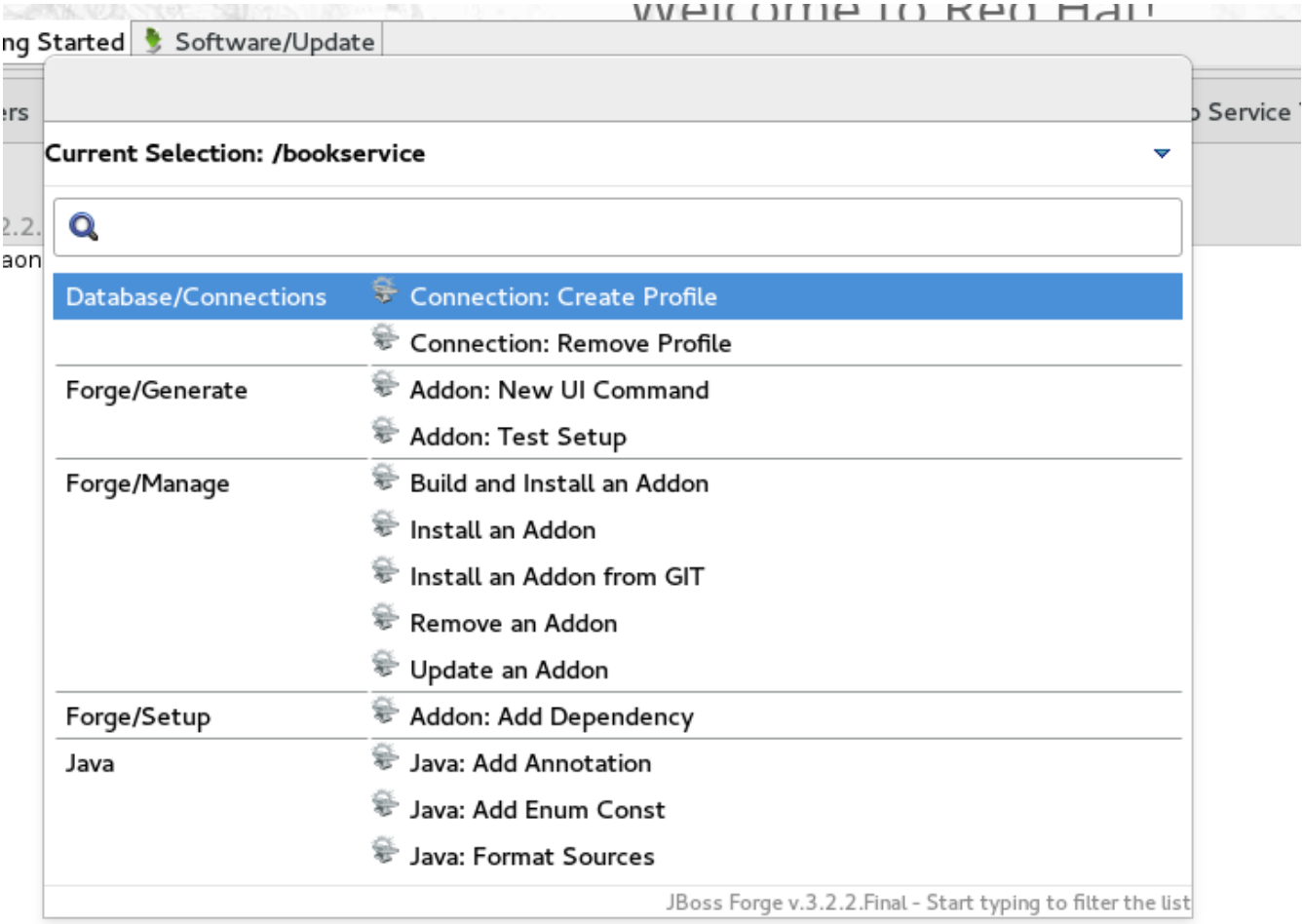
Devoxx.FR - Hands-on Labs

Table of Content

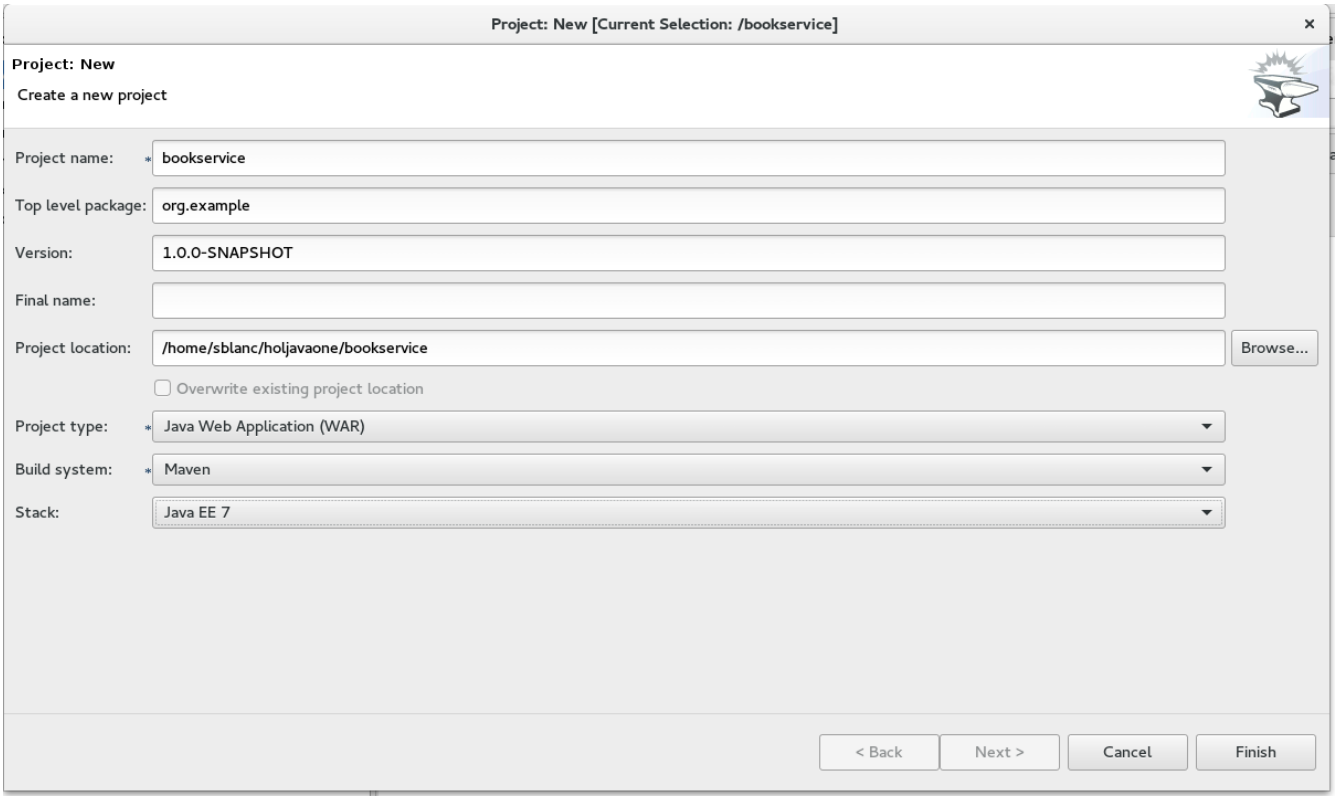
1. Starting the Forge Wizard

Press `Ctrl + 4` (`Cmd+4` on MacOS):



You can also use at any time the Forge Console.
If the Forge console is not visible in your eclipse workspace, you can open it by navigating to Window → Show View → Other → Forge → Forge Console.

2. Creating the BookService



For each command you will also see the console equivalent, choose whatever you want (UI or console)

```
project-new --named bookservice --stack JAVA_EE_7
```

Look at the structure of the created project, you will see it's a classic Maven Java project

From now, in the screenshots don't take into account the package name since it's different from yours. Stick to `org.bookservice.`

2.1. Creating the model

Let's start to define our domain objects : our bookService will have an `author` and a `book`.

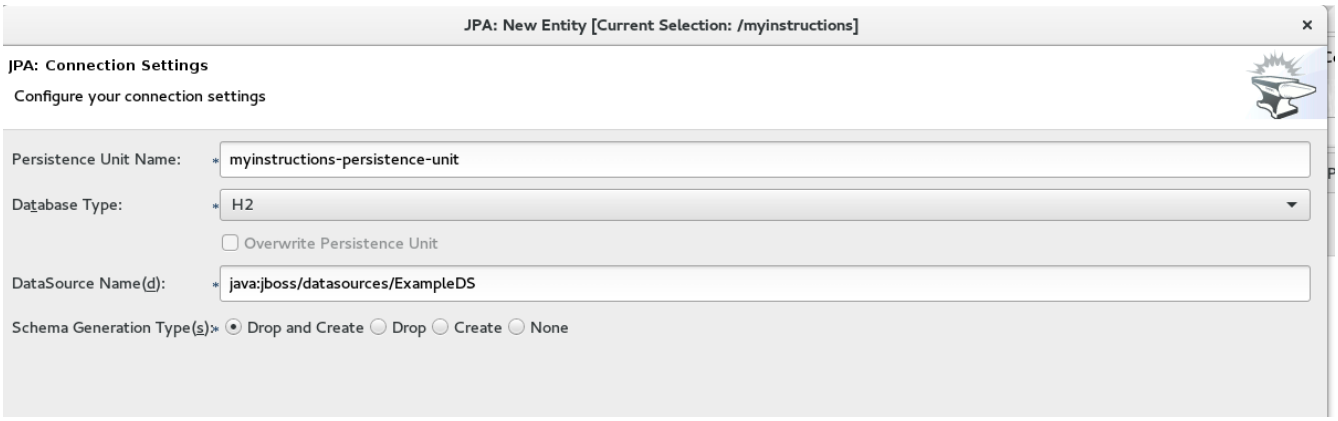
2.1.1. Creating the Author entity

In the Forge UI, type `jpa: new entity`

Since it's our first JPA entity, we need to setup our persistence layer :

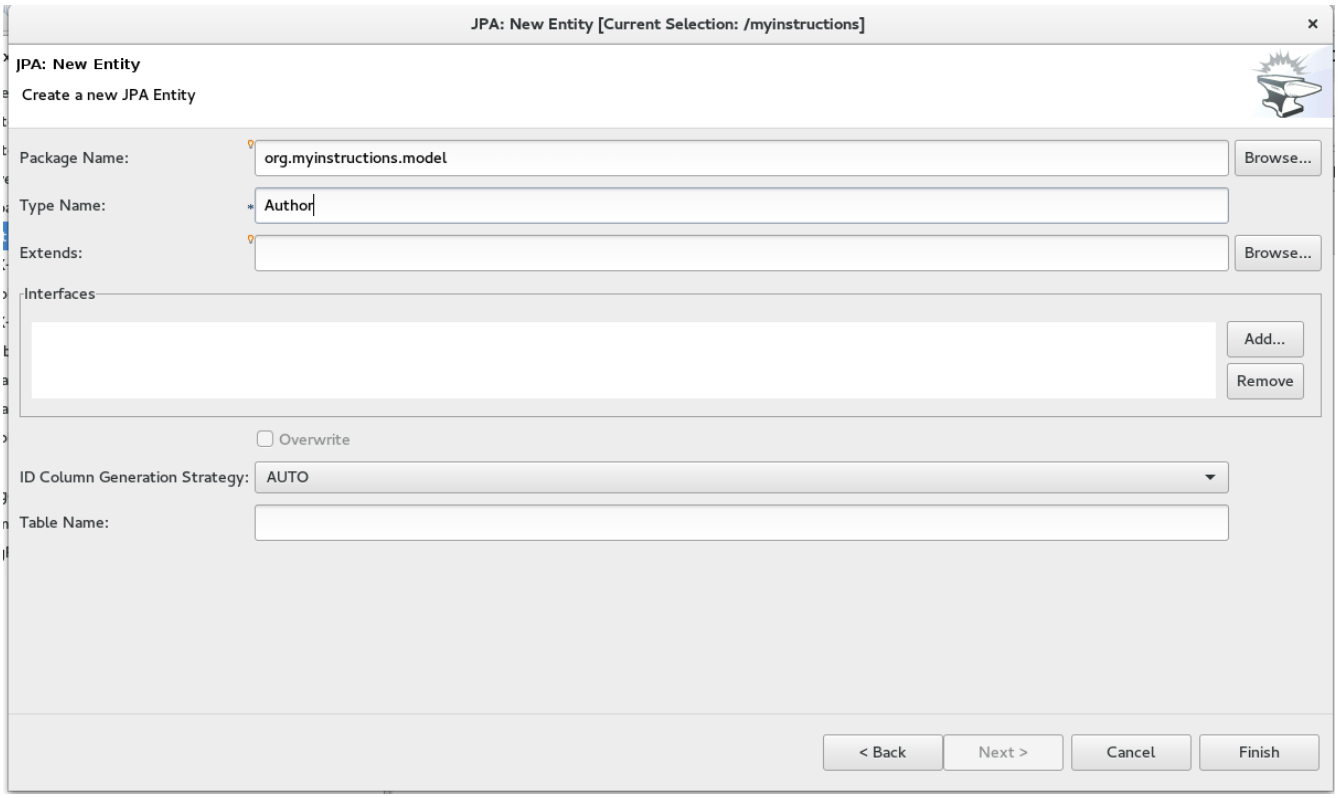
You can keep all the default :

We go for `hibernate`



For this workshop, we will be using the `h2` database.

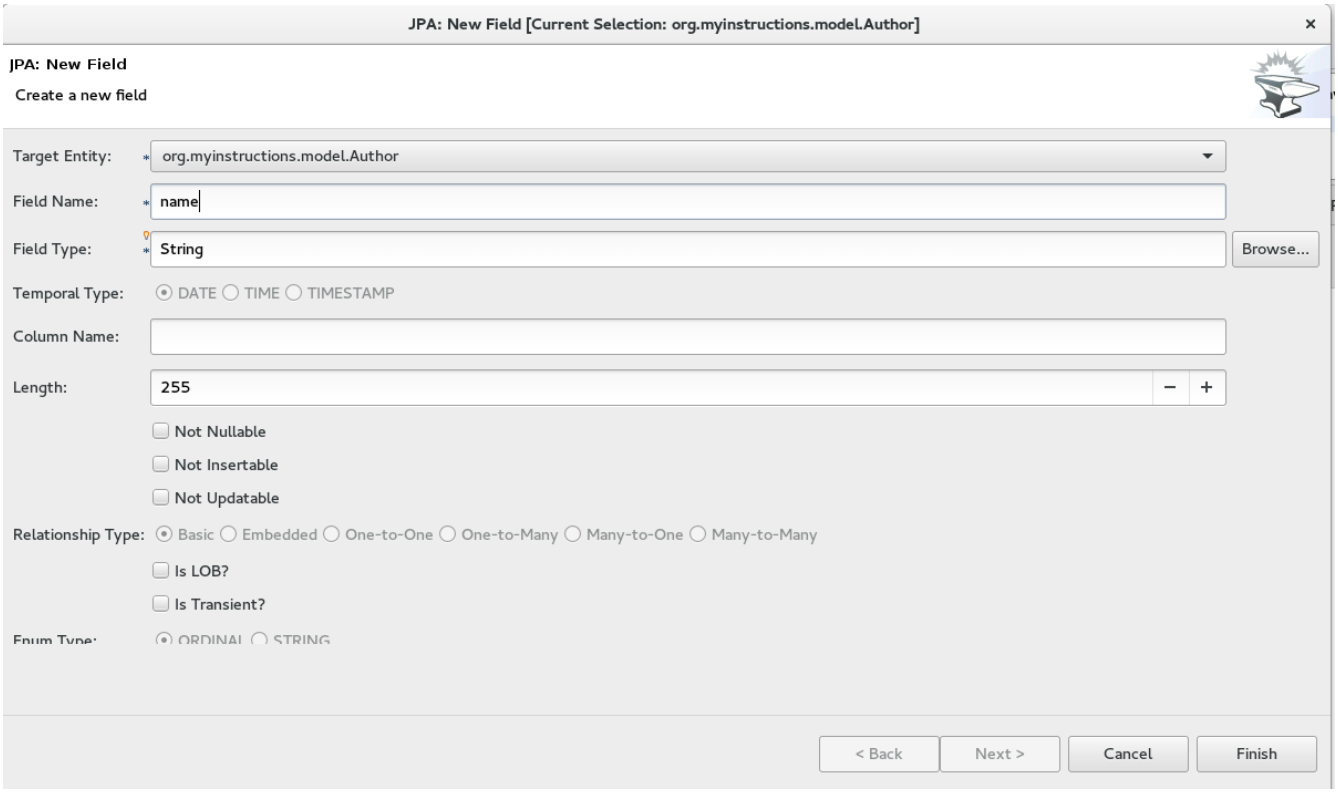
Then finally we can create our `author` entity




```
jpa-new-entity --named Author
```

Look at the class that has been generated. Let's add a field now :

`jpa: new field`



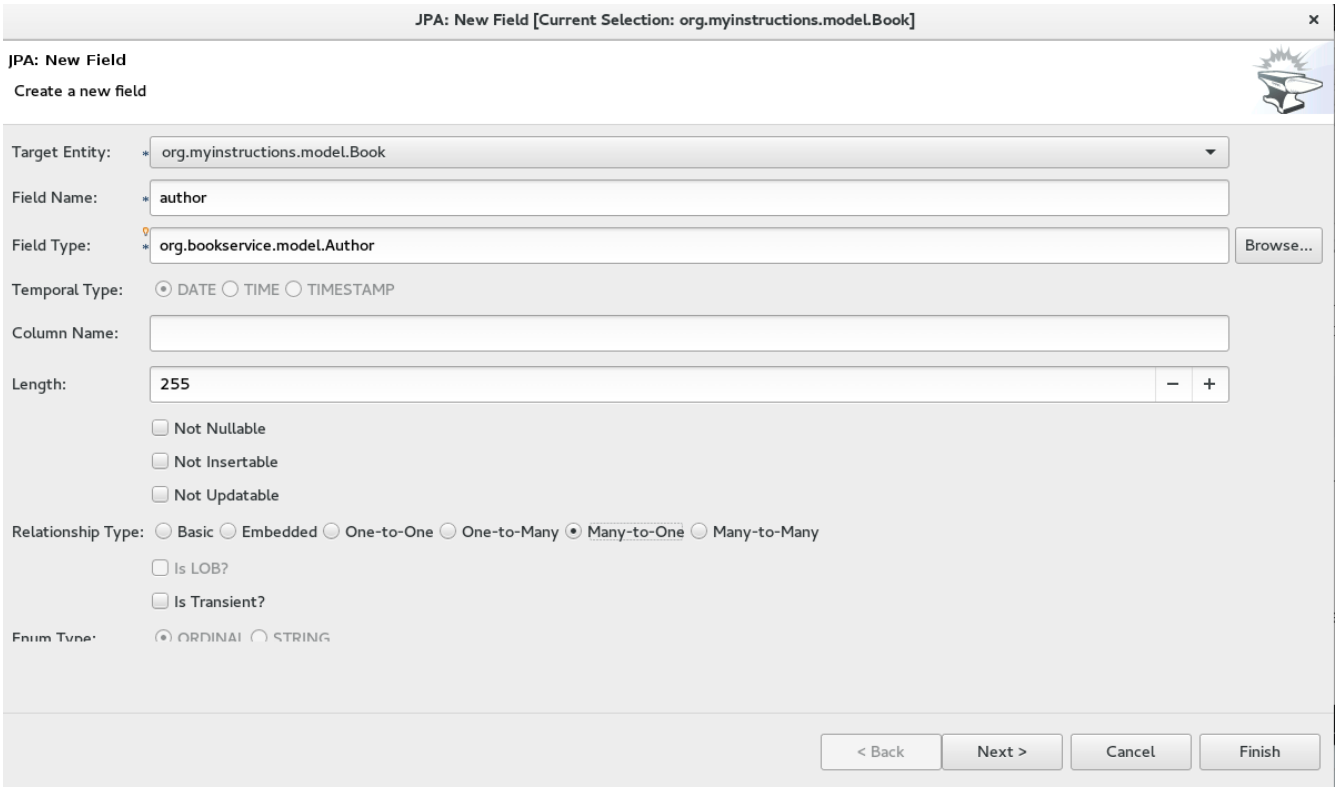

```
jpa-new-field --named name
```

2.1.2. Creating the Book entity

Like for the previous step, create a `Book` entity and add the following fields :

- `title`
- `isbn`

The third field is a bit more complex since it will be a `Many-to-One` relationship with `author`




```
jpa-new-field --named author --type org.bookservice.model.Author --relationship-type
Many-to-One
```

Observe the entities and the different annotations.

2.1.3. Creating the SellingPoint entity

You are now an expert to create JPA entities, so I just give you the console commands :

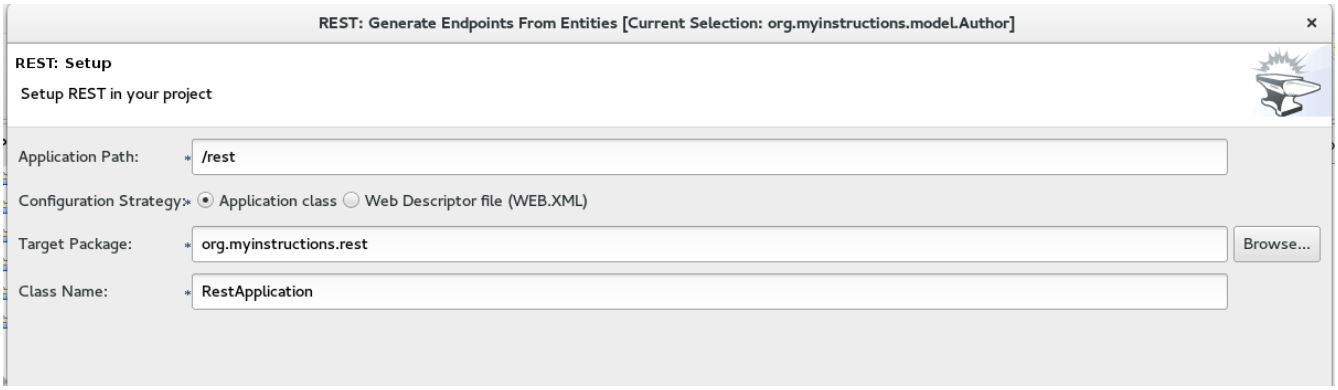
```
jpa-new-entity --named SellingPoint
jpa-new-field --named name
jpa-new-field --named latitude --type Double
jpa-new-field --named longitude --type Double
```

2.2. Creating the REST endpoints

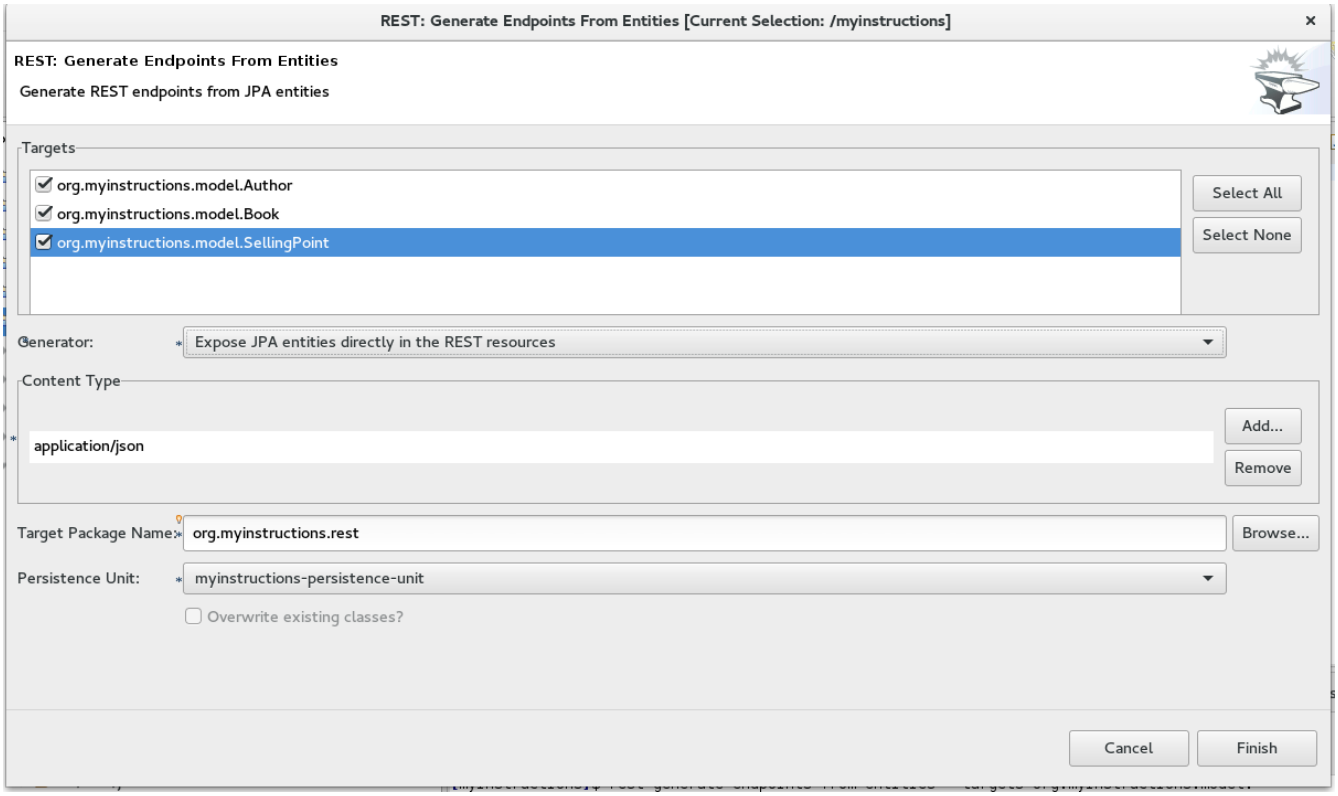
Now that we have defined our model, let's expose the CRUD (Create Read Update Delete) APIs and expose it as REST Webservice.

```
rest: generate ⋯
```

Like for the persistence layer, since it's the first time, you need to setup the REST layer, you can keep the defaults :



Then, select your entities :




```
rest-generate-endpoints-from-entities --targets org.bookservice.model.*
```

Look at the generated endpoints.

2.3. Deploy our Bookservice

We have now enough stuff to do a first deployment ! A Wildfly 10 Server has already been configured for you.

2.3.1. Add bootstrap data

Let's add a small `sql` script that will create some entries when we start our app. Inside the `src/main/resources/` create a file named `import.sql` and add this :

```
insert into Author  (id, name, version) values (1000,'Seb',0);
insert into Author  (id, name, version) values (1001,'George',0);

insert into Book  (id, title, isbn, author_id, version) values (1000,'Forge for Pro',
'1001', 1000, 0);
insert into Book  (id, title, isbn, author_id, version) values (1001,'Swarm for Pro',
'1002', 1001, 0);
```

2.3.2. Deploy with Eclipse

- Right click on the project and choose `run as\run on Server`

Just follow the instructions, and you can just hit "next" for each step.

2.3.3. Deploy manually

- Open a terminal
- go to your project : `cd workspace/bookservice`
- Build the project : `mvn clean package`
- Open a second terminal
- Go to Wildfly server bin folder: `cd /home/wildfly-10.0.0.Final`
- Start the server : `./standalone.sh`
- Copy WAR file to the deployment folder : `cp /home/workspace/bookservice/target/bookservice.war /home/wildfly-10.0.0.Final/standalone/deployments`

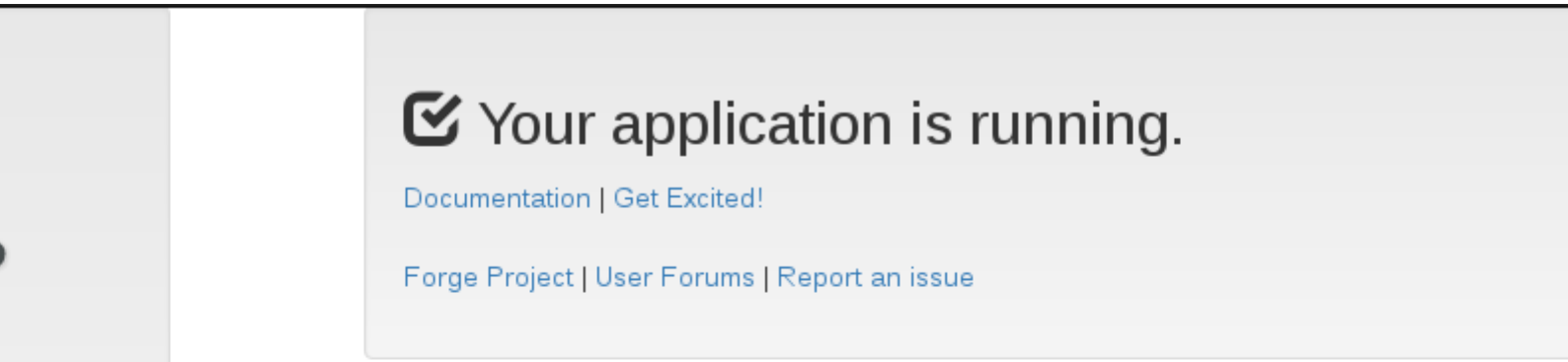
The `bookservice` should now be deployed, browse to `http://localhost:8080/bookservice/rest/authors` it should returns an array with 2 authors.

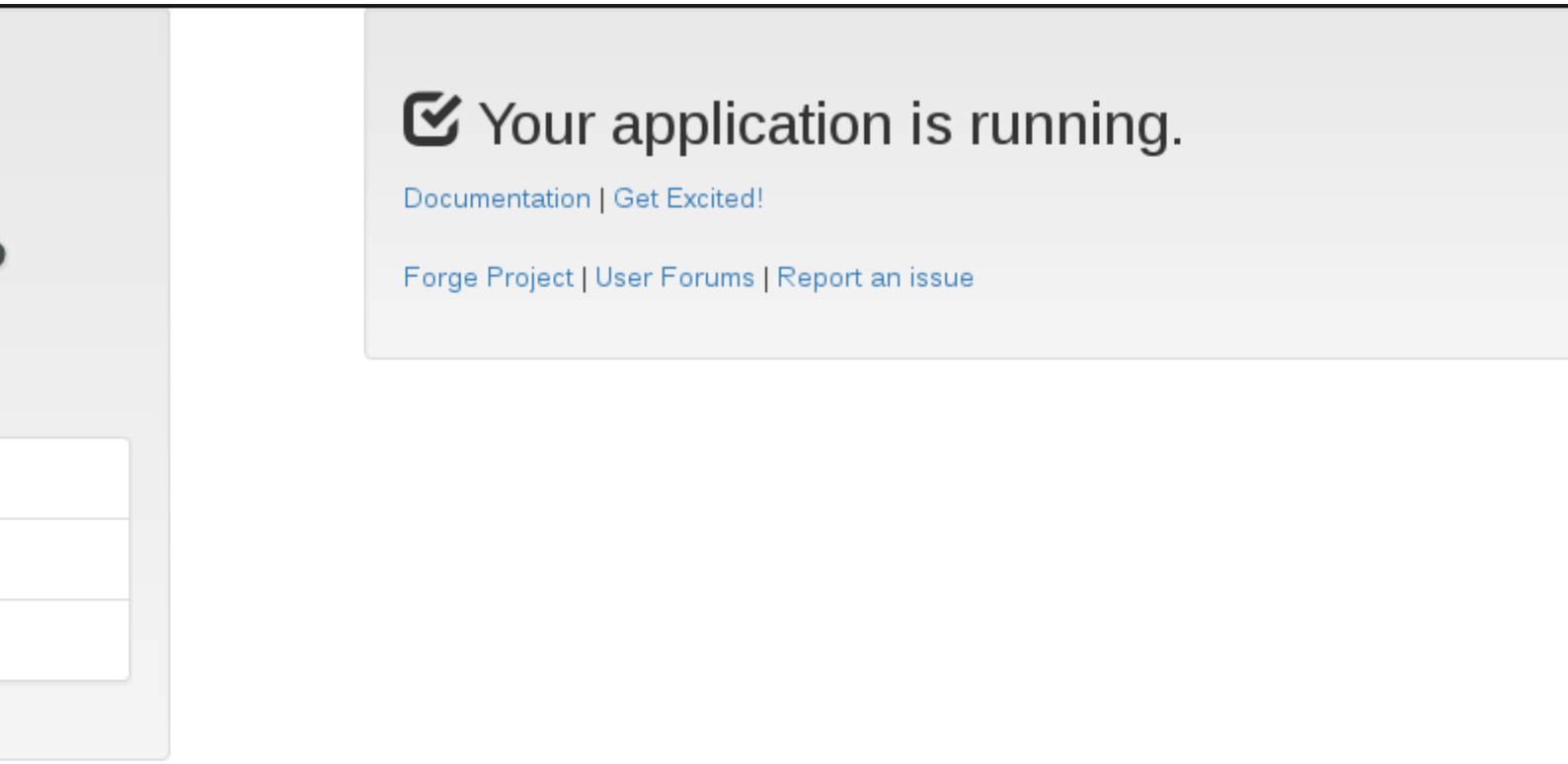
2.4. Scaffolding the front-end

Let's create a CRUD Web client, with Forge it's really easy :

- Start the Forge UI and search for `scaffold: generate`.
- Choose for `AngularJS` for the Scaffold Type.
- Select all the entities

You're done ! Build the app and deploy it again, now browse to `http://localhost:8080/bookservice`


Your application is running.
Documentation | Get Excited!
Forge Project | User Forums | Report an issue



Play a bit around, try all the CRUD operations. Look also at the generated frontend scripts that are using AngularJS.

3. Decomposing our App into Microservices

Now, let's decompose this application into 3 different microservices :

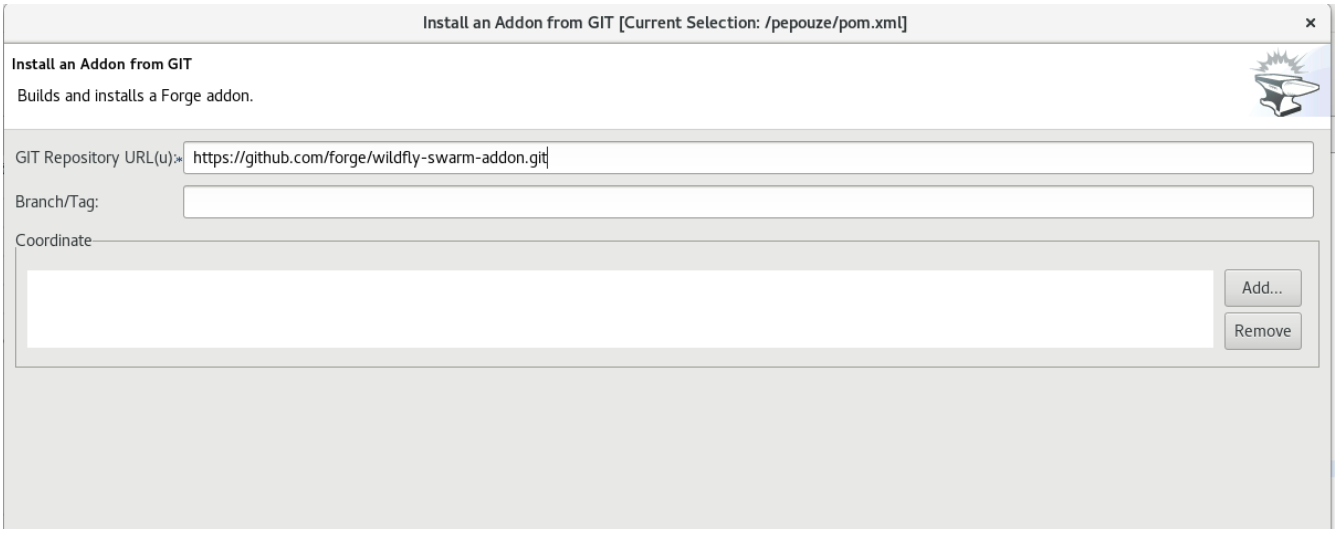
- The book Service
- The SellingPoint Service
- The Front-end

Let's start with turning our app into a Swarm Microsevice

3.1. Bookservice Swarm application

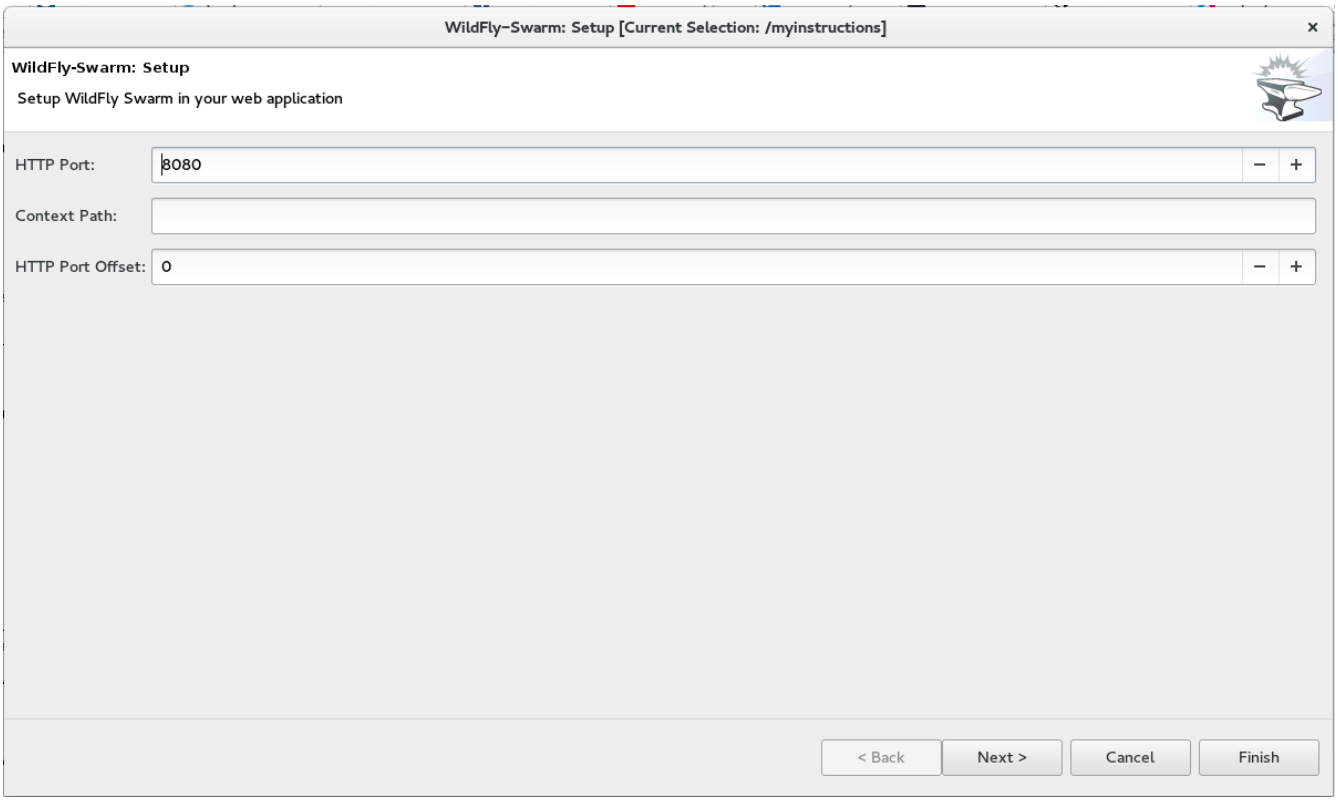
3.1.1. Add the Swarm Addon

Simple call the `Install an Addon from GIT` command :




```
addon-install-from-git --url https://github.com/forge/wildfly-swarm-addon.git
```

To turn our app into a Swarm app it's really easy, from the Forge UI search for `Wildfly-Swarm: Setup` , keep the default and click `finish`

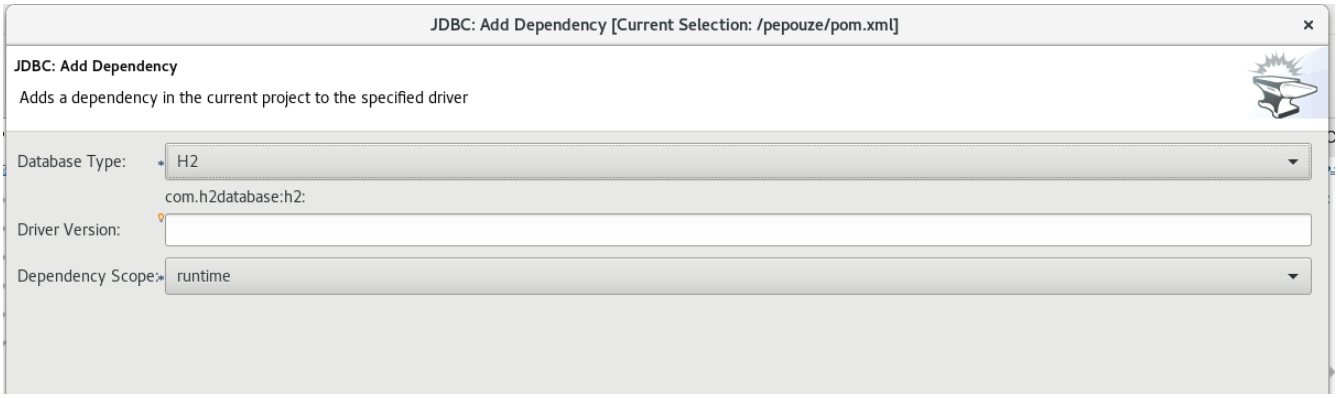


Inspect your `pom.xml` to see the changes.

```
wildfly-swarm-setup
```

3.1.2. Add JDBC Driver

Let's add the H2 JDBC Driver, with the command `JDBC Add Dependency`

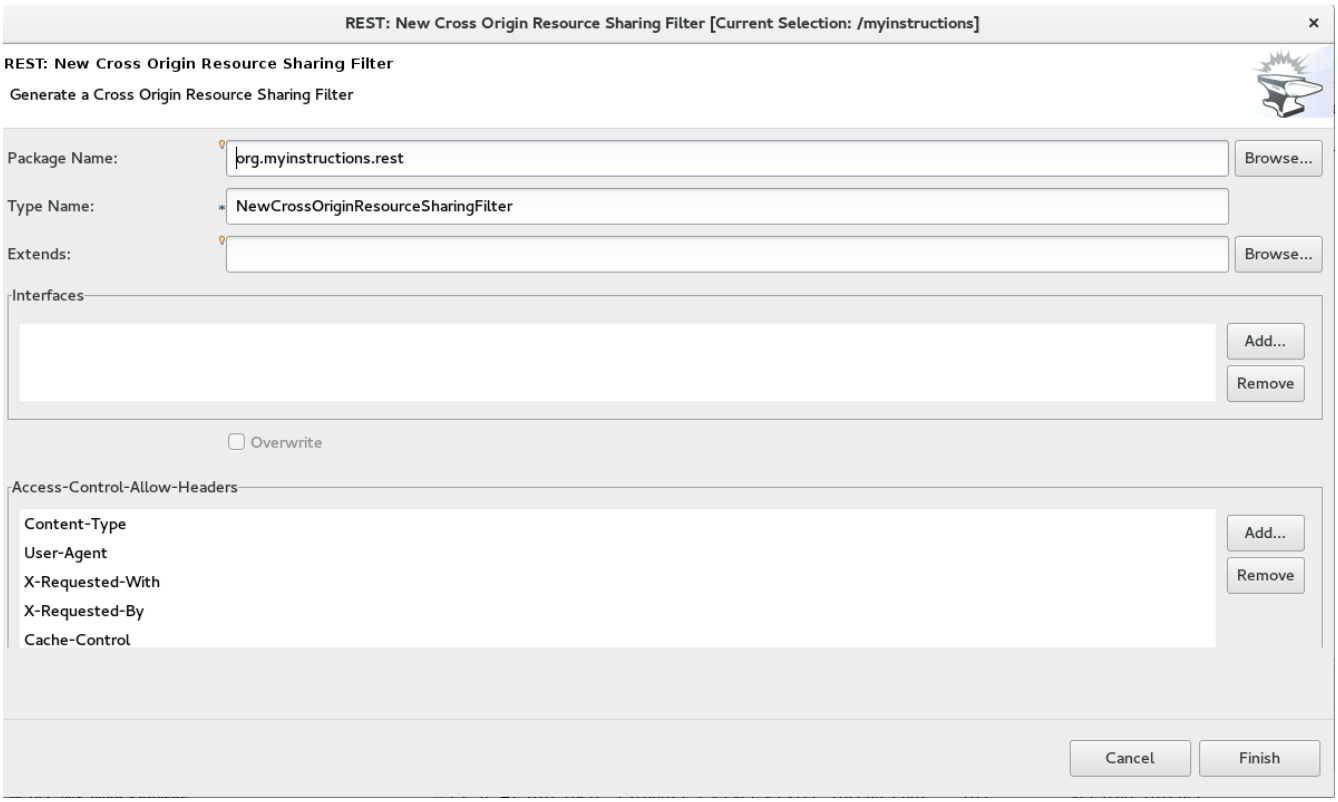



```
jdbc-add-dependency --db-type H2 --scope runtime
```

3.1.3. Enable CORS

CORS is W3C recommendation to allow browser requesting resource from another domain.

Our microservice will be consumed from other domains, therefore we need to enable CORS, that's also easy with Forge, search for `REST: New Cross Origin Resource Sharing Filter` :




```
rest-new-cross-origin-resource-sharing-filter
```

3.1.4. Run Swarm app

You have two options from a terminal :

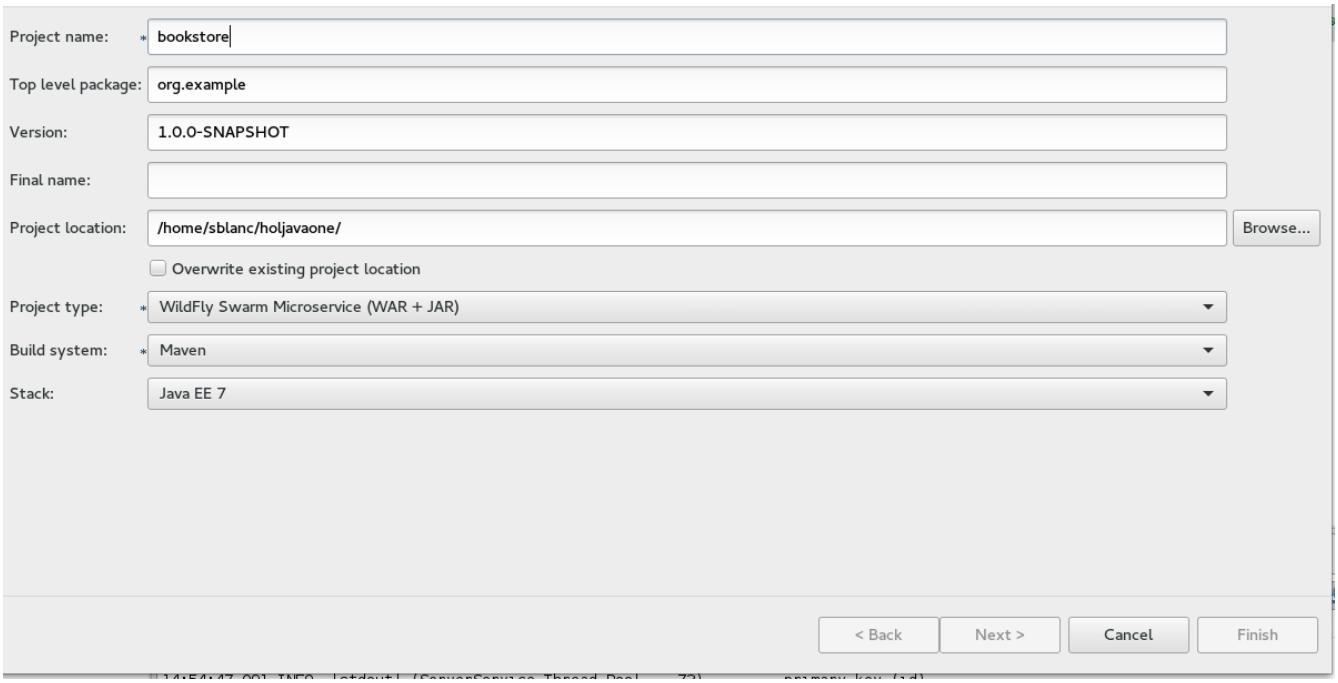
- `mvn clean wildfly-swarm:run`
- Or first build the app with `mvn clean package` and after that run the JAR `java -jar target/bookservice-swarm.jar`

Ok, the back-end is now running as a microservice, let's extract the front-end into a standalone microservice as well.

3.2. Front-end Swarm application

Create a new project with Forge called `bookstore` and for Project type choose "Wildfly Swarm Microservice", for the stack choose "JavaEE 7".

For the project location, be sure to be in the workspace root : `/home/osboxes/workspace`



Then you have the `Swarm` setup wizard, set the port to `8081` and on the next page select the fraction `undertow`.

Undertow is a flexible performant web server written in java, providing both blocking and non-blocking API's based on NIO.

```
project-new --named bookstorefrontend --stack JAVA_EE_7 --type wildfly-swarm --http
-port 8081
wildfly-swarm-add-fraction --fractions undertow
```

Now we need to copy, the contents of the `src/main/webapp` folder from our first project `bookservice` into the `src/main/webapp` of the `bookstore` project. You can do this using the Eclipse explorer or by running this forge command :

```
mv ../bookservice/src/main/webapp/ src/main/
```

3.2.1. Update the front-end AngularJS Rest services

Our front-end must now call a remote REST service, open first :

`src/main/webapp/scripts/services/AuthorFactory.js`

find where we specify the URL and replace it with `http://localhost:8080/rest/authors/:AuthorId`.

Do the same for `BookFactory.js`, we will change `SellingPointFactory.js` later.

3.2.2. Deploy the app

- `mvn clean wildfly-swarm:run`
- Or first build the app with `mvn clean package` and after that run the JAR `java -jar target/bookstore-swarm.jar`

You can browse to `http://localhost:8081` to make sure everything is running.

3.3. SellingPoint Swarm App

Let's rewrite the SellingPoint service from scratch and let's add some cool features like `hibernate-search` and `geospatial queries`. Now that you know how Forge works I just give you the script, run each line or use the UI, it's up to you.

Forge may not add import statements, so be sure to do that manually if that happens ;)

If the Forge console is not visible in your eclipse workspace, you can open it by navigating to Window → Show View → Other → Forge → Forge Console.

```
project-new --named sellingPoint --stack JAVA_EE_7 --type wildfly-swarm --http-port
8082
wildfly-swarm-add-fraction --fractions hibernate-search datasources ejb jpa jaxrs
# create Book entity and relationship with Author
jpa-new-entity --named Book
jpa-new-field --named isbn
java-add-annotation --annotation org.hibernate.search.annotations.Field --on-property
isbn

# create Book entity and relationship with Author
jpa-new-entity --named SellingPoint
jpa-new-field --named name
java-add-annotation --annotation org.hibernate.search.annotations.Indexed
java-add-annotation --annotation org.hibernate.search.annotations.Spatial
jpa-new-field --named latitude --type Double
jpa-new-field --named longitude --type Double
java-add-annotation --annotation org.hibernate.search.annotations.Longitude --on
-property longitude
java-add-annotation --annotation org.hibernate.search.annotations.Latitude --on
-property latitude
jpa-new-field --named books --type org.sellingPoint.model.Book --relationship-type
Many-to-Many --fetch-type EAGER
java-add-annotation --annotation org.hibernate.search.annotations.IndexedEmbedded --on
-property books

scaffold-generate --provider AngularJS --generate-rest-resources --targets
org.sellingPoint.model.*

# enable CORS
rest-new-cross-origin-resource-sharing-filter

jdbc-add-dependency --db-type H2 --scope runtime
```

3.3.1. Add our geolocation search endpoint

In `src/main/java/org/sellingPoint/rest/SellingPointEndpoint.java` , create a new method :

```
@GET
@Path("/inrange/{isbn}")
@Produces("application/json")
public List<SellingPoint> listByLocation(@PathParam("isbn") String isbn,
@QueryParam("latitude") Double latitude,
    @QueryParam("longitude") Double longitude) {
  FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(em);
①
  QueryBuilder builder = fullTextEntityManager.getSearchFactory().buildQueryBuilder()
②
      .forEntity(SellingPoint.class).get();

  org.apache.lucene.search.Query luceneQuery = builder

      .spatial().within(5,
Unit.KM).ofLatitude(latitude).andLongitude(longitude).createQuery();
  org.apache.lucene.search.Query keyWordQuery = builder

      .keyword().onField("books.isbn").matching(isbn).createQuery();
  org.apache.lucene.search.Query boolQuery =
builder.bool().must(luceneQuery).must(keyWordQuery).createQuery();
  javax.persistence.Query hibQuery =
fullTextEntityManager.createFullTextQuery(boolQuery, SellingPoint.class);
  return hibQuery.getResultList();
}
```

① `org.hibernate.search.jpa.FullTextEntityManager` and `org.hibernate.search.jpa.Search`

② `org.hibernate.search.query.dsl.QueryBuilder`

Don't forget the imports !

3.3.2. Add initial bootstrap data

Let's add a small `sql` script that will create some entries when we start our app. Inside the `src/main/resources/` create a file named `import.sql` and add this :

```
insert into Book (id, isbn, version) values (1000, '1000',0);
insert into Book (id, isbn, version) values (1001, '1001',0);
insert into Book (id, isbn, version) values (1002, '1002',0);

insert into SellingPoint (id, latitude, longitude, name, version) values (2000,
43.5780, 7.0545, 'bob', 0);
insert into SellingPoint (id, latitude, longitude, name, version) values (2001,
43.574357, 7.1142449, 'chris',0);

insert into SellingPoint_Book (SellingPoint_id, books_id) values (2000,1000);
insert into SellingPoint_Book (SellingPoint_id, books_id) values (2000,1001);
```

Please change the longitude and latitude depending on where you are.

3.3.3. Deploy the app

- `mvn clean wildfly-swarm:run`
- Or first build the app with `mvn clean package` and after that run the JAR `java -jar target/sellingpoint-swarm.jar`

3.3.4. Updating the front-end app to consume the SellingPoint Service

In the `bookstore` app go to `src/main/webapp/scripts/services/SellingPointFactory.js` and replace the content with :

```
angular.module('bookservice').factory('SellingPointResource', function($resource){
    var resource =
$resource('http://localhost:8082/rest/sellingpoints/inrange/:Isbn',{Isbn:'@isbn'},{'qu
eryAll':{method:'GET',isArray:true},'query':{method:'GET',isArray:false},'update':{met
hod:'PUT'}});
    return resource;
});
```

In `src/main.webapp/scripts/controllers/searchSellingPointController.js` replace with :

```
angular.module('bookservice').controller('SearchSellingPointController',
function($scope, $window, $http, $filter, SellingPointResource ) {

    navigator.geolocation.getCurrentPosition(function(position){
        $scope.latitude = position.coords.latitude;
        $scope.longitude = position.coords.longitude;
        $scope.$apply();

    }, function(error){});

    $scope.search={};
    $scope.currentPage = 0;
    $scope.pageSize= 10;
    $scope.searchResults = [];
    $scope.filteredResults = [];
    $scope.pageRange = [];
    $scope.numberOfPages = function() {
        var result = Math.ceil($scope.filteredResults.length/$scope.pageSize);
        var max = (result == 0) ? 1 : result;
        $scope.pageRange = [];
        for(var ctr=0;ctr<max;ctr++) {
            $scope.pageRange.push(ctr);
        }
        return max;
    };
```

```
    $scope.performSearch = function() {
        $scope.searchResults = SellingPointResource.queryAll({Isbn:$scope.isbn,
longitude:$scope.longitude,latitude:$scope.latitude},function(){
            $scope.filteredResults = $filter('searchFilter')($scope.searchResults,
$scope);
            $scope.currentPage = 0;
        });
    };

    $scope.previous = function() {
       if($scope.currentPage > 0) {
           $scope.currentPage--;
       }
    };

    $scope.next = function() {
       if($scope.currentPage < ($scope.numberOfPages() - 1) ) {
           $scope.currentPage++;
       }
    };

    $scope.setPage = function(n) {
       $scope.currentPage = n;
    };

    //$scope.performSearch();
});
```

And finally, in the view `src/main/webapp/views/SellingPoint/search.html` :

```
<div>
    <h3>Search books around you</h3>
    <form id="SellingPointSearch" class="form-horizontal">


    <div class="form-group">
        <label for="name" class="col-sm-2 control-label">ISBN</label>
        <div class="col-sm-10">
            <input id="name" name="name" class="form-control" type="text" ng-
model="isbn" placeholder="Enter the SellingPoint Name"></input>
        </div>
    </div>

    <div class="form-group">
        <label for="latitude" class="col-sm-2 control-label">Latitude</label>
        <div class="col-sm-10">
            <input id="latitude" name="latitude" class="form-control" type="text" ng-
model="latitude" placeholder="Enter the SellingPoint Latitude"></input>
        </div>
```

```
    </div>

    <div class="form-group">
        <label for="longitude" class="col-sm-2 control-label">Longitude</label>
        <div class="col-sm-10">
            <input id="longitude" name="longitude" class="form-control" type="text"
ng-model="longitude" placeholder="Enter the SellingPoint Longitude"></input>
        </div>
    </div>


        <div class="form-group">
            <div class="col-md-offset-2 col-sm-10">
                <a id="Search" name="Search" class="btn btn-primary" ng-
click="performSearch()"><span class="glyphicon glyphicon-search"></span> Search</a>
            </div>
        </div>
    </form>
</div>
<div id="search-results">
        <div class="table-responsive">
        <table class="table table-responsive table-bordered table-striped clearfix">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Latitude</th>
                    <th>Longitude</th>
                </tr>
            </thead>
            <tbody id="search-results-body">
                <tr ng-repeat="result in filteredResults |
startFrom:currentPage*pageSize | limitTo:pageSize">
                    <td><a
href="#/SellingPoints/edit/{{result.id}}">{{result.name}}</a></td>
                    <td><a
href="#/SellingPoints/edit/{{result.id}}">{{result.latitude}}</a></td>
                    <td><a
href="#/SellingPoints/edit/{{result.id}}">{{result.longitude}}</a></td>
                </tr>
            </tbody>
        </table>
    </div>

        <ul class="pagination pagination-centered">
        <li ng-class="{disabled:currentPage == 0}">
            <a id="prev" href ng-click="previous()">«</a>
        </li>
        <li ng-repeat="n in pageRange" ng-class="{active:currentPage == n}" ng-
click="setPage(n)">
            <a href ng-bind="n + 1">1</a>
        </li>
```

```
        <li ng-class="{disabled: currentPage == (numberOfPages() - 1)}">
            <a id="next" href ng-click="next()">»»</a>
        </li>
    </ul>

</div>
```

Rebuild your app and deploy it, you should now be able to perform Spatial Queries

4. Secure All the Things !!

4.1. Keycloak Introduction

4.2. Deploy Keycloak Server

- Unzip the Keycloak distribution
- Add the admin user by running `$KEYCLOAK_SERVER/bin/add-user-keycloak.sh` (optional)

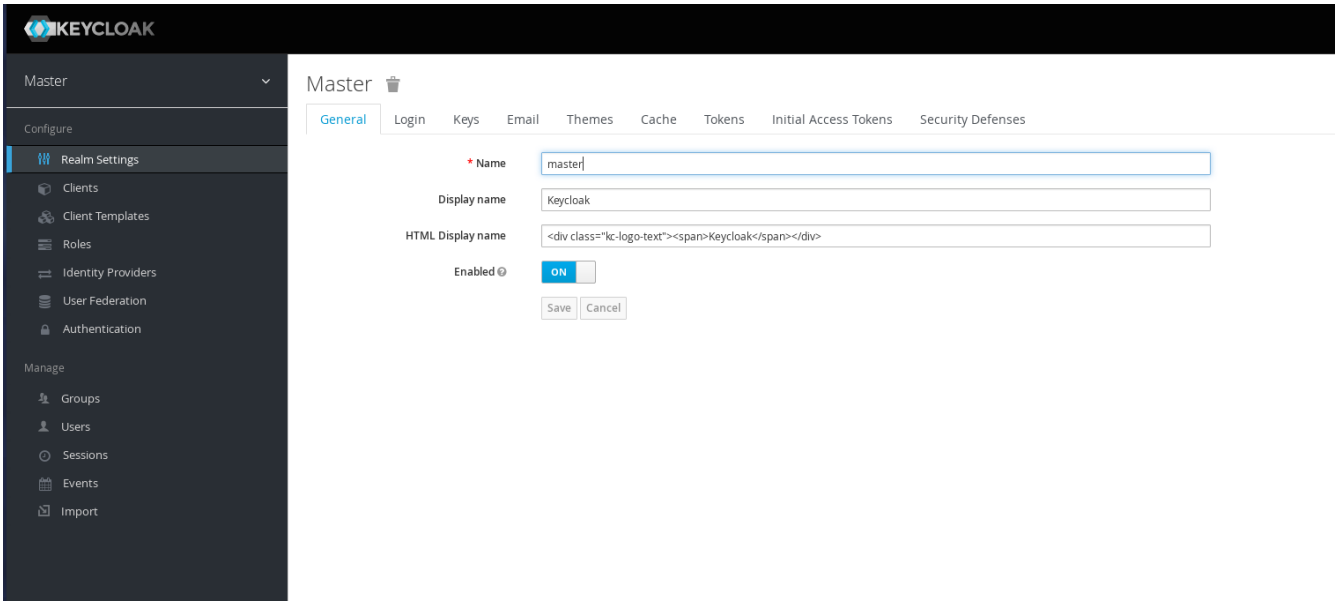
This is all you have to do to run a KeyCloak Server

4.2.1. Deploy the app

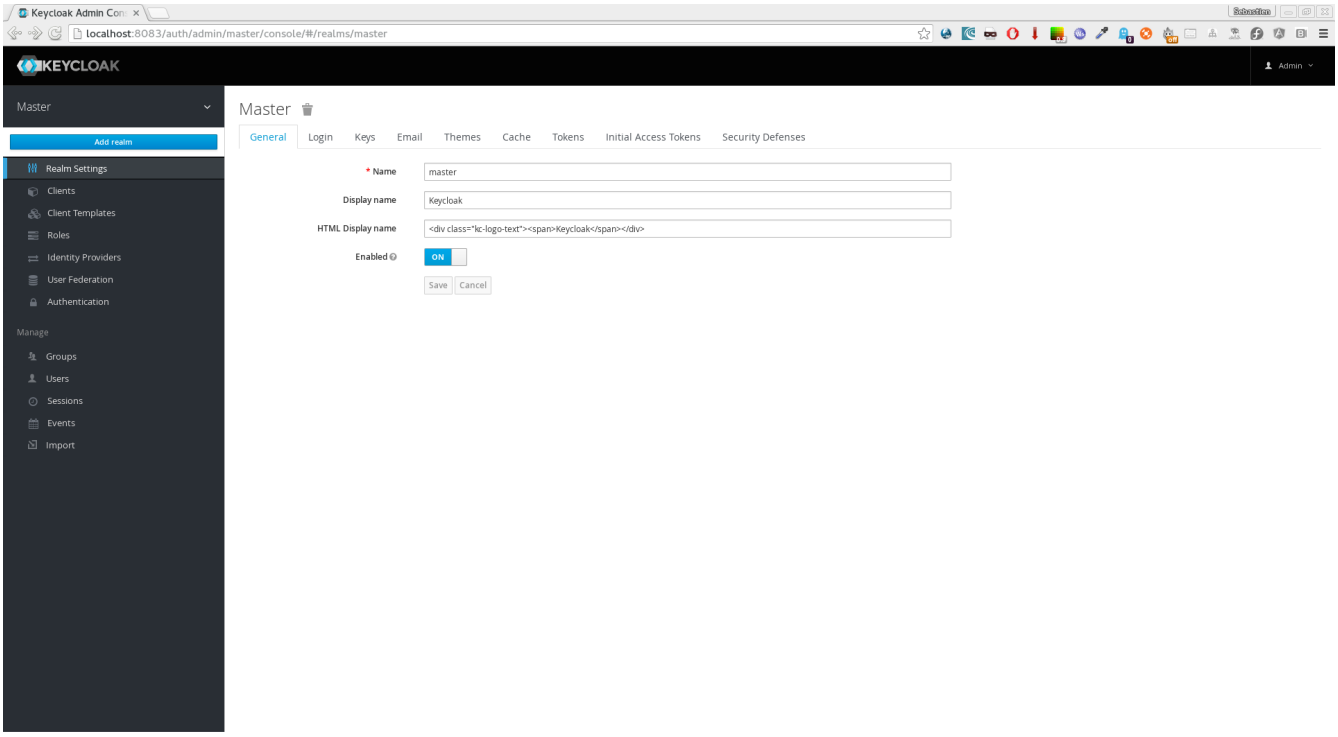
- `./standalone.sh -Djboss.socket.binding.port-offset=100`

4.2.2. Create a new Realm

- Open the KeyCloak management console at http://localhost:8180/auth
- Login and browse to the master realm



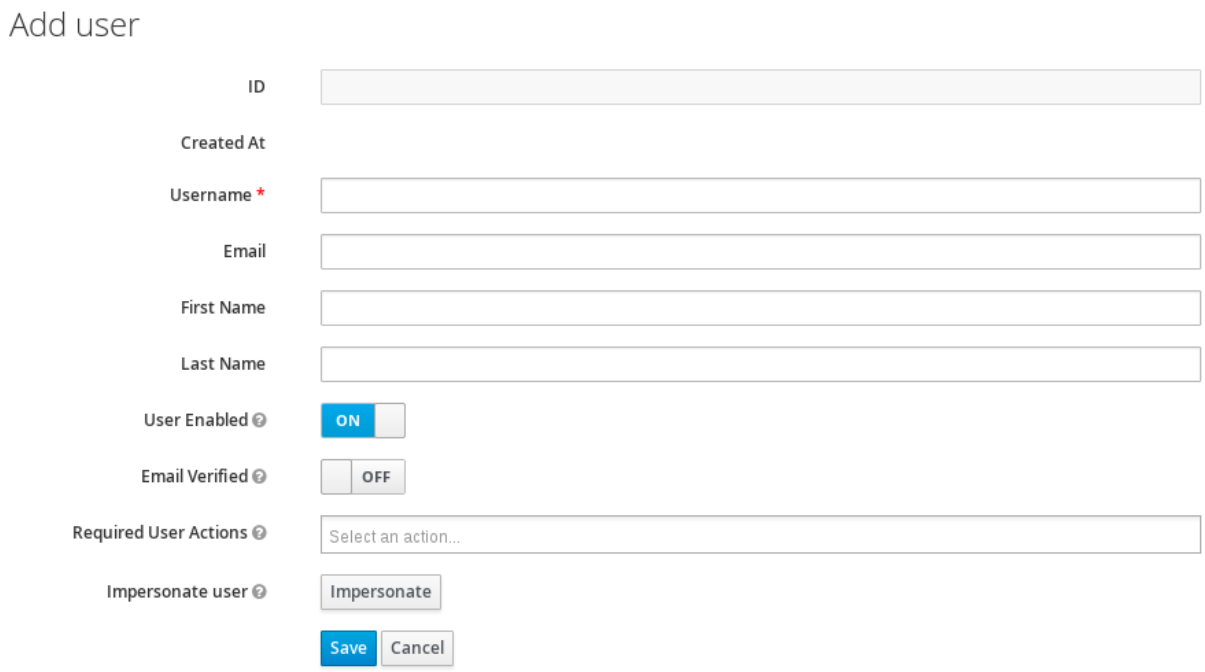
- Create a new realm `javaonehol`



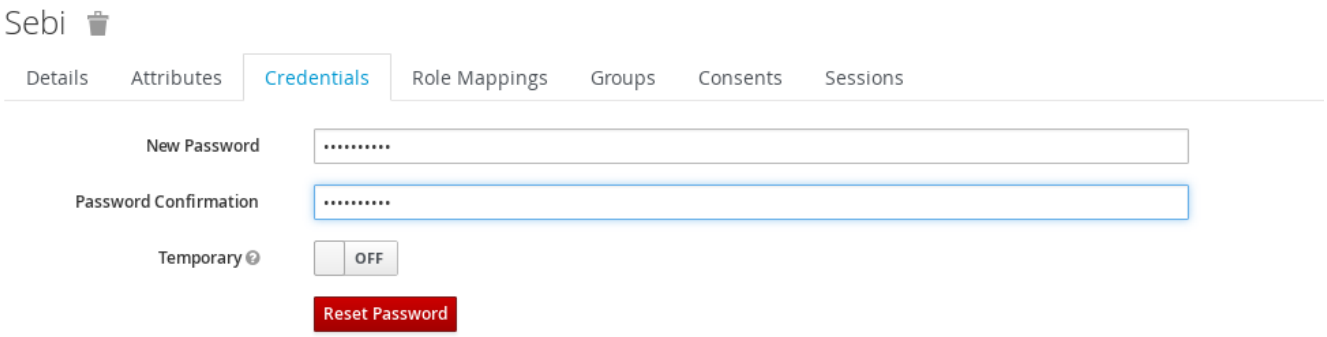
4.2.3. Configure Keycloak Server

Create a `user` role

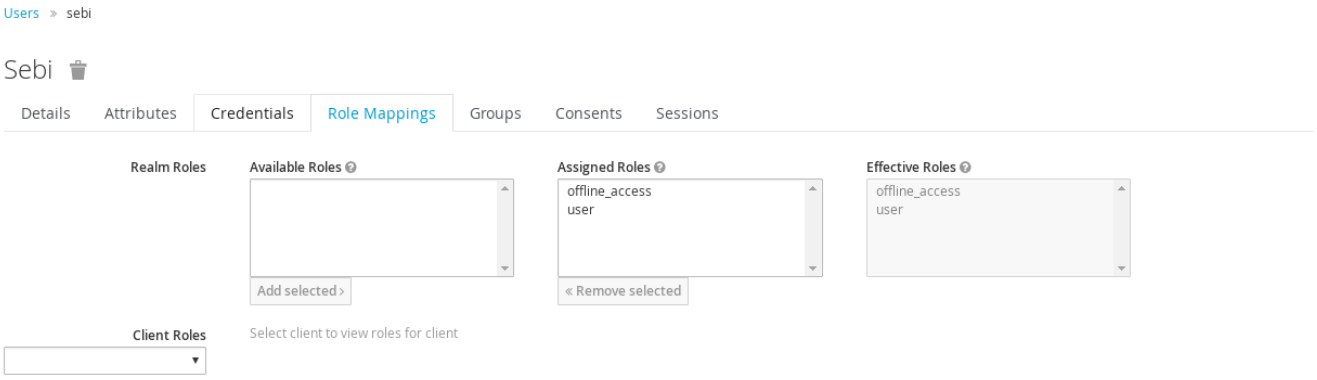
Create a new user



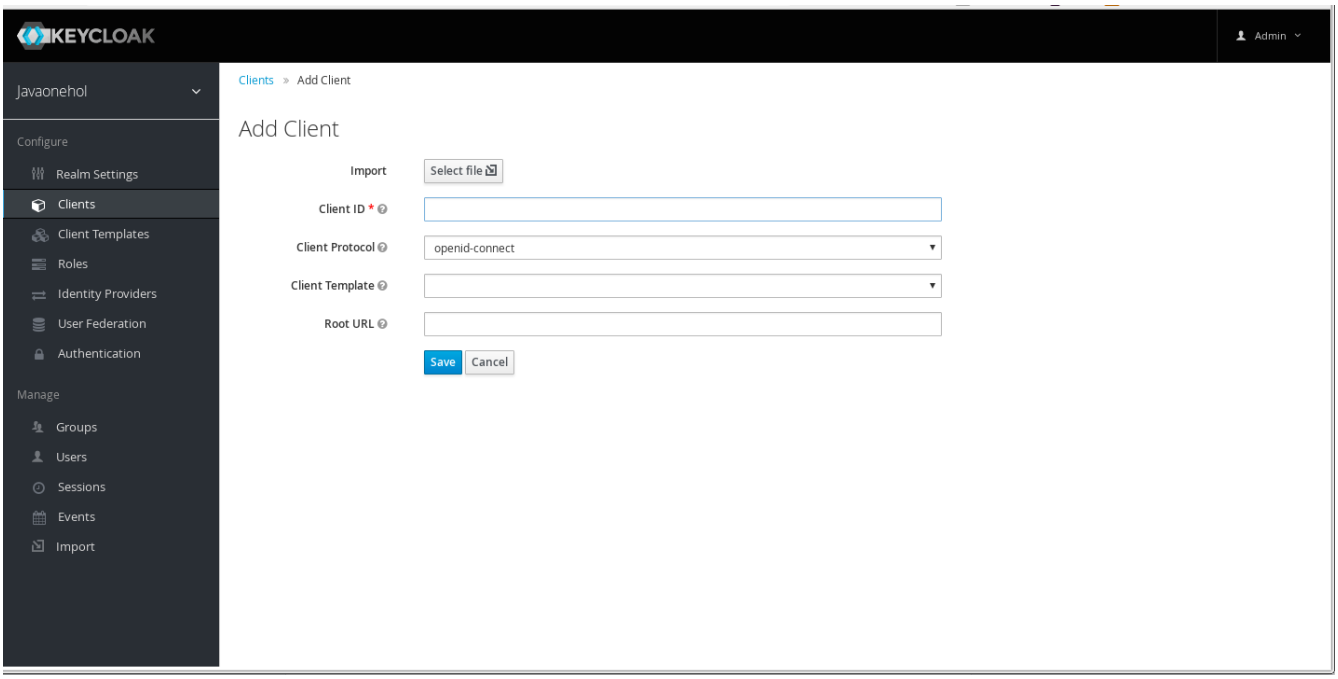
Reset credentials



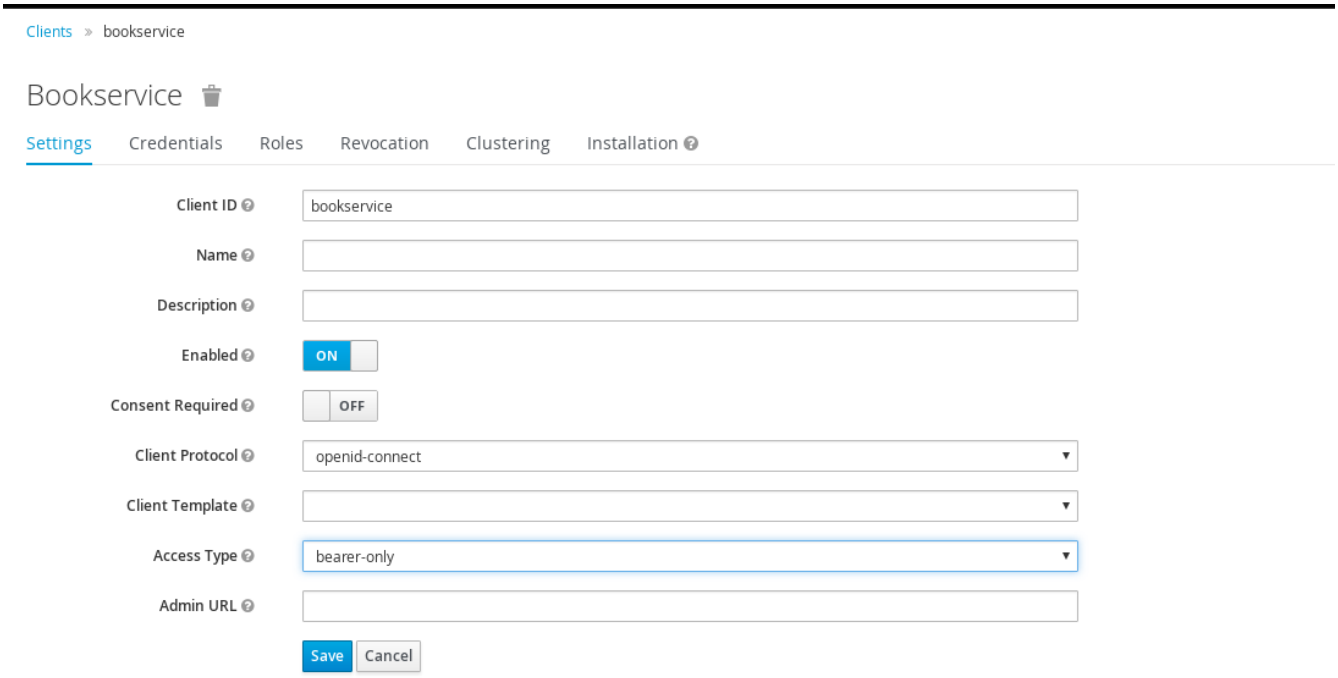
4.2.4. Set Role Mapping



4.2.5. Create `bookservice` client



On the next screen set the client to `bearer-only`:



Download `keycloak.json`:

- In the installation tab , select `Keycloak OIDC JSON` format and download the file
- Copy it to `bookservice/src/main/webapp/WEB-INF`

4.2.6. Create `bookstore` client

- Be sure to set the redirect URL
- Mind setting the **Web Origins** field to *

Download `keycloak.json`:

- In the installation tab , select `Keycloak OIDC JSON` format and download the file
- Copy it to `bookservice/src/main/webapp`

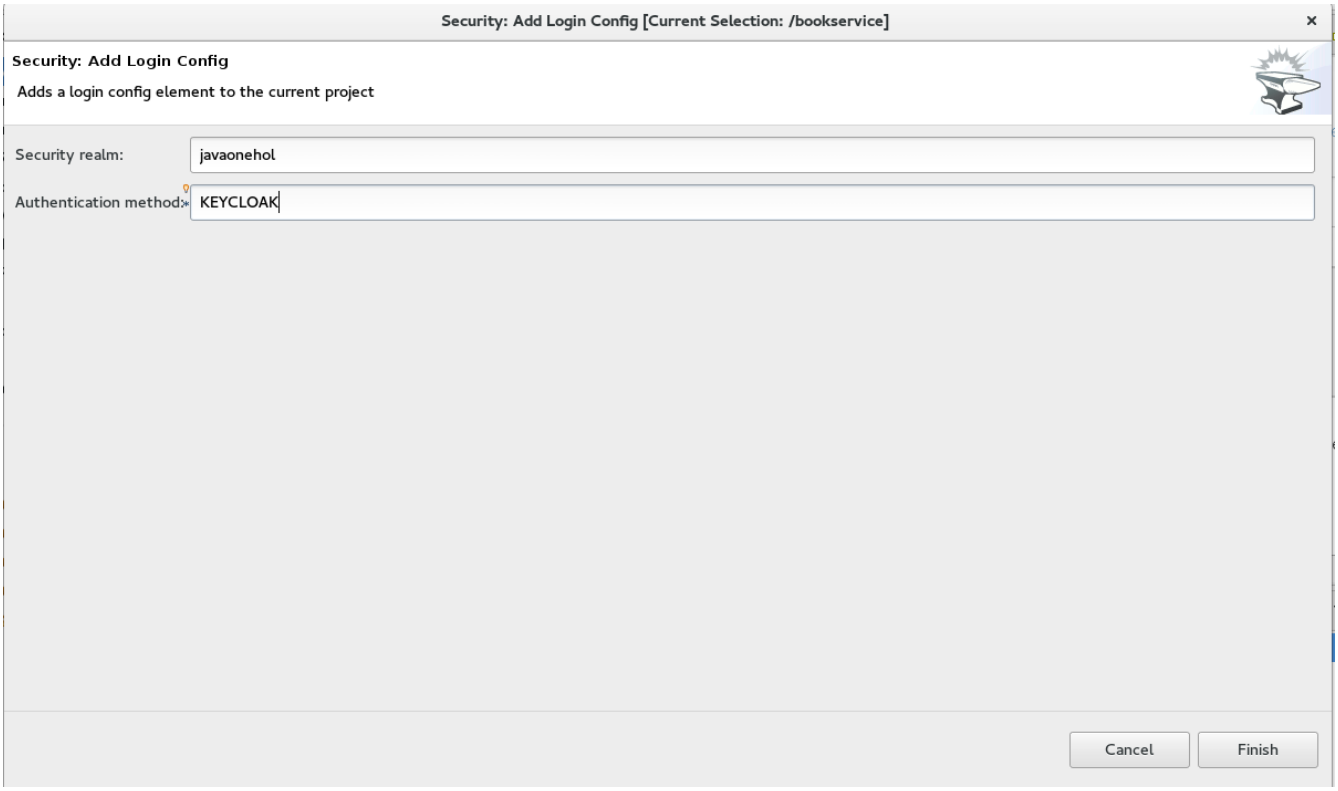
4.2.7. Create `sellingpoint` client

Same instructions as for the `bookservice` client.

4.3. Secure BookService App

Go back to Eclipse and open your Forge UI:

- Add a new swarm fraction: `keycloak`
- Search for "Security Add Login Config":
  - Realm: `javaonehol`
  - Authentification method: `KEYCLOAK`



- Search for "Security Add constraint" and add the following constraint:

```
security-add-constraint --web-resource-name Book --url-patterns /rest/* --security
-roles user
```

- The `web.xml` security part should look like that:

```
<login-config>
  <auth-method>KEYCLOAK</auth-method>
  <realm-name>javaonehol</realm-name>
</login-config>
<security-role>
  <role-name>user</role-name>
</security-role>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Book</web-resource-name>
    <url-pattern>/rest/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>user</role-name>
  </auth-constraint>
</security-constraint>
```

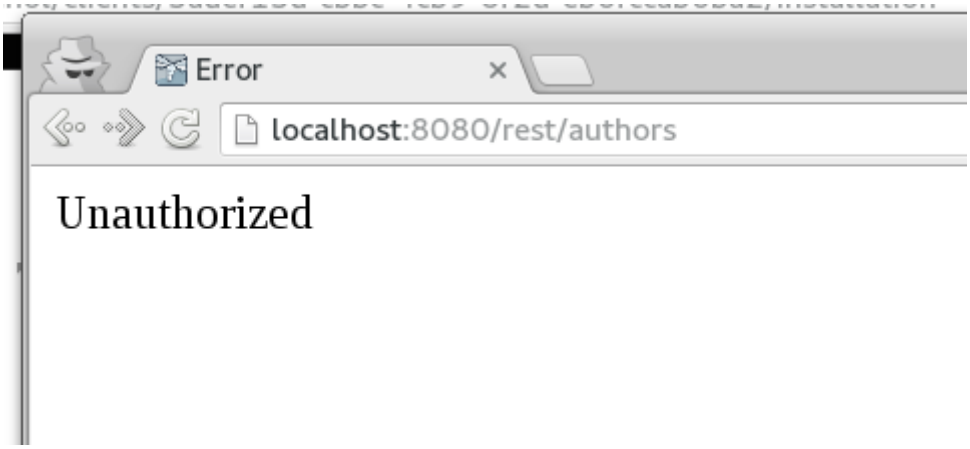
- KeyCloak will handle CORS for us, so make sure to remove the CORS filter: `rm src/main/java/org/bookservice/rest/NewCrossOriginResourceSharingFilter.java`
- Open `src/main/webapp/WEB-INF/keycloak.json` and add this key:

```
enable-cors: true
```

- Since we *manually* added the `keycloak` fraction, the autodetect feature won't work anymore. We need to change the `fractionDetectMode` in the Maven plugin. Inside the plugin definition add a `configuration` section:

```
<configuration>
    <fractionDetectMode>force</fractionDetectMode>
</configuration>
```

Redeploy the application and try to access any endpoints…



4.4. Secure BookStore front-end App

- In the `src/main/webapp/app.html`, at the section where all the JS scripts are loaded, add this one as first:

```
<script src="http://localhost:8180/auth/js/keycloak.js"></script>
```

- In the `<html>` tag remove the `ng-app` attribute
- In `src/main/webapp/scripts/app.js` append:

```
var keycloak = new Keycloak('keycloak.json');

angular.element(document).ready(function() {
  keycloak.init({ onLoad: 'login-required' }).success(function () {
    angular.bootstrap(document, ["bookservice"]);
  }).error(function () {
    console.log("ERROR");
  });
});

angular.module('bookservice').factory('authInterceptor', function($q) {
  return {
    request: function (config) {
      var deferred = $q.defer();
      if (keycloak.token) {
        keycloak.updateToken(5).success(function() {
          config.headers = config.headers || {};
          config.headers.Authorization = 'Bearer ' + keycloak.token;
          deferred.resolve(config);
        }).error(function() {
          deferred.reject('Failed to refresh token');
        });
      }
      return deferred.promise;
    }
  };
});

angular.module('bookservice').config(function($httpProvider) {
  $httpProvider.defaults.useXDomain = true;
  $httpProvider.interceptors.push('authInterceptor');
});
```

Redeploy the app, you should now be redirected to a login screen.

4.5. Secure SellingPoint