Spring Security Dynamic Authentication & Authorization Project

Mustafa Can AYDIN

August 2, 2021

Contents

.

1 Introduction

In this project a dynamic authorization security system where all the information regarding clients access rights to the desired URLs such as his/her roles and his/her permissions are retrieved from the database when needed and modified dynamically so as to provide mechanism to modify access rights without changing the back-end code. This project gives flexibility for applications where roles and permissions of the users changes regularly. Dynamic authorization mechanism provides the flexibility of of changing authorities of its clients without deploying the application to the server each time privilege change needed. This application uses Spring boot framework with dependencies such as spring web, spring security, java json web token, and java persistence api.

2 Structure

2.1 Controller

Project includes authentication and authorization of the user according to roles and permission he/she possesses. Applications' model consist of three main classes : User, Role, Permission. Each user has unique user name, and email; a password and roles and permissions. Each role has permissions and names and each permission has names indicating which url a permission has access to. Application implements a internal filter where each time a client makes a http request it identifies the client by the json web token attached to http header's authorization field. If there is token provided client has only access right only the urls that are declared public. This json web token attached to http upon the login of the user and permits user to access his/her desired urls accoriding to his/her privileges. After one day (arbitrarily chosen) token expires and user needs to login again to have access rights.

Project includes classes enabling client to make http requests including controller classes and classes contained in package named payload, classes determining database structure located in model and repository packages and classes permorming the functionalities of the authorization and authentication.

Controller classes permits client to make http request to the server. Json objects's fields are determined in the payload.request package, and the responses for the http requests of the client determined by the classes of the package payload.response.

2.2 Persistence Layer

After clients requests reaches to the server, out application needs to perform main functionalities of the applicaiton which are authenticaiton and authorization. However, before performing those persistence layer of the

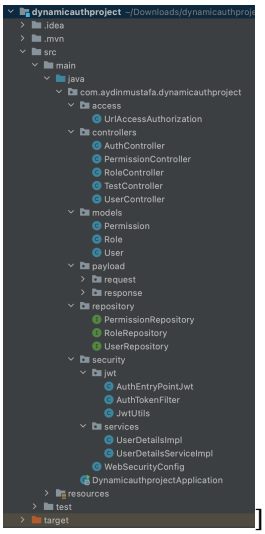


Figure 1: File structure of the project.

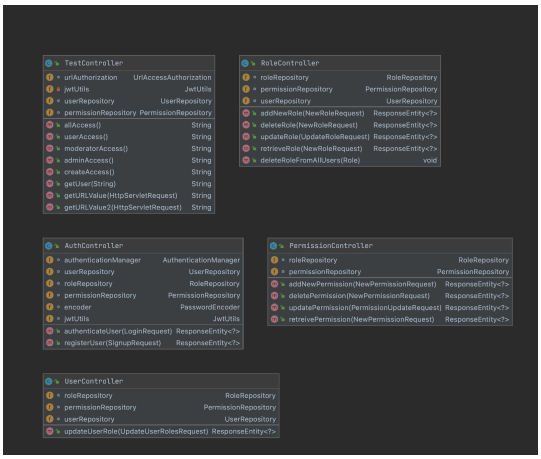
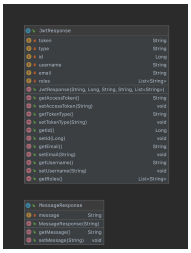


Figure 2: Detailed view of controller classes.

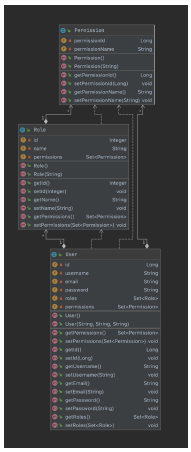
(a) Request package



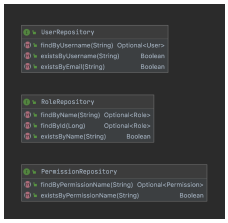
(b) response package

Figure 3: Figures showing payload package

application needs to be examined. So briefly packages repository and models need mentioning.



(a) model package



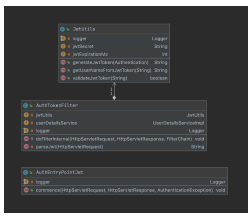
(b) repository package

Figure 4: Figures showing models and repositories

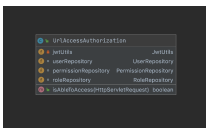
Application includes three main classes that will form the basis of our database: User.java, Role.java, and Permission.java. These classes determines relationship between entities of users, roles, and permissions. According to this model each user stores set of roles and permissions and each role stores set of permissions in addition to their required fields such as id and name. Even though there exist many to many relationship between the users and roles, roles doesnt stores list of users having the role since according to assumptions of this application changes to users and permissions will make up vast majority of the all the operation and changes to roles will be mainly changes to its permissions.

2.3 Bussiness Layer

Authorization and authentication functionalities implemented with the help of json web tokens. Json web tokens have their package named jwt and authorization filer called url access athorization have its own package called access.



(a) JWT package



(b) url access authorization class

Figure 5: Figures showing jwt and access packages

2.4 General View

And lastly general view of the project

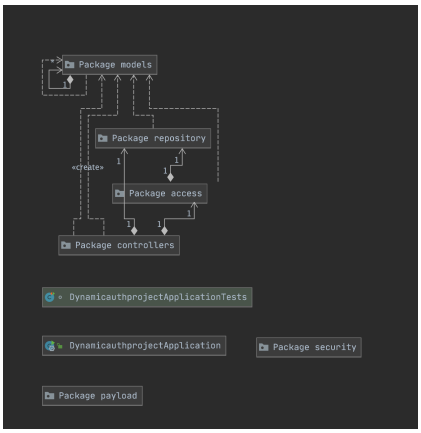


Figure 6: General view of the project.

3 Functionality

Project gives functionalities of communication with database and performing CRUD operations with the help of JPA. Privileged users can create/read-/update/delete roles and permissions from the database. Each time a user requests to access to an URL, application checks for users roles and permissions that the his/her roles provides. After all the permission names which stores the url addresses extracted from the database comparing them to requested url address will decide whether or not user has access rights to the url that is being requested. if one of the permissions of the user matches to the url being requested than user directed to the url otherwise application gives user access denied message.

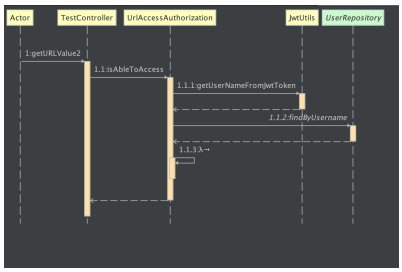


Figure 7: Sequence diagram of dynamic authorization mechanism.

4 Assumptions & Constraints

4.1 Assumptions

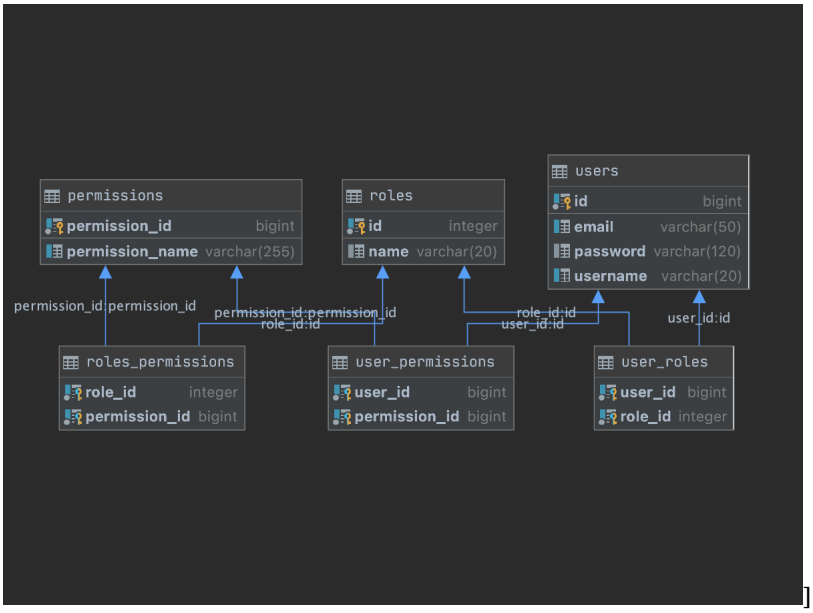
- Roles won't be deleted frequently so each role doesn't have to store set of users who have the role. This implies application saves space and increases readability due to reduction in complexity of relationships but at the same time makes deletion of a role costly operation with complexity of O(N*M) where N equals to number of users and M equals to average number of roles each user has.

4.2 Constraints

- Role names are chosen with prefix to ease readability but the application does not enforce this convention.
- In order for urls to be accessed using dynamic authorization each url has to be stored in the permissions table of the database. For instance, it one were to access a child of a url which he/she has permission to, it should explicitly be declared in the database. (`localhost:8080/url` does not enable accesses to `localhost:8080/url/xxx`)

5 Database Structure

5.1 ER Diagram



]

Figure 8: Image showing entity relations diagram.

6 CRUD Operations

6.1 CRUD operations for roles

These operations are:

- Create a role
- Retrieve a role
- Update role
- Delete a role

6.2 CRUD operations for permissions

These operations are:

- Create a permission
- Update a permission
- Retrieve a permission
- Delete a permission

7 R&D Discussion

- Instead of JPA, MyBatis could have been used in order for optimization to be made to SQL used. Since this project will store all the information in database it is reasonable to assume there would be much more emphasis on the database and SQL.
- Regular expressions could be added so as to allow each permission to comprared to the string of the requested url where a permission object includes statements such as *.
  Example : apit/test/** equals to set of api/test/xxx urls

8 Compilation and Running

Requirements : Java 1.8 & maven & PostgreSQL

8.1 Compilation

```
mvn install
```

8.2 Execution

```
java -jar target/dynamicAuthProject.jar
```

9 Examples

create new user account

POST request to `http://localhost:8080/api/auth/signup`

```
{
     "username":"username1",
     "password":"password",
     "role":["ROLE_MOD1"],
     "email":"email
}
```

login to user account

POST request to `http://localhost:8080/api/auth/signin`

```
{
     "username":"username1",
     "password":"password",
}
```

retrieve user information

POST request to `http://localhost:8080/api/user/retrieveUser`

```
{
     "username":"username1",
}
```

update user role

POST request to `http://localhost:8080/api/user/updateUserRole`

```
  {
     "username":"mustafa_test1",
     "role": ["ROLE_MOD1"]
}
```

creare a new role

POST request to `http://localhost:8080/api/role/addRole`

```
{
     "role_name":"ROLE_MOD10",
     "permissions":["/api/test/url223"]
}
```

delete a role

POST request to `http://localhost:8080/api/role/deleteRole`

```
{
     "role_name":"ROLE_MOD2",


}
```

retrieve a role

POST request to `http://localhost:8080/api/role/retrieveRole`

```
{
     "role_name":"ROLE_MOD2",
}
```

update the role

POST request to `http://localhost:8080/api/role/updateRole`

```
{
     "role_name":"ROLE_MOD1",
     "new_role_name":"ROLE_MOD1",
     "new_permission_set":["/api/test/url2"]
}
```

add new permissions

POST request to `http://localhost:8080/api/permission/addPermission`

```
  {
     "permission_name":"/api/test/url22"
}
```

delete the permission

POST request to `http://localhost:8080/api/permission/deletePermission`

```
  {
     "permission_name":"/api/test/url22"
}
```

update the permission

POST request to `http://localhost:8080/api/permission/updatePermission`

```
      {
     "permission_name":"/api/test/url22",
     "new_permission_name":"/api/test/url223"
}
```

retrieve the permission

POST request to `http://localhost:8080/api/permission/retrievePermission`

```
  {
     "permission_name":"/api/test/url22"
}
```

10 Conclusion & Assessment

To sum it up, this is a system that performs authention and authorization dynamically. It is capable of authorizing user acording to provileges that they have without hard coding those privileges to the back-end code and instead retrienig neccessary information from the database.