Minor Project 2

Building a company schema using Oauth 2.0 for security

Piyush Aggarwal

Agenda

What will be doing in this project ?

- Configure Spring Security + database.
- Create an Authorization Server.
- Create a Resource Server.
- Get an access token and a refresh token from Authorization Server.
- Get a secured Resource from resource server using an access token.

Note : (As a grant type, we will use a password and use BCrypt Password Encoder to hash our passwords).

Some conceptual things that you need to know

Ques. 1 Why Use OAuth 2.0 ?

The OAuth 2.0 specification defines a delegation protocol that is useful for conveying authorization decisions across a network of web-enabled applications and APIs. OAuth is used in a wide variety of applications, including providing mechanisms for user authentication**.**

Ques. 2 What are some terms you should need to understand/familiarize with ?

Resource owner (the User) – an entity capable of granting access to a protected resource (for example end-user). (In this example our postman, will be our user)

Resource server (the API server) – the server hosting the protected resources, capable of accepting responding to protected resource requests using access tokens.(In this case localhost is our resource server)

OAuth Client – an application making protected resource requests on behalf of the resource owner and with its authorization. (In this case our spring boot app server is our OAuth Client)

Authorization server – the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

Some conceptual things that you need to know (contd.)

Ques. 3 What are the different types of grant types or you can say diff mechanisms for authorization ?

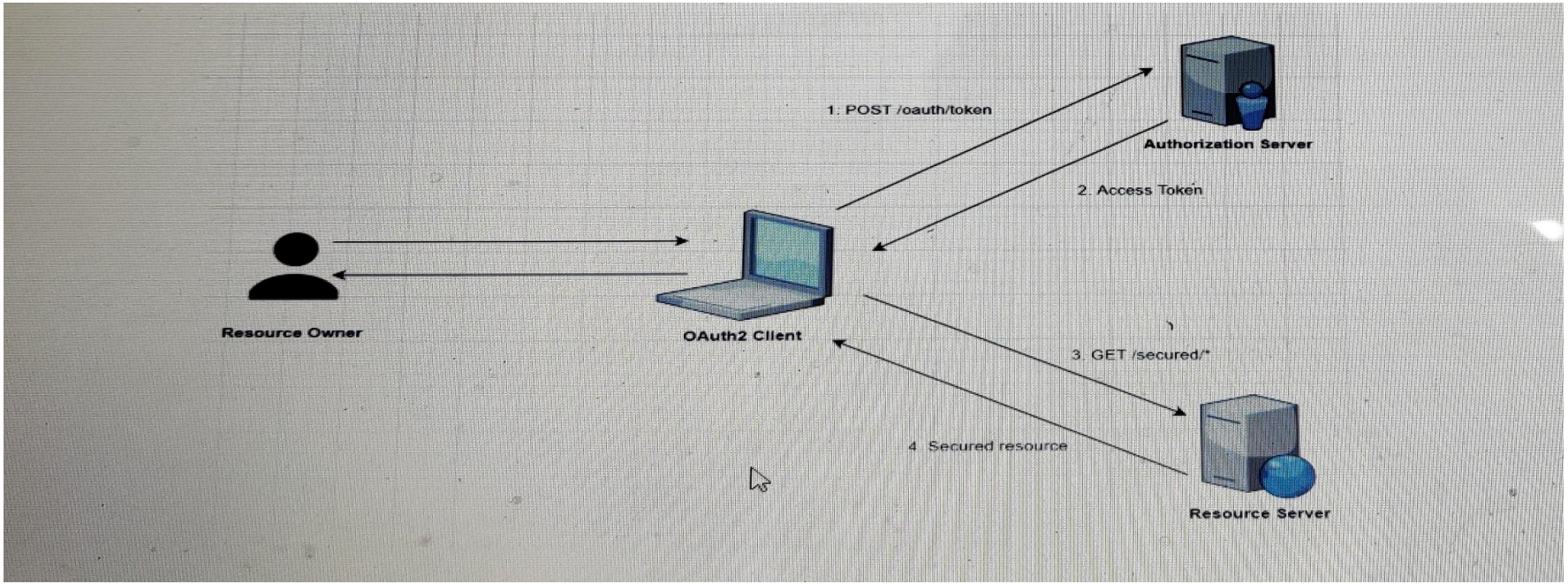
Grant Types

OAuth 2 provides several "grant types" for different use cases. The grant types defined are:

- Authorization Code
- Password
- Client credentials
- Implicit

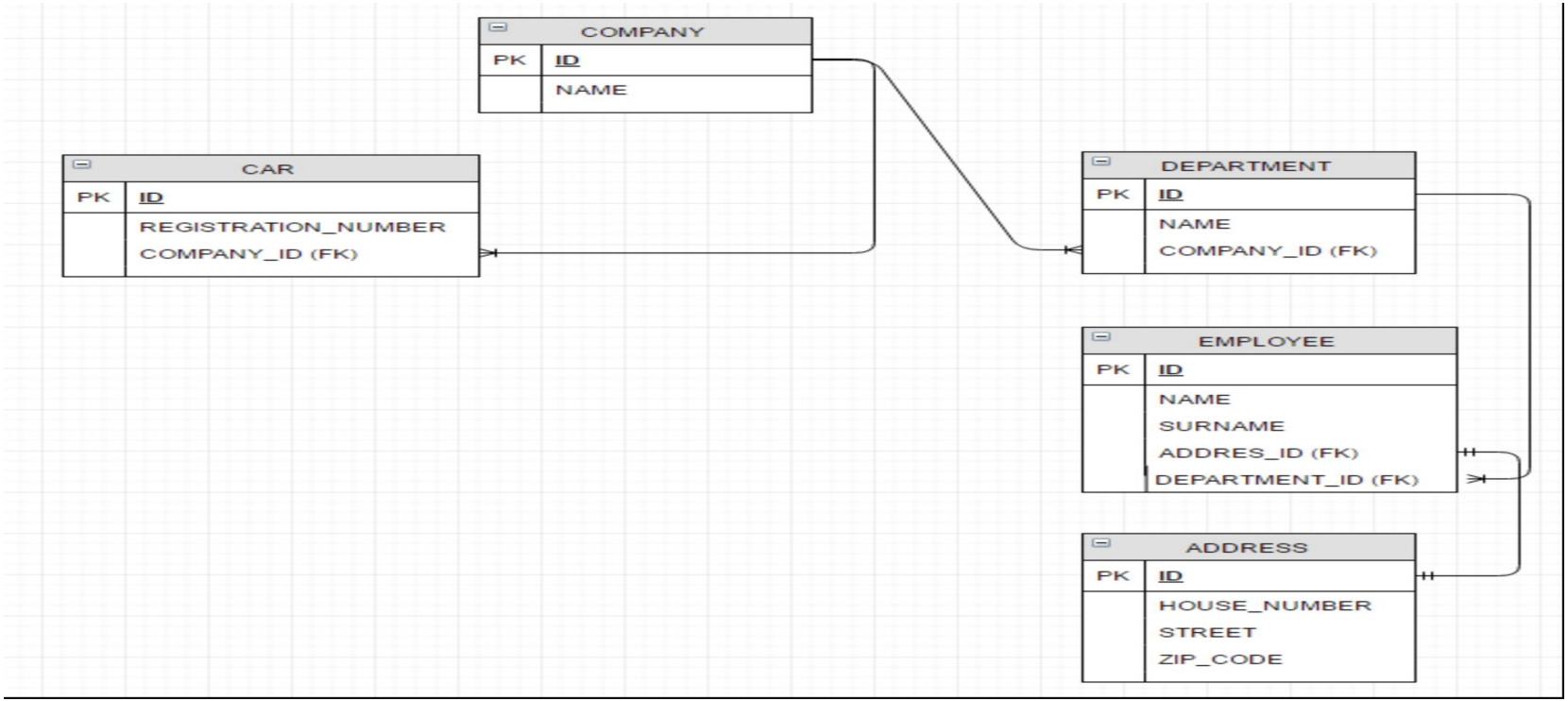
For more information on grant types refer to this article -> https://oauth.net/2/grant-types/

Overall flow of a Password Grant:



Piyush Aggarwal

Business Schema



Note : There is an office table also, missed that in picture, these are the 6 tables related to business logic

Piyush Aggarwal

Authorities a user can have

Based on CRUD operations for Company and Department objects ,we want to define following access rules:

COMPANY_CREATE

COMPANY_READ

COMPANY_UPDATE

COMPANY_DELETE

DEPARTMENT_CREATE

DEPARTMENT_READ

DEPARTMENT_UPDATE

DEPARTMENT_DELETE

Piyush Aggarwal

Oauth 2 Client Set up

We need to create the following tables in the database (for internal purposes of OAuth2 implementation and names should be exactly similar to these) :

OAUTH_CLIENT_DETAILS ->
https://github.com/spring-projects/spring-security-oauth/blob/2.2.1.RELEASE/spring-security-oauth2/src/main/java/org/springframework/security/oauth2/provider/client/JdbcClientDetailsService.java

OAUTH_CLIENT_TOKEN ->
https://github.com/spring-projects/spring-security-oauth/blob/2.2.1.RELEASE/spring-security-oauth2/src/main/java/org/springframework/security/oauth2/client/token/JdbcClientTokenServices.java

OAUTH_ACCESS_TOKEN ->
https://github.com/spring-projects/spring-security-oauth/blob/2.2.1.RELEASE/spring-security-oauth2/src/main/java/org/springframework/security/oauth2/provider/token/store/JdbcTokenStore.java

OAUTH_REFRESH_TOKEN ->
https://github.com/spring-projects/spring-security-oauth/blob/2.2.1.RELEASE/spring-security-oauth2/src/main/java/org/springframework/security/oauth2/provider/token/store/JdbcTokenStore.java

OAUTH_CODE ->
https://github.com/spring-projects/spring-security-oauth/blob/2.2.1.RELEASE/spring-security-oauth2/src/main/java/org/springframework/security/oauth2/provider/code/JdbcAuthorizationCodeServices.java

OAUTH_APPROVALS ->
https://github.com/spring-projects/spring-security-oauth/blob/2.2.1.RELEASE/spring-security-oauth2/src/main/java/org/springframework/security/oauth2/provider/approval/JdbcApprovalStore.java

Loading data in these Oauth2.0 related tables

Let's name a resource ‘resource-server-rest-api’ which will be accessible on resource server. For this server, we define two clients called:

spring-security-oauth2-read-client (authorized grant types: read)

spring-security-oauth2-read-write-client (authorized grant types: read, write)

INSERT INTO OAUTH_CLIENT_DETAILS(CLIENT_ID, RESOURCE_IDS, CLIENT_SECRET, SCOPE, AUTHORIZED_GRANT_TYPES, AUTHORITIES, ACCESS_TOKEN_VALIDITY, REFRESH_TOKEN_VALIDITY)

VALUES ('spring-security-oauth2-read-client', 'resource-server-rest-api',

/*spring-security-oauth2-read-client-password1234*/'$2a$04$WGq2P9egiOYoOFemBRfsiO9qTcyJtNRnPKNBl5tokP7lP.eZn93km',

'read', 'password,authorization_code,refresh_token,implicit', 'USER', 10800, 2592000);

INSERT INTO OAUTH_CLIENT_DETAILS(CLIENT_ID, RESOURCE_IDS, CLIENT_SECRET, SCOPE, AUTHORIZED_GRANT_TYPES, AUTHORITIES, ACCESS_TOKEN_VALIDITY, REFRESH_TOKEN_VALIDITY)

VALUES ('spring-security-oauth2-read-write-client', 'resource-server-rest-api',

/*spring-security-oauth2-read-write-client-password1234*/'$2a$04$soeOR.QFmClXeFlrhJVLWOQxfHjsJLSpWrU1iGxcMGdu.a5hvfY4W',

'read,write', 'password,authorization_code,refresh_token,implicit', 'USER', 10800, 2592000);

Note that password is hashed with BCrypt (4 rounds).

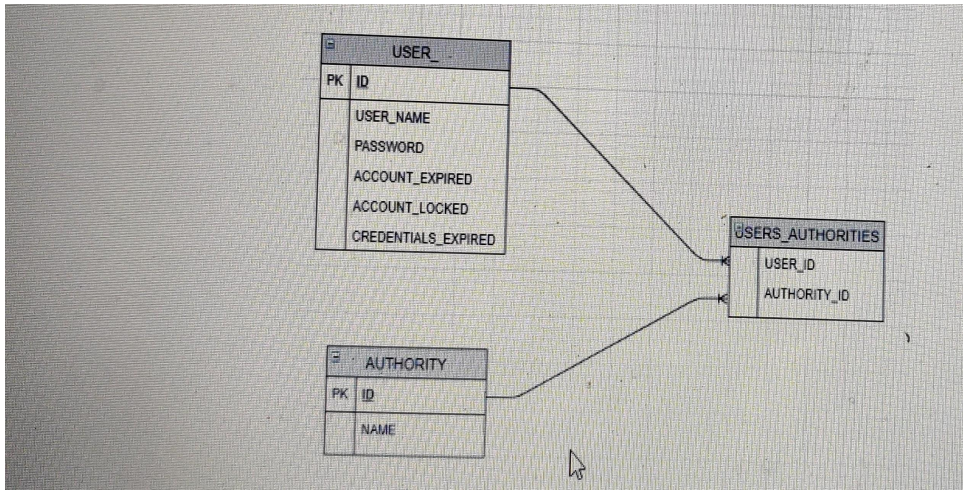
Authorities and Users Set Up

Now we will add users in the DB along with authorities

Spring Security comes with two useful interfaces:

- **UserDetails - provides core user information.**
- **GrantedAuthority - represents an authority granted to an Authentication object.**

To store authorization data we will define following data model:



Piyush Aggarwal

Inserting diff authorities in DB using sql script

```
INSERT INTO AUTHORITY(ID, NAME) VALUES (1, 'COMPANY_CREATE');
INSERT INTO AUTHORITY(ID, NAME) VALUES (2, 'COMPANY_READ');
INSERT INTO AUTHORITY(ID, NAME) VALUES (3, 'COMPANY_UPDATE');
INSERT INTO AUTHORITY(ID, NAME) VALUES (4, 'COMPANY_DELETE');
INSERT INTO AUTHORITY(ID, NAME) VALUES (5, 'DEPARTMENT_CREATE');
INSERT INTO AUTHORITY(ID, NAME) VALUES (6, 'DEPARTMENT_READ');
INSERT INTO AUTHORITY(ID, NAME) VALUES (7, 'DEPARTMENT_UPDATE');
INSERT INTO AUTHORITY(ID, NAME) VALUES (8, 'DEPARTMENT_DELETE');
```

Piyush Aggarwal

Inserting users in DB

```
INSERT INTO USER_(ID, USER_NAME, PASSWORD, ACCOUNT_EXPIRED, ACCOUNT_LOCKED, CREDENTIALS_EXPIRED, ENABLED)
  VALUES (1, 'admin', /*admin1234*/'$2a$08$qvrzQZ7jJ7oy2p/msL4M0.l83Cd0jNsX6AJUitbgRXGzge4j035ha', FALSE, FALSE, FALSE, TRUE);

INSERT INTO USER_(ID, USER_NAME, PASSWORD, ACCOUNT_EXPIRED, ACCOUNT_LOCKED, CREDENTIALS_EXPIRED, ENABLED)
  VALUES (2, 'reader', /*reader1234*/'$2a$08$dwYz8O.qtUXboGosJFsS4u19LHKW7aCQ0LXXuNlRfjjGKwj5NfKSe', FALSE, FALSE, FALSE, TRUE);

INSERT INTO USER_(ID, USER_NAME, PASSWORD, ACCOUNT_EXPIRED, ACCOUNT_LOCKED, CREDENTIALS_EXPIRED, ENABLED)
  VALUES (3, 'modifier', /*modifier1234*/'$2a$08$kPjzxewXRGNRiluL4FtQH.mhMn7ZAFBYKB3ROz.J24IX8vDAcThsG', FALSE, FALSE, FALSE, TRUE);

INSERT INTO USER_(ID, USER_NAME, PASSWORD, ACCOUNT_EXPIRED, ACCOUNT_LOCKED, CREDENTIALS_EXPIRED, ENABLED)
  VALUES (4, 'reader2', /*reader1234*/'$2a$08$vVXqh6S8TqfHMs1SlNTu/.J25iUCrpGBpyGExA.9yI.llDRadR6Ea', FALSE, FALSE, FALSE, TRUE);
```

Inserting user authorities in DB

```
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 1);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 2);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 3);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 4);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 5);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 6);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 7);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 8);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (1, 9);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (2, 2);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (2, 6);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (3, 3);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (3, 7);
INSERT INTO USERS_AUTHORITIES(USER_ID, AUTHORITY_ID) VALUES (4, 9);
```

Note that the password is hashed with BCrypt (8 rounds).

Password Encoders

Since we are going to use different encryptions for OAuth2 client and user, we will define separate password encoders for encryption:

OAuth2 client password – BCrypt (4 rounds)

User password - BCrypt (8 rounds)

```
@Configuration
public class Encoders {
    @Bean
    public PasswordEncoder oauthClientPasswordEncoder() {
        return new BCryptPasswordEncoder(4);
    }
    @Bean
    public PasswordEncoder userPasswordEncoder() {
        return new BCryptPasswordEncoder(8);
```

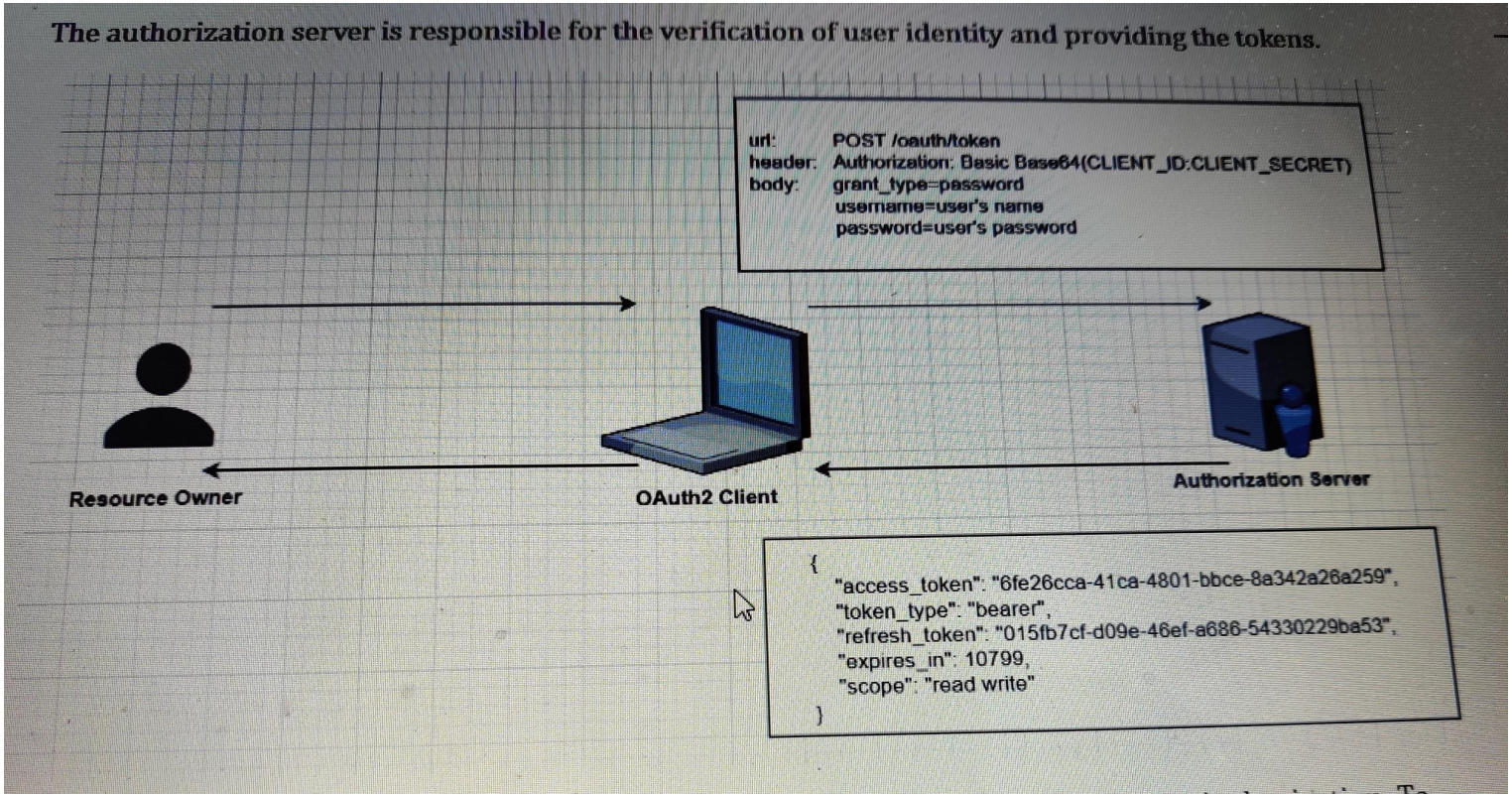
Configs for different entities (res. Server, auth server)

We have defined different configurations for different entities :

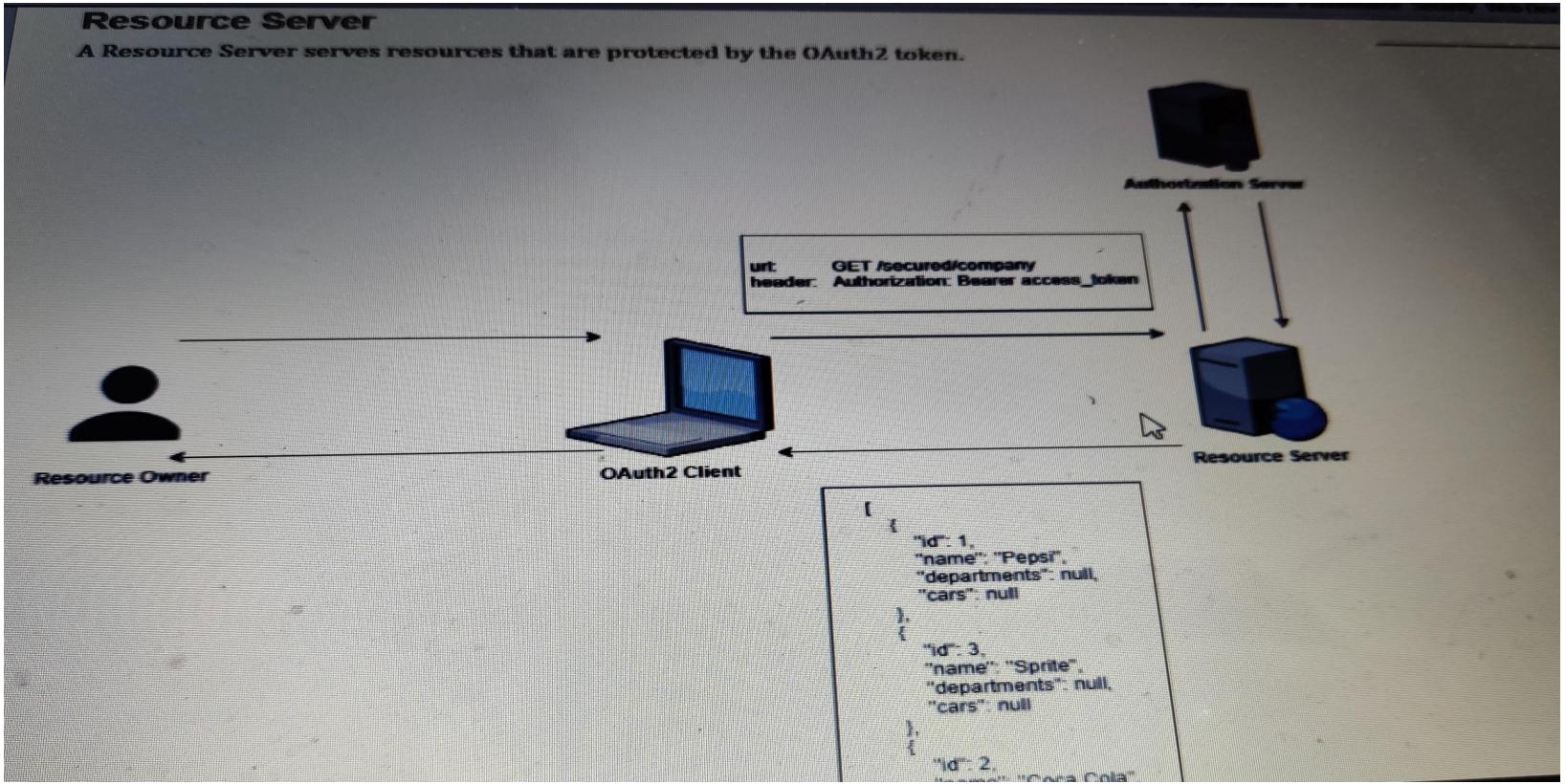
1. Resource server :
2. Authorization server :
3. Oauth Client

Refer to respective config files in code for more details

Flow of communication b/w Oauth2.0 client and authorization server



Flow of communication b/w Oauth2.0 client and Resource Server



Some important dependencies that we used so far in this project

1. Flyway -> This dependency is useful for migrating your data (written in sql scripts into the database when application starts up)

   For more info refer this -> https://www.baeldung.com/database-migrations-with-flyway

2. Lombok -> This dependency is useful for reducing code by using @getter, @setter, @constructor annotations and you don't need to write methods for that

   For more info refer this -> https://www.baeldung.com/intro-to-project-lombok

Some important dependencies that we used so far in this project (contd.)

3. JPAmodelGen -> Useful for generating meta model classes in the target folder

Note : (You need to do mvn clean package in order to generate)

For more info why we need this -> https://www.baeldung.com/hibernate-criteria-queries-metamodel

Postman Request (1st request for getting token)

```
curl --location --request POST 'localhost:8080/oauth/access_token' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--header 'Authorization: Basic
c3ByaW5nLXNlY3VyaXR5LW9hdXRoMi1yZWFkLXdyaXRlLWNsaWVudDpzcHJpbmctc2VjdXJpdHktb2F1dGggyLXJlYWQtd3JpdGUtY2
xpZW50LXBhc3N3b3JkMTIzNA==' \

--form 'client_id=spring-security-oauth2-read-write-client' \

--form 'password=admin1234' \

--form 'username=admin' \

--form 'grant_type=password'
```

Piyush Aggarwal

Postman request 2 (for getting secured resource)

This is for getting all the companies

```
curl --location --request GET 'localhost:8080/secured/company' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--header 'Authorization: Bearer df1b652c-1412-4274-903c-361bef96c85a' \

--form 'client_id=spring-security-oauth2-read-client' \

--form 'password=reader1234' \

--form 'username=reader' \

--form 'grant_type=password'
```

Note : pls change the bearer token :p

Piyush Aggarwal

Postman request 3 (additional request)

For getting company with id 1:

```
curl --location --request GET 'localhost:8080/secured/company/1' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--header 'Authorization: Bearer df1b652c-1412-4274-903c-361bef96c85a' \

--form 'client_id=spring-security-oauth2-read-client' \

--form 'password=reader1234' \

--form 'username=reader' \

--form 'grant_type=password'
```

Postman request 4 (additional)

This is for creating a company (post request)

```
curl --location --request POST 'localhost:8080/secured/company' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--header 'Authorization: Bearer df1b652c-1412-4274-903c-361bef96c85a' \

--header 'Content-Type: text/plain' \

--data-raw '{

"id" : 5,

"name" : "Soda",

}'
```

Piyush Aggarwal