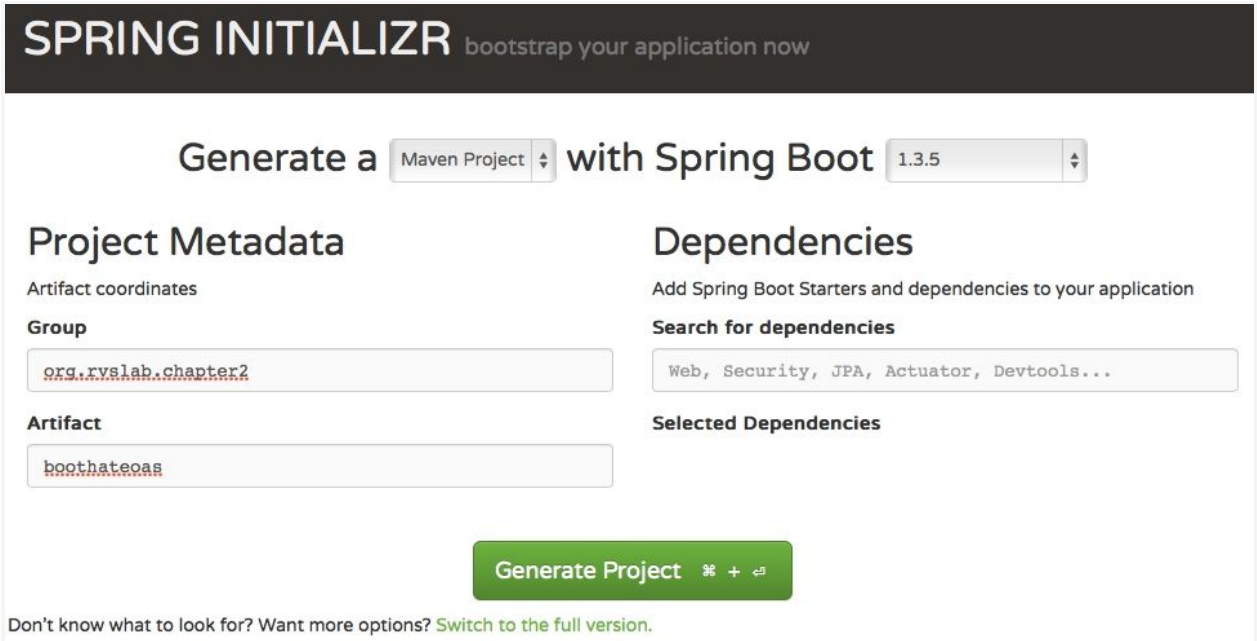
Lab 5 : HATEOAS

In order to use Spring Initilizr, go to https://start.spring.io:



Fill the details, such as whether it is a Maven project, Spring Boot version, group, and artifact ID, as shown earlier, and click on **Switch to the full version** link under the **Generate Project** button. Select **Web**, **HATEOAS**, and **Rest Repositories HAL Browser**. Make sure that the Java version is 8 and the package type is selected as **JAR**:

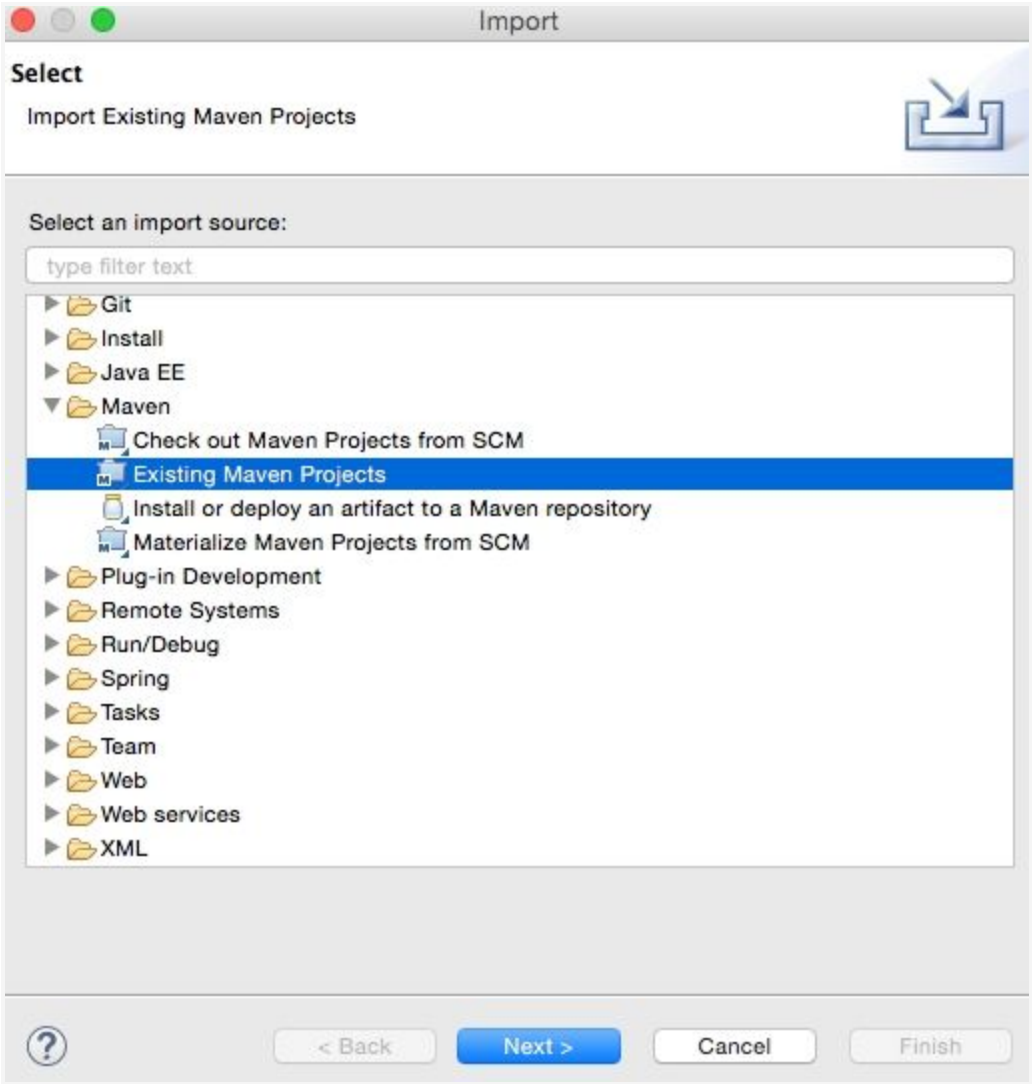
Web

- [ ] **Web**
  Full-stack web development with Tomcat and Spring MVC
- [ ] **Websocket**
  Websocket development with SockJS and STOMP
- [ ] **WS**
  Contract-first SOAP service development with Spring Web Services
- [ ] **Jersey (JAX-RS)**
  the Jersey RESTful Web Services framework
- [ ] **Ratpack**
  Spring Boot integration for the Ratpack framework
- [ ] **Vaadin**
  Vaadin
- [ ] **Rest Repositories**
  Exposing Spring Data repositories over REST via spring-data-rest-webmvc
- [ ] **HATEOAS**
  HATEOAS-based RESTful services
- [ ] **Rest Repositories HAL Browser**
  Browsing Spring Data REST repositories with an HTML UI
- [ ] **Mobile**
  Simplify the development of mobile web applications with spring-mobile
- [ ] **REST Docs**
  Document RESTful services by combining hand-written and auto-generated documentation

Once selected, hit the **Generate Project** button. This will generate a Maven project and download the project as a ZIP file into the download directory of the browser.

Unzip the file and save it to a directory of your choice.

Open STS, go to the **File** menu and click on **Import**:



Navigate to **Maven** | **Existing Maven Projects** and click on **Next**.

Click on **Browse** next to **Root Directory** and select the unzipped folder. Click on **Finish**. This will load the generated Maven project into STS' **Project Explorer**.

Edit the `Application.java` file to add a new REST endpoint, as follows:

Copy

```
@RequestMapping("/greeting")
@ResponseBody
public HttpEntity<Greet> greeting(@RequestParam(value = "name", required = false,
defaultValue = "HATEOAS") String name) {
        Greet greet = new Greet("Hello " + name);

greet.add(linkTo(methodOn(GreetingController.class).greeting(name)).withSelfRel());
```

```
        return new ResponseEntity<Greet>(greet, HttpStatus.OK);
}
```

Note that this is the same `GreetingController` class as in the previous example. However, a method was added this time named `greeting`. In this new method, an additional optional request parameter is defined and defaulted to `HATEOAS`. The following code adds a link to the resulting JSON code. In this case, it adds the link to the same API:

Copy

```
greet.add(linkTo(methodOn(GreetingController.class).greeting(name)).withSelfRel());
```

In order to do this, we need to extend the `Greet` class from `ResourceSupport`, as shown here. The rest of the code remains the same:

Copy

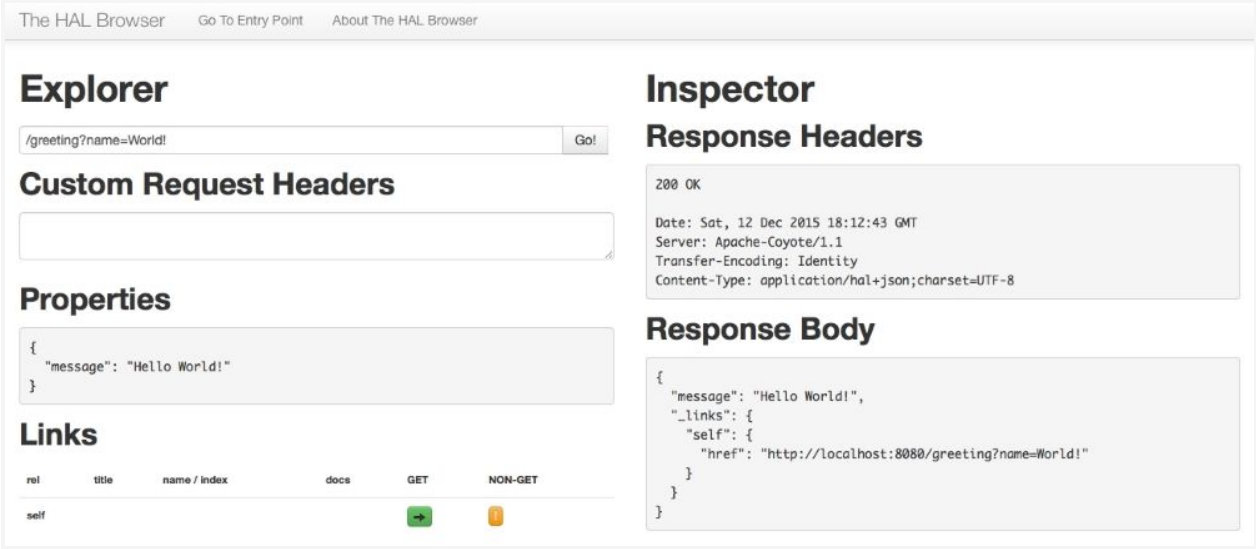
```
class Greet extends ResourceSupport{
```

The `add` method is a method in `ResourceSupport`. The `linkTo` and `methodOn` methods are static methods of `ControllerLinkBuilder`, a utility class for creating links on controller classes. The `methodOn` method will do a dummy method invocation, and `linkTo` will create a link to the controller class. In this case, we will use `withSelfRel` to point it to itself.

This will essentially produce a link, `/greeting?name=HATEOAS`, by default. A client can read the link and initiate another call.

Run this as a Spring Boot app. Once the server startup is complete, point the browser to `http://localhost:8080`.

This will open the HAL browser window. In the **Explorer** field, type `/greeting?name=World!` and click on the **Go** button. If everything is fine, the HAL browser will show the response details as shown in the following screenshot:



As shown in the screenshot, the **Response Body** section has the result with a link with `href` pointing back to the same service. This is because we pointed the reference to itself. Also, review the **Links** section. The little green box against **self** is the navigable link.

It does not make much sense in this simple example, but this could be handy in larger applications with many related entities. Using the links provided, the client can easily navigate back and forth between these entities with ease.