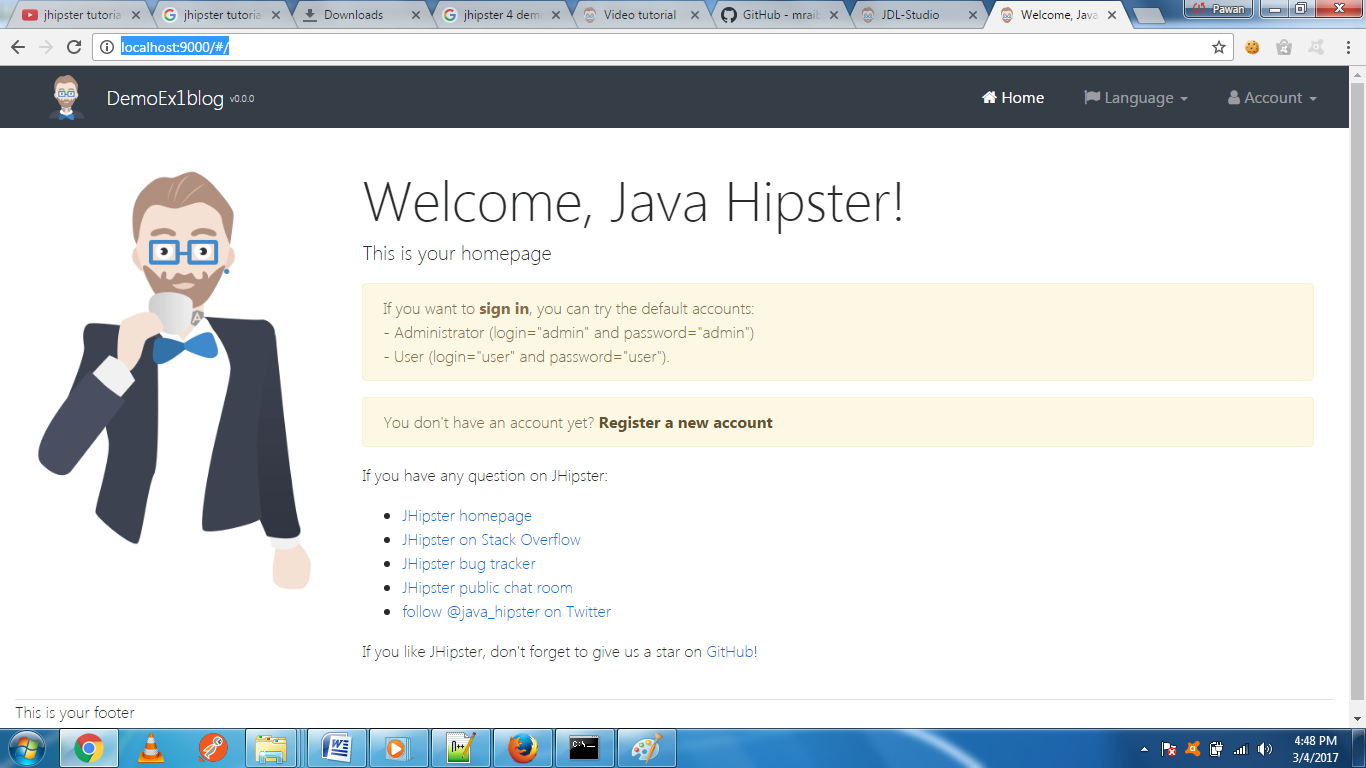
After that to run the project we execute command 2 command :

1. npm start
2. mvn spring-boot:run





navigate to http://localhost:9000 in your favorite browser.

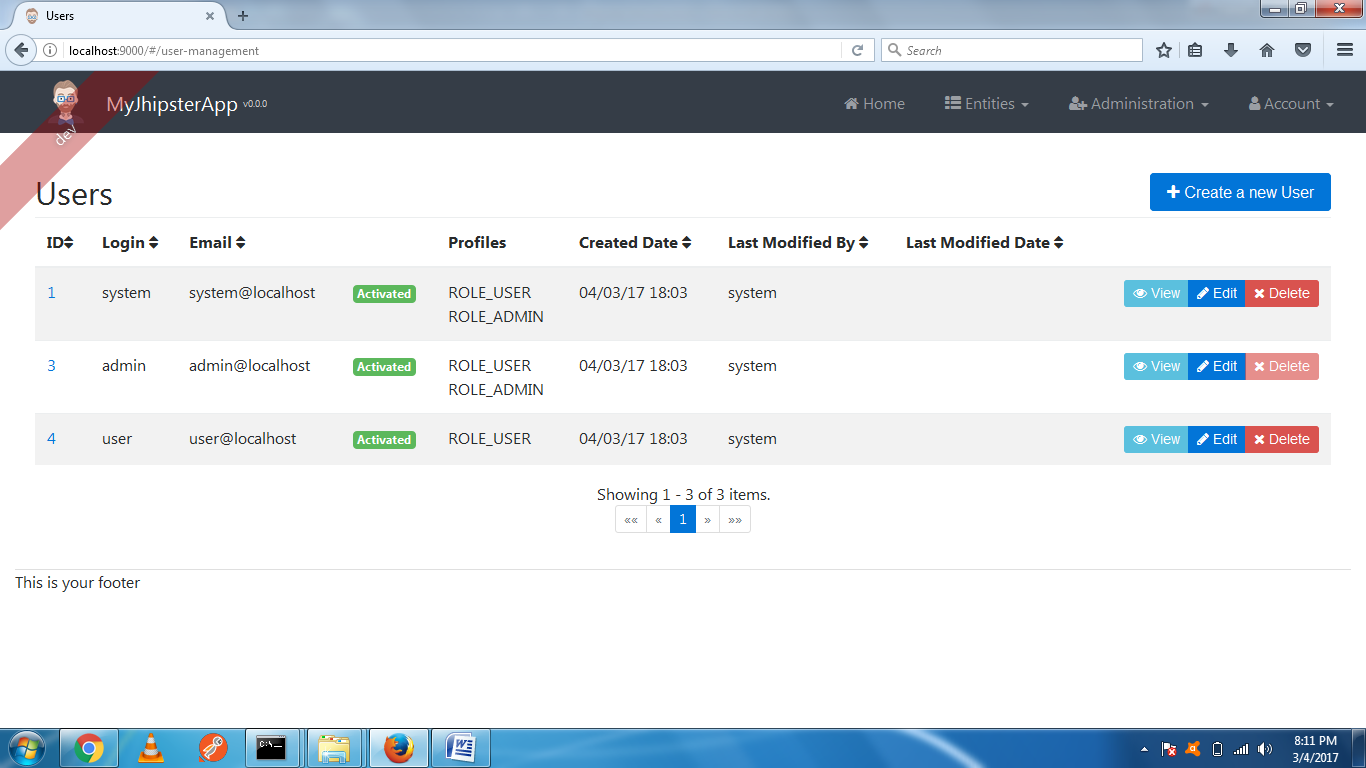


Default homepage

Sign in with username admin and password admin and you’ll have access to navigate through the Administration section. This section offers nice looking UIs on top of some Spring Boot’s many monitoring and configuration features. It also allows you to administer users:

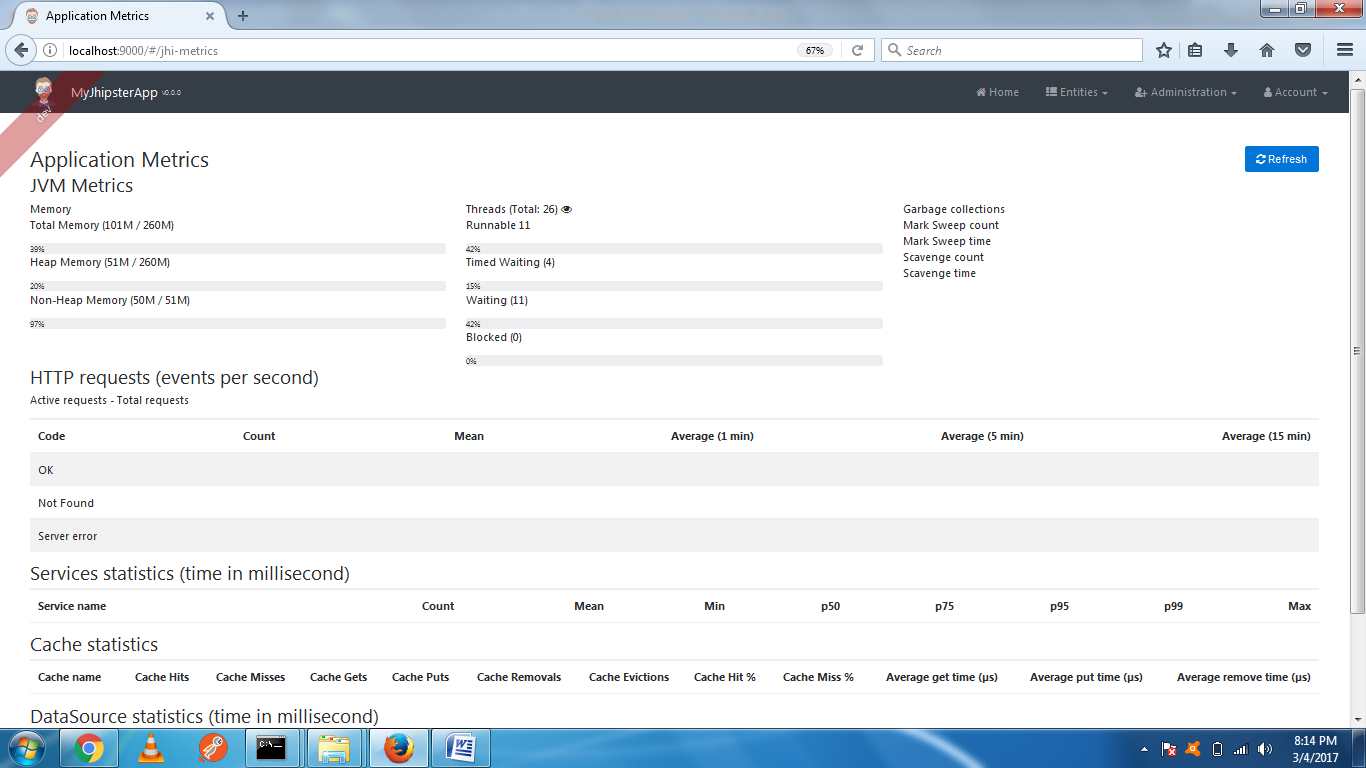
User management

It gives you insights into Application

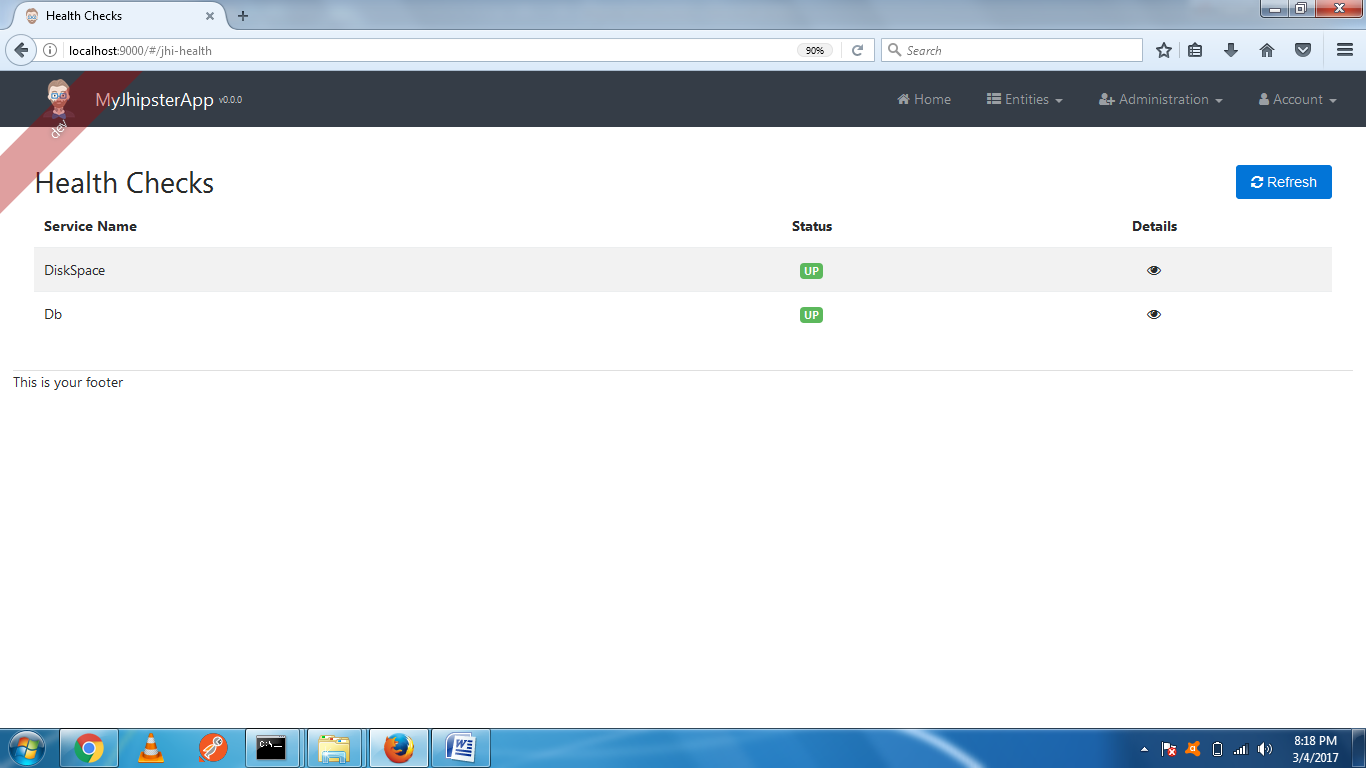


Application metrics

And it allows you to see the Swagger docs associated with its API.



**Health Checks :** it tell the health of the pc or server



**Creating an entity**

**Introduction**

Once you have created your application, you will want to create entities. For example, you might want to create an Author and a Book entity. For each entity, you will need:

* A database table
* A Liquibase change set
* A JPA Entity
* A Spring Data JPA Repository
* A Spring MVC REST Controller, which has the basic CRUD operations
* An Angular router, a component and a service
* An HTML view
* Integration tests, to validate everything works as expected
* Performance tests, to see if everything works smoothly

If you have several entities, you will likely want to have relationships between them. For this, you will need:

* A database foreign key
* Specific JavaScript and HTML code for managing this relationship

The “entity” sub-generator will create all the necessary files, and provide a CRUD front-end for each entity (see project structure). The sub generator can be invoked by running yo jhipster:entity <entityName> --[options].

**Entity fields**

For each entity, you can add as many fields as you want. You will need to input the field names and their types, and JHipster will generate for you all the required code and configuration, from the Angular HTML view to the Liquibase changelog.

Those fields cannot contain reserved keywords in the technologies you are using. For example, if you use MySQL:

1. You cannot use Java reserved keywords (as your code will not compile)
2. You cannot use MySQL reserved keywords (as your database schema update will fail)

**Field types**

1. String: A Java String.
2. Integer: A Java Integer.
3. Long: A Java Long.
4. Float: A Java Float.
5. Double: A Java Double.
6. BigDecimal: A java.math.BigDecimal object, used when you want exact mathematic calculations (often used for financial operations).
7. LocalDate: A java.time.LocalDate object, used to correctly manage dates in Java.
8. ZonedDateTime: A java.time.ZonedDateTime object, used to correctly manage dates and times in Java.
9. Boolean: A Java Boolean.
10. Enumeration: A Java Enumeration object. When this type is selected, the sub-generator will ask you what values you want in your enumeration, and it will create a specific enum class to store them.
11. Blob: A Blob object, used to store some binary data. When this type is selected, the sub-generator will ask you if you want to store generic binary data, an image object, or a CLOB (long text). Images will be handled specifically on the Angular side, so they can be displayed to the end-user.

**Validation**

Validation can be set up for each field. Depending on the field type, different validation options will be available.

Validation will be automatically generated on:

1. the HTML views, using the AngularJS validation mechanism
2. the Java domain objects, using Bean Validation

Bean validation will then be used to automatically validate domain objects when they are used in:

1. Spring MVC REST controllers (using the @Valid annotation)
2. Hibernate/JPA (entities are automatically validated before being saved)

Validation information will also be used to generate more precise database column metadata:

1. Required fields will be marked non-nullable
2. Fields which have a maximum length will have the same column length

Validation has a few limitations:

1. We don’t support all validation options from AngularJS and Bean Validation, as we only support those which are common to both APIs
2. Regular Expression patterns don’t work the same in JavaScript and in Java, so if you configure one, you might need to tweak one of the generated patterns
3. JHipster generates unit tests that work for generic entities, without knowing your validation rules: it is possible that the generated tests do not pass the validation rules. In that case, you will need to update the sample values used in your unit tests, so that they pass the validation rules.

**Entity relationships**

Entity relationships are only available for SQL databases.

**Pagination**

Pagination uses the Link header, as in the GitHub API. JHipster provides a custom implementation of this specification on both the server (Spring MVC REST) and client (AngularJS) sides.

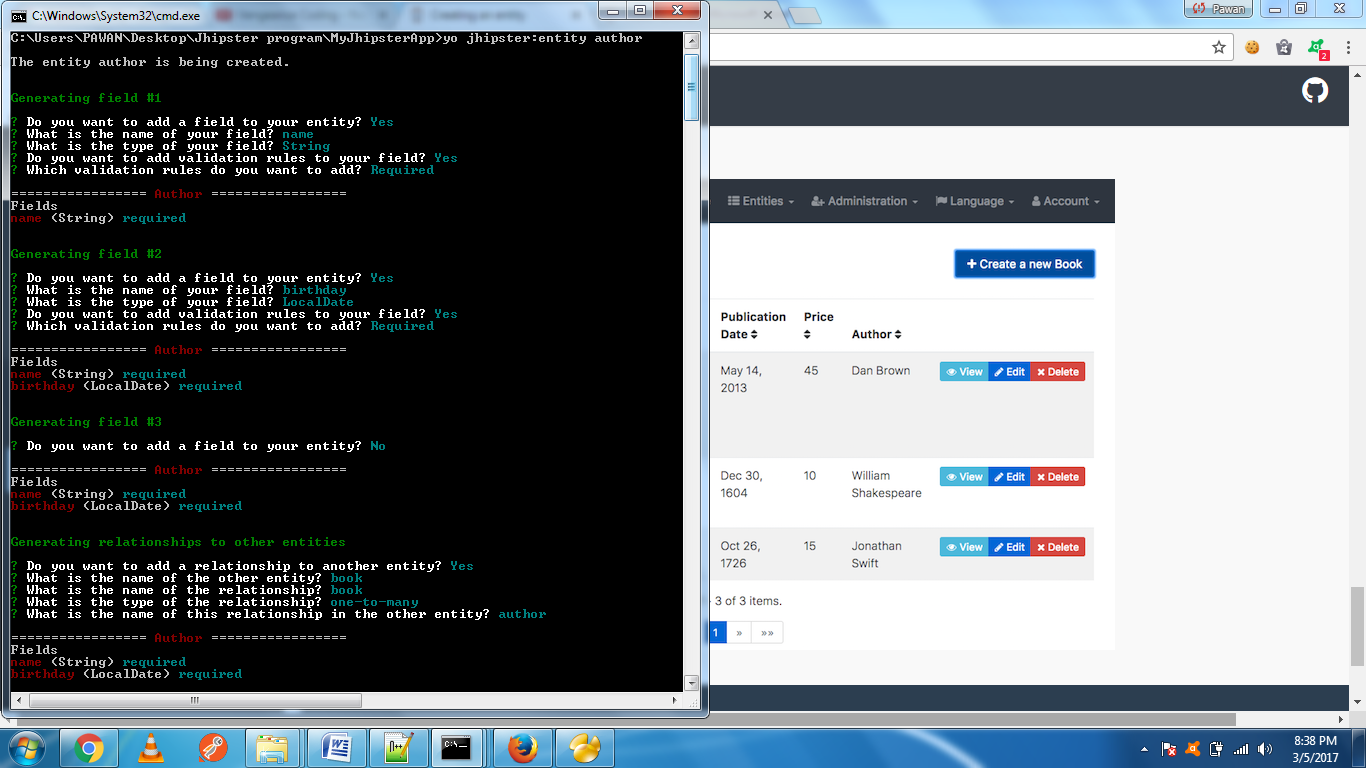
When the entity is generated, JHipster provides 4 pagination options:

1. No pagination (in that case, the back-end won’t be paginated)
2. A simple pager, based on the Bootstrap pager
3. A complete pagination system, based on the Bootstrap pagination component
4. An infinite scroll system, based on the infinite scroll directive

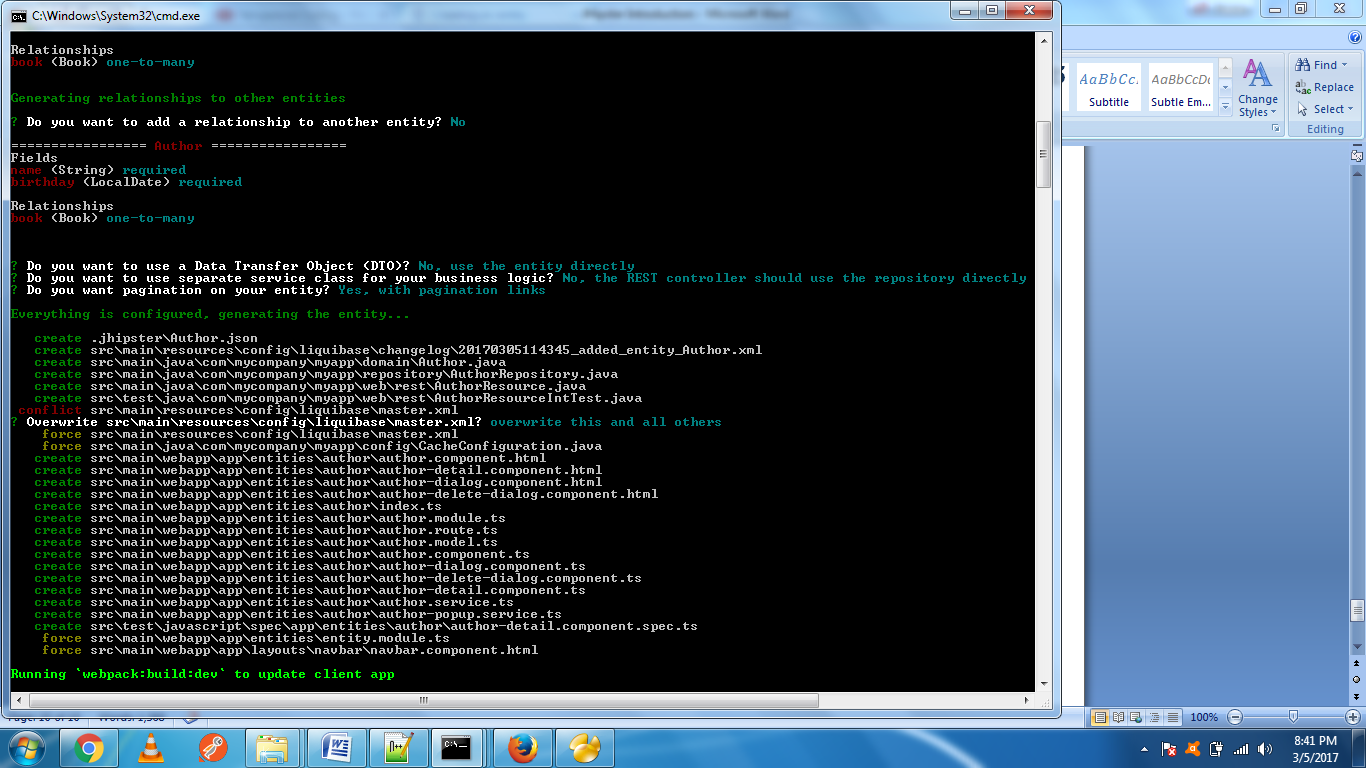
**Updating an existing entity**

The entity configuration is saved in a specific .json file, in the .jhipster directory. So if you run the sub-generator again, using an existing entity name, you can update or regenerate the entity.

**Entity Example**

****

**Overwrite src\main\resources\config\liquibase\master.xml? //a //mean all**

****

Generate the “Book” entity

yo jhipster:entity book

Do the same procedure for book entity as we did above in picture, the book has:

a “title”, of type “String”

a “description”, of type “String”

a “publicationDate”, of type “LocalDate”

a “price”, of type “BigDecimal”

The relationships of the book with author:

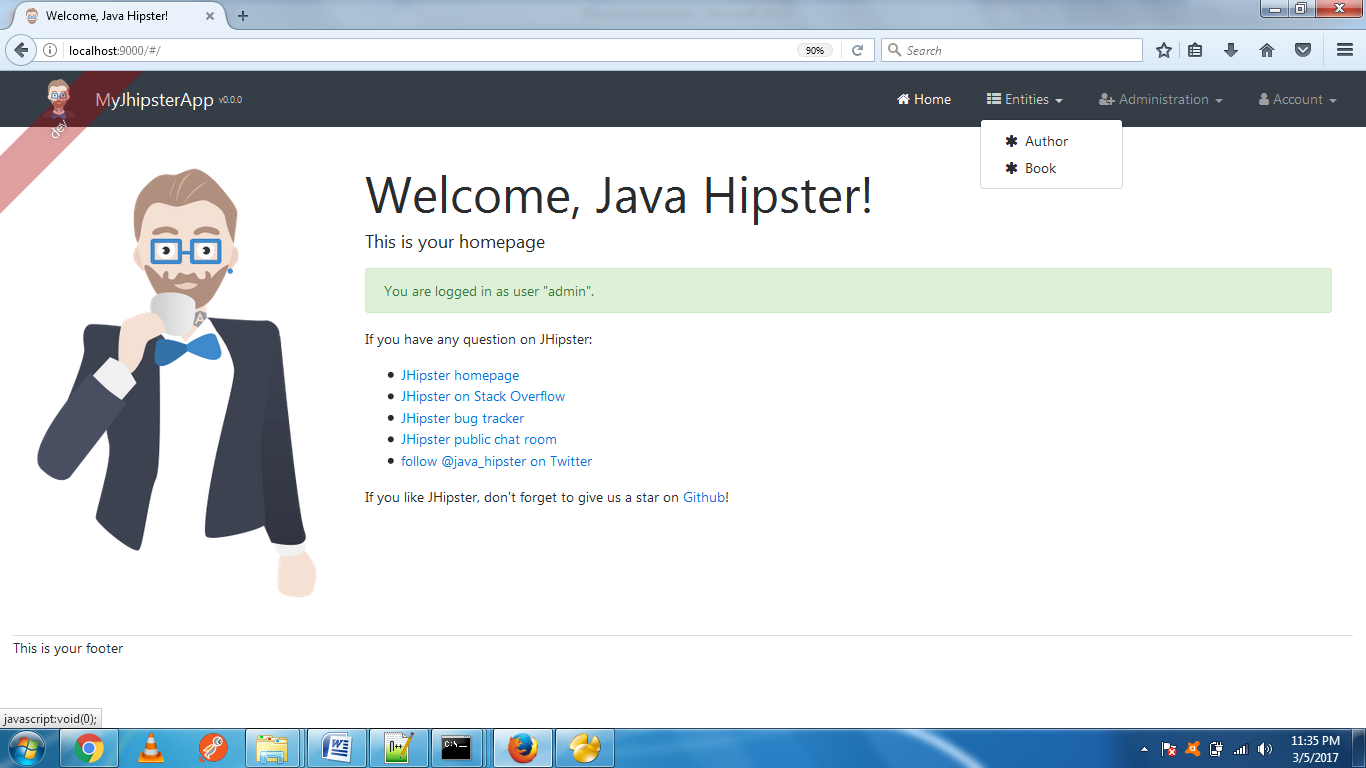
Has many-to-one relationship with the “author” entity

And this relationship uses the “name” field (from the Author entity) to be displayed

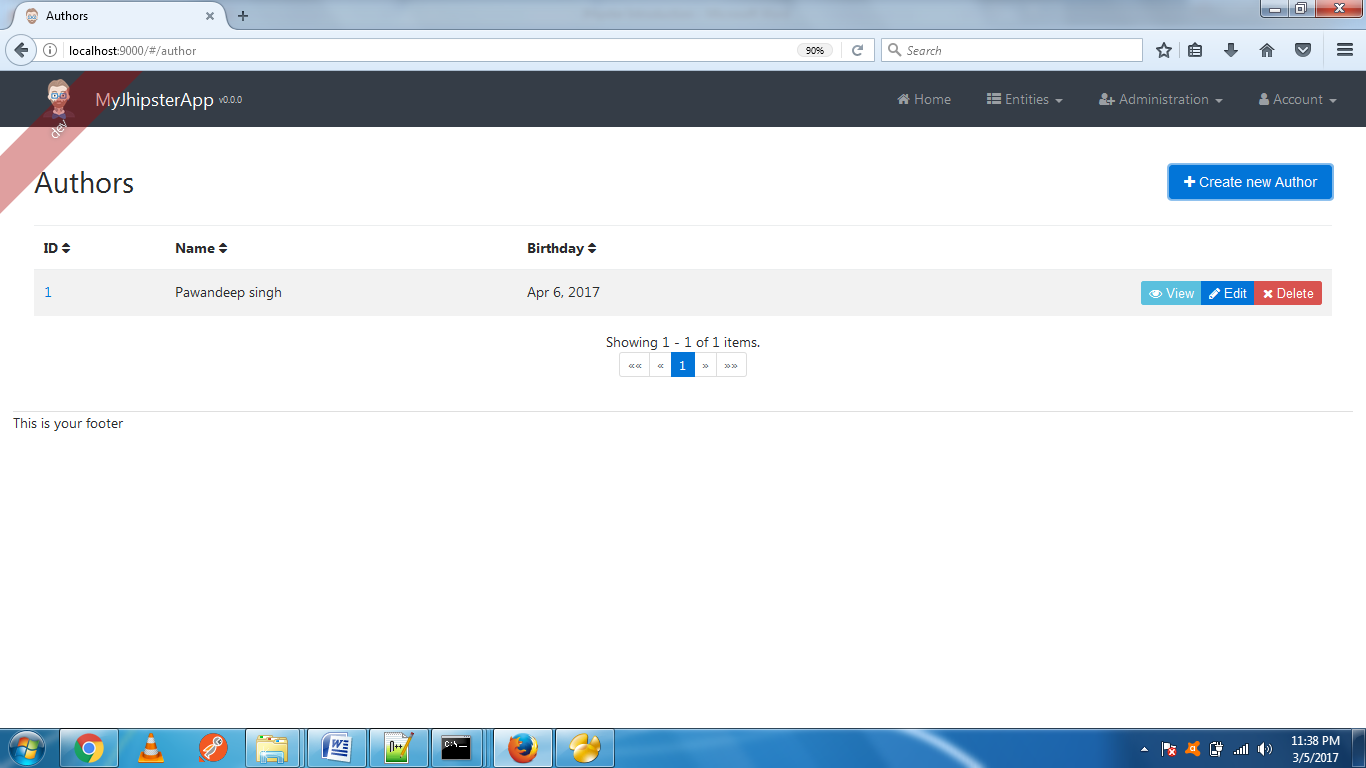
We need to re-run the program with command : mvc spring-boot:run

If every going fine

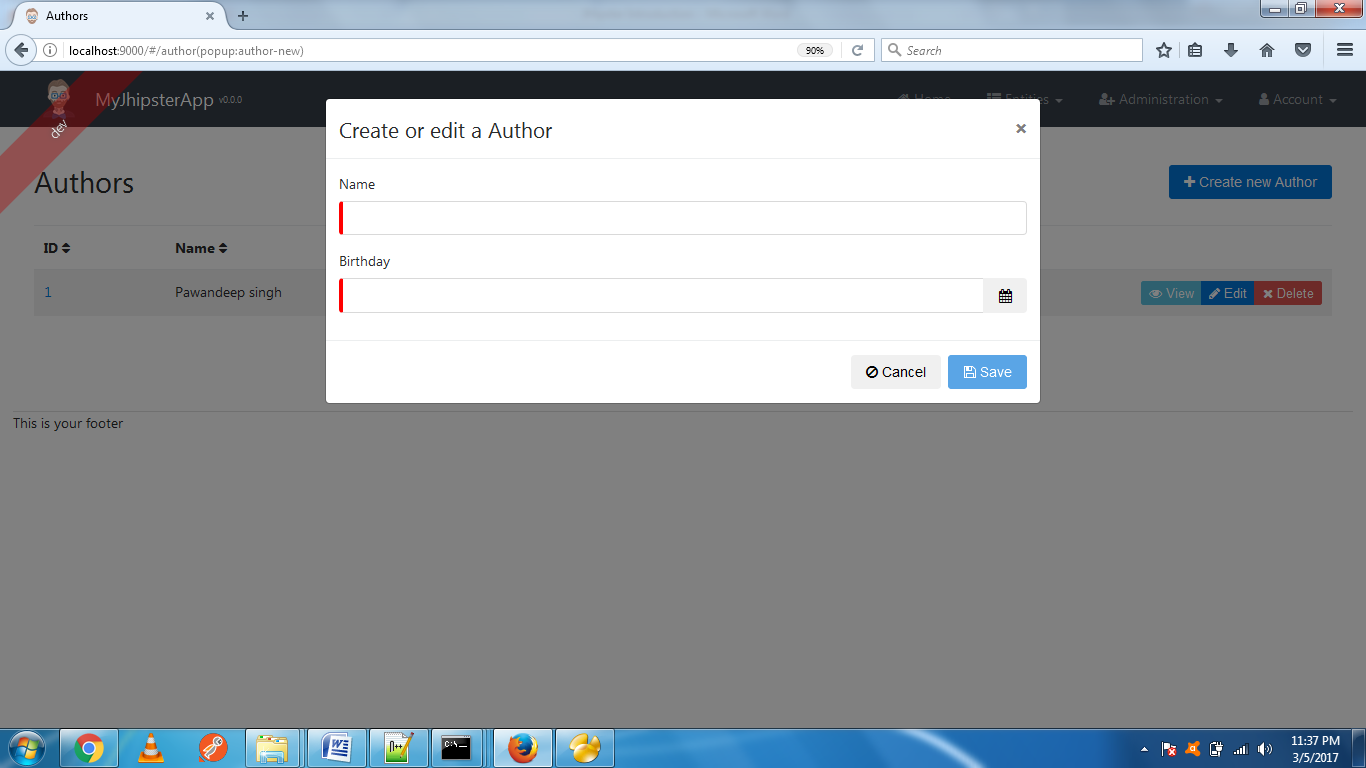
We will get output like bellow figure:



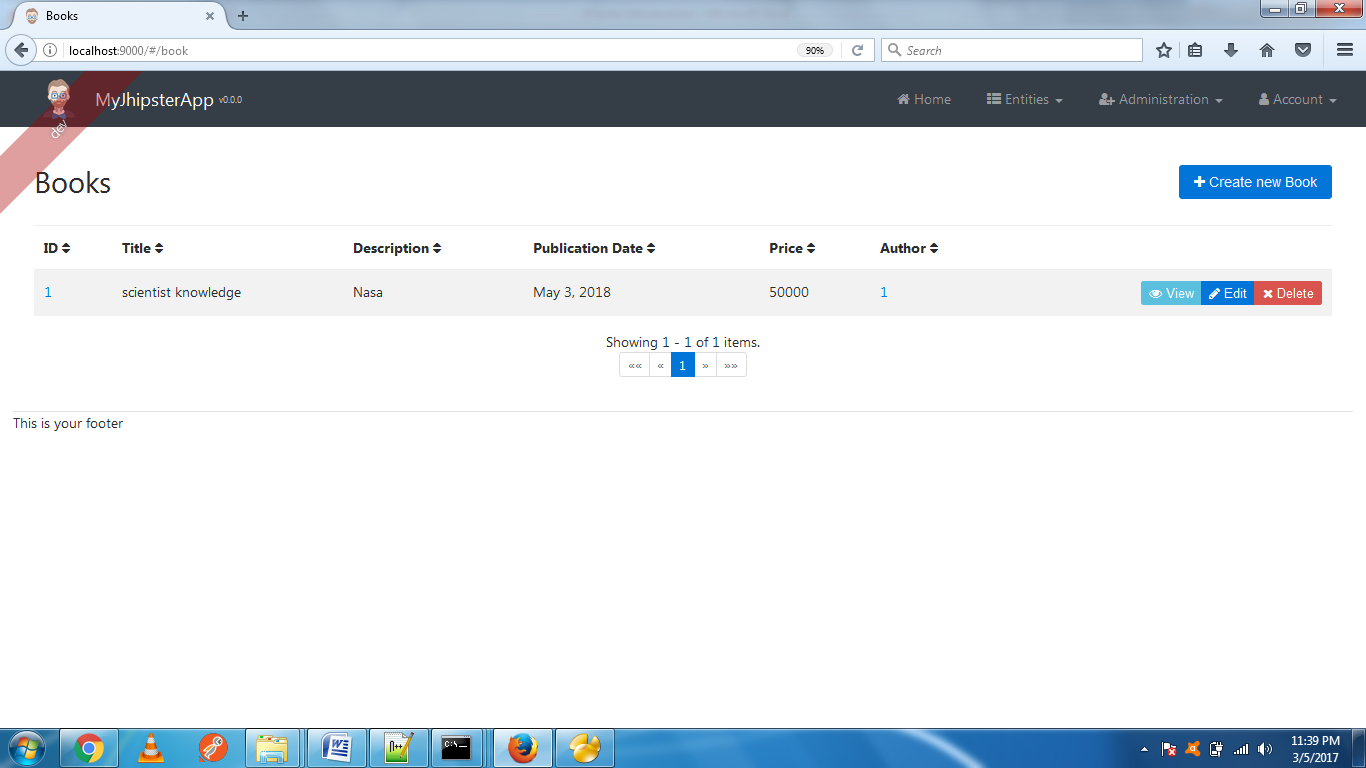
**In Author entity**

****

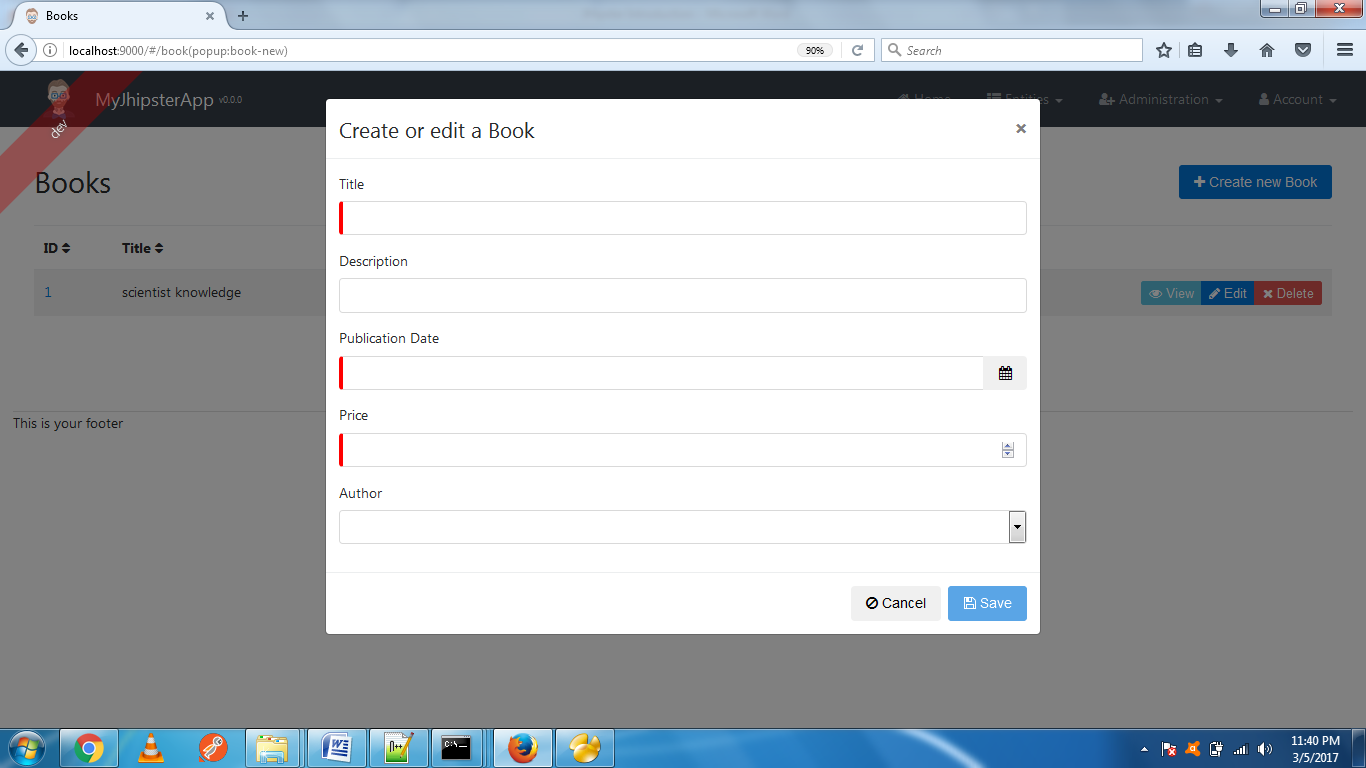
We can create new Author by clicking Create new Author button



In book entity



We can create new book detail by click the create new book button



Note :- After adding new details please check the in sql also.