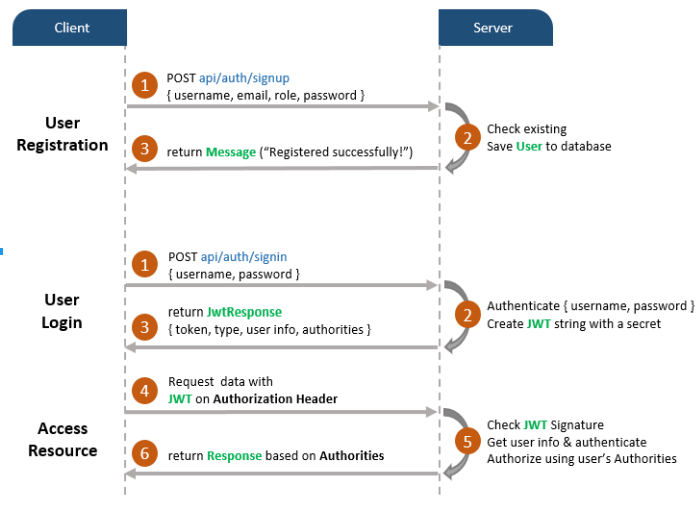
# **Spring boot security + Oauth2 authentication + JWT example:**

**1.1 OAuth2** is a token-based security authentication and authorization framework that breaks security down into four components.

*OAuth defines four components –*

* **Resource Owner** – Which users are allowed to access the service, and what they can do with the service. Each application registered by the resource owner will be given an application name that identifies the application along with an application secret key. The combination of the application name and the secret key are part of the credentials that are passed when authenticating an OAuth 2 token.
* **Client** – the application (user is using) which require access to user data on the resource server.
* **Resource Server** – store user’s data and http services which can return user data to authenticated clients.
* **Authorization Server** – responsible for authenticating user’s identity and gives an authorization token. This token is accepted by resource server and validate your identity.



Refer : <https://bezkoder.com/spring-boot-jwt-authentication/>

Web service security is an extremely complicated subject. You have to understand who’s going to call your services , how they’re going to call your service , and what actions they’re going to take with your code. OA uth 2 allows you to protect your REST -based services across these different scenarios through different authentication schemes called grants.

*OAuth 2 specification has four types of grants:*

* Password
* Client credential
* Authorization code
* Implicit

**1.2 Starting using Spring and Oauth2**

To understand how to set up the authentication and authorization pieces of Oauth2, you’re going to implement the OAuth 2 password grant type. To implement this grant, you’ll do the following:

***Step1:*** *User Registration on Spring Boot APIs*

**1.2.1** The first step is to allow new users to register themselves. The classes that we will create in this feature will belong to a new package called com.oauth2.example.authorizationserver.model Let's create this package and add a new entity class called User to it.

**1.2.2** To manage the persistence layer of this entity, we will create an interface called UserRepository. This interface will be an extension of [JpaRepository](http://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)—which gives us access to some common methods like ***save*** — and will be created in the same package of the ***User class***.

**1.2.3** The endpoint that enables new users to register will be handled by a [@Controller class](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Controller.html). We will call this controller RegistrationController and add it to the same package as the ***User class***

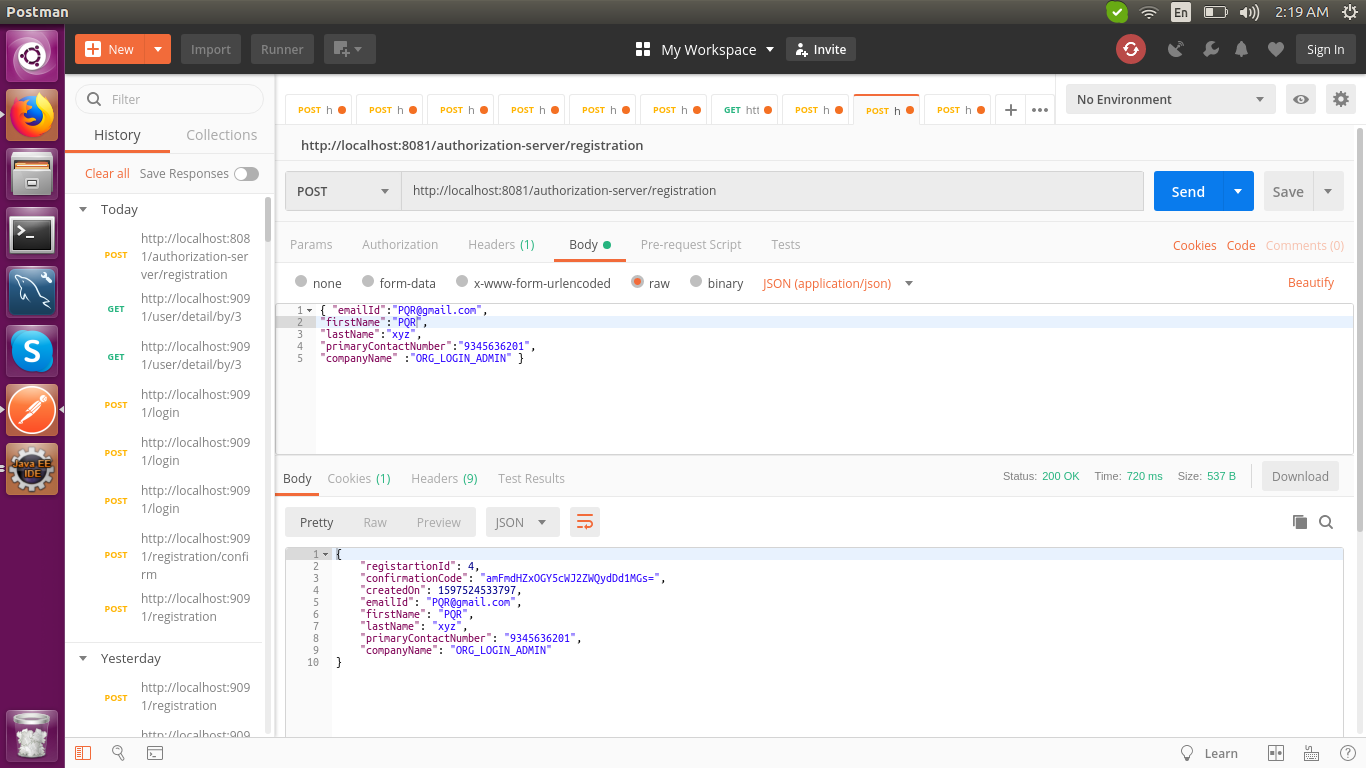
**1.2.4** It does is *BCryptPasswordEncoder* the password of the new user (holding it as plain text wouldn't be a good idea) and then save it to the database.

*1.2.5 Registration Example with Postman:*

Post Api: http://localhost:8081/authorization-server/registration

Request :{ "emailId":"PQR@gmail.com", "firstName":"PQR", "lastName":"xyz", "primaryContactNumber":"934563620","companyName" :"ORG\_LOGIN\_ADMIN\_2" }

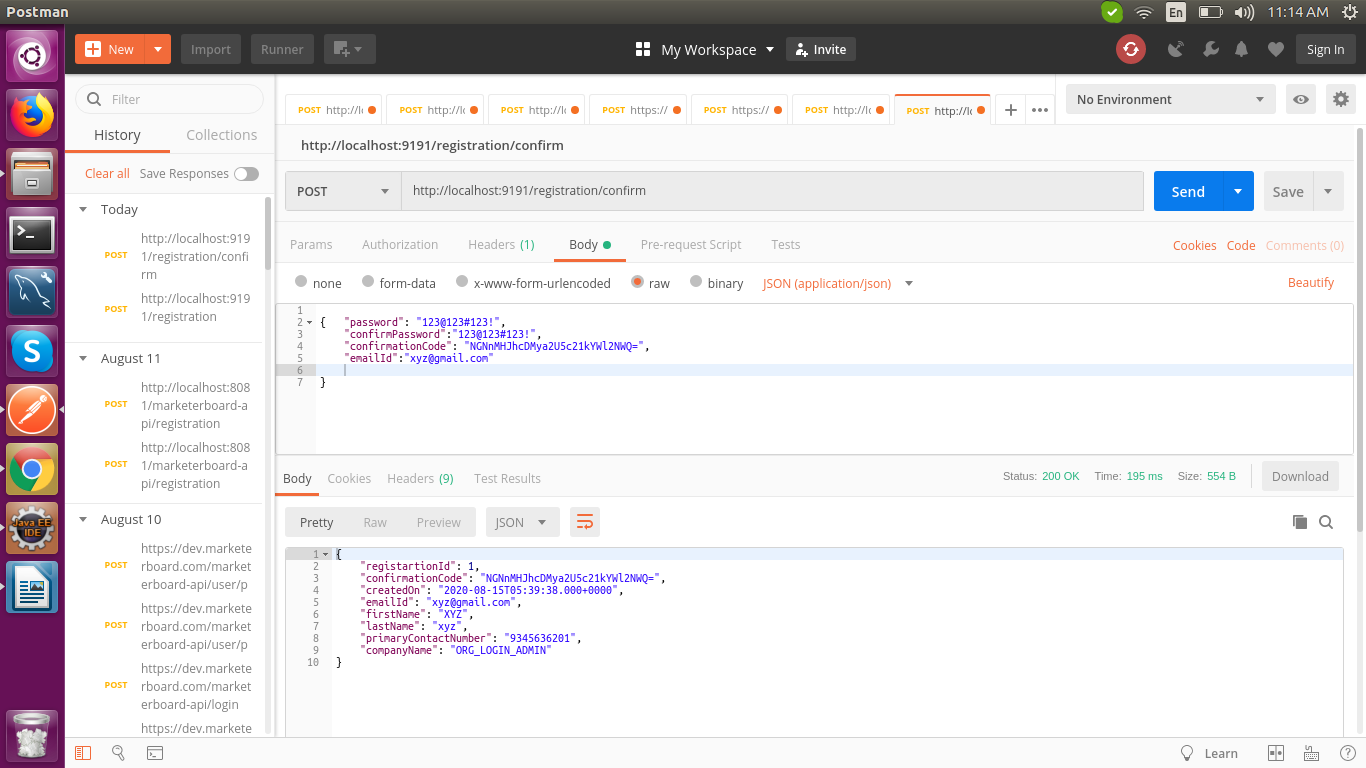
**Demo :**



*1.2.6 Confirm Password & Create Orgnization ,User Profile :*

Api : http://localhost:8081/authorization-server/registration/confirm

Request:{"password": "abc@123", "confirmPassword":"abc@123","emailId":"PQR@gmail.com" "confirmationCode":"amFmdHZxOGY5cWJ2ZWQydDd1MGs="}

**Demo :**

## **1.3 User Authentication and Authorization on Spring Boot**

***1.3.1 Aauthentication vs. Authorization :***

* Authentication is the act of a user proving who they are by providing credentials.
* Authorization determines whether a user is allowed to do whatthey’re trying to do.

For instance, the user could prove his identity by providing a *user ID and password*, but he may not be authorized to look at sensitive data such as payroll data. For the purposes of our discussion, a user must be *authenticated before authorization* takes place.

***1.3.2 Registering client applications with the OAuth2 service***

* The first thing to notice in the code is that you’re extending Spring’s *AuthenticationServerConfigurer* class and then marking the class with a *@Configuration* annotation.
* It provides the basic mechanisms for carrying out key authentication and authorization functions.

*For the OAuth2Config class you’re going to override three methods :*

* The first method, configure(),is used to define what client applications are registered with your authentication service. The configure() method takes a single parameter called clients of type ClientDetailsServiceConfigurer .
* The first thing you do in this method is register which client applications are allowed to access services protected by the OAuth2 service.
* I’m using “access” here in the broadest terms, because you control what the users of the client applications can do later by checking whether the user that the service is being invoked for is authorized.

**For example:**

clients.inMemory().withClient("client")

.secret(passwordEncoder.encode("password"))

.authorizedGrantTypes("password","authorization\_code","refresh\_token","implicit")

.scopes("READ", "WRITE");

* The two method calls *withClient() and secret()* provide the name of the application that you’re registering along with a secret that will be presented when the application calls your OAuth 2 server to receive an OAuth 2 access token.
* The next method, a*uthorizedGrantTypes()* , is passed a comma-separated list of the authorization grant types that will be supported by your OAuth 2 service. In your service, you’ll support the password and client credential grants.
* The *scopes()* method is used to define the boundaries that the calling application can operate in when they’re asking your OA uth 2 server for an access token.

***1.3.3 Defining the User ID, password and roles for your application***

create a custom implementation of [*UserDetailsService*](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetailsService.html) to help Spring Security loading user-specific data in the framework, and extend the class to customize the security framework to our needs.

To set up users (and their roles),start by extending the *WebSecurityConfigurerAdapter* class and mark it with the *@Configuration* annotation.

You need to provide the OAuth2 server a mechanism to authenticate users and return the user information about the authenticating user.

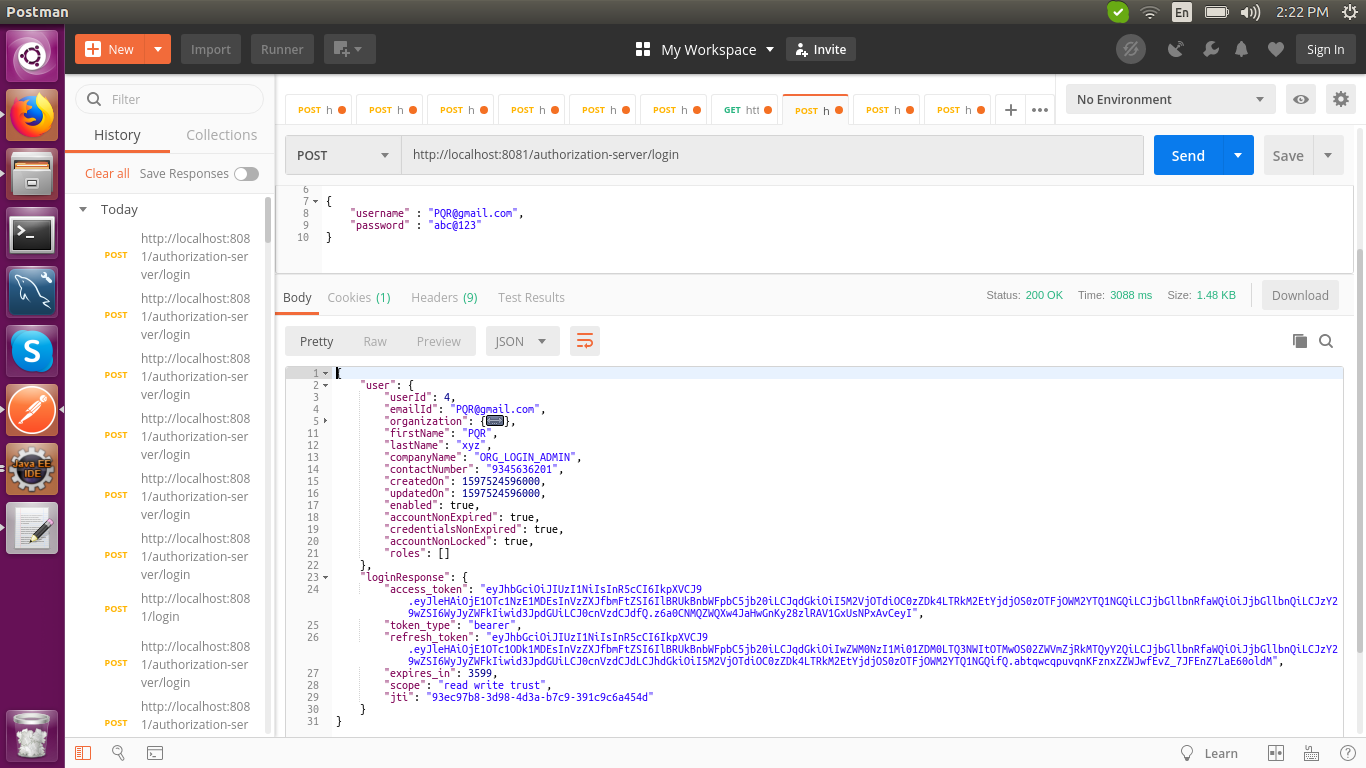
Two beans in *WebSecurityConfigurerAdapter* implementation:

1) AuthenticationManagerBean().

2) UserDetailsService() .

*Step2 : User Login on Spring Boot APIs*

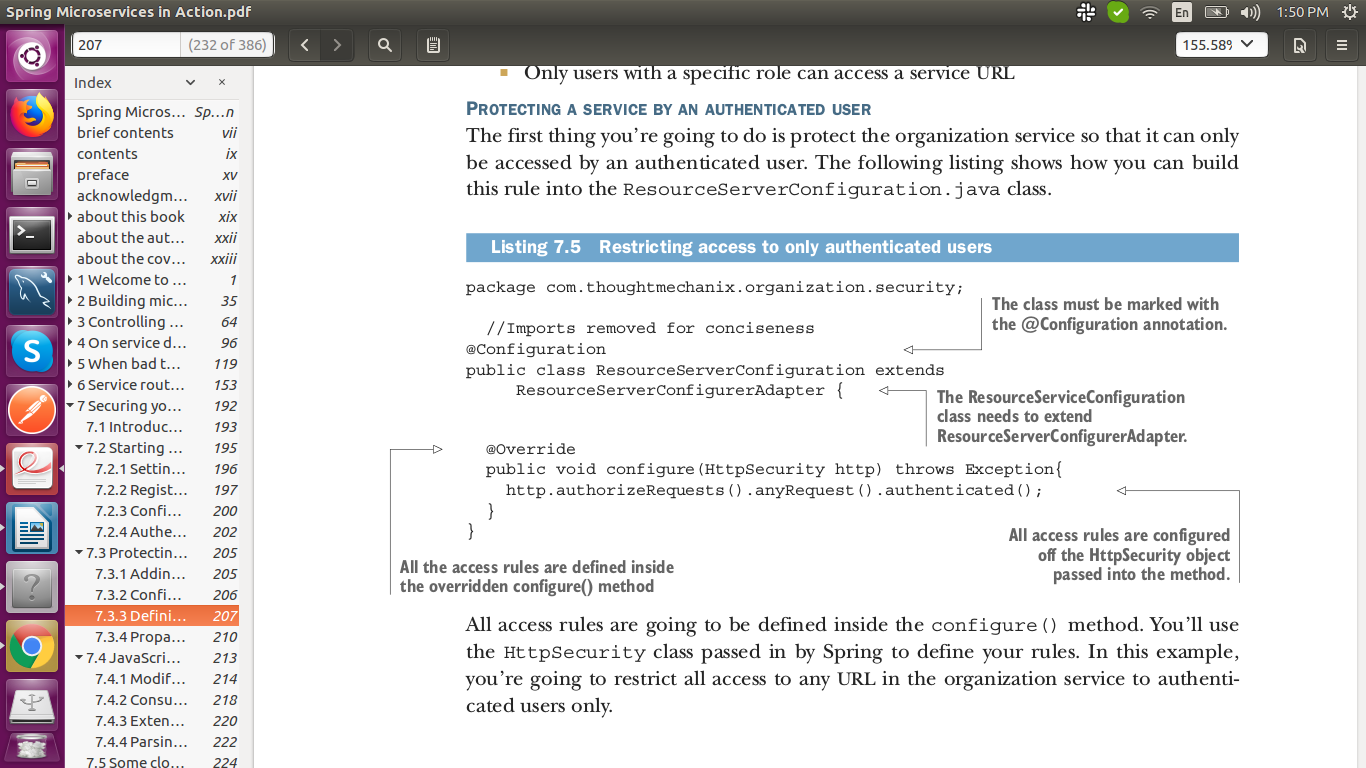
**Api :** <http://localhost:8081/authorization-server/login>



**1.4 Restricting access to only authenticated users**

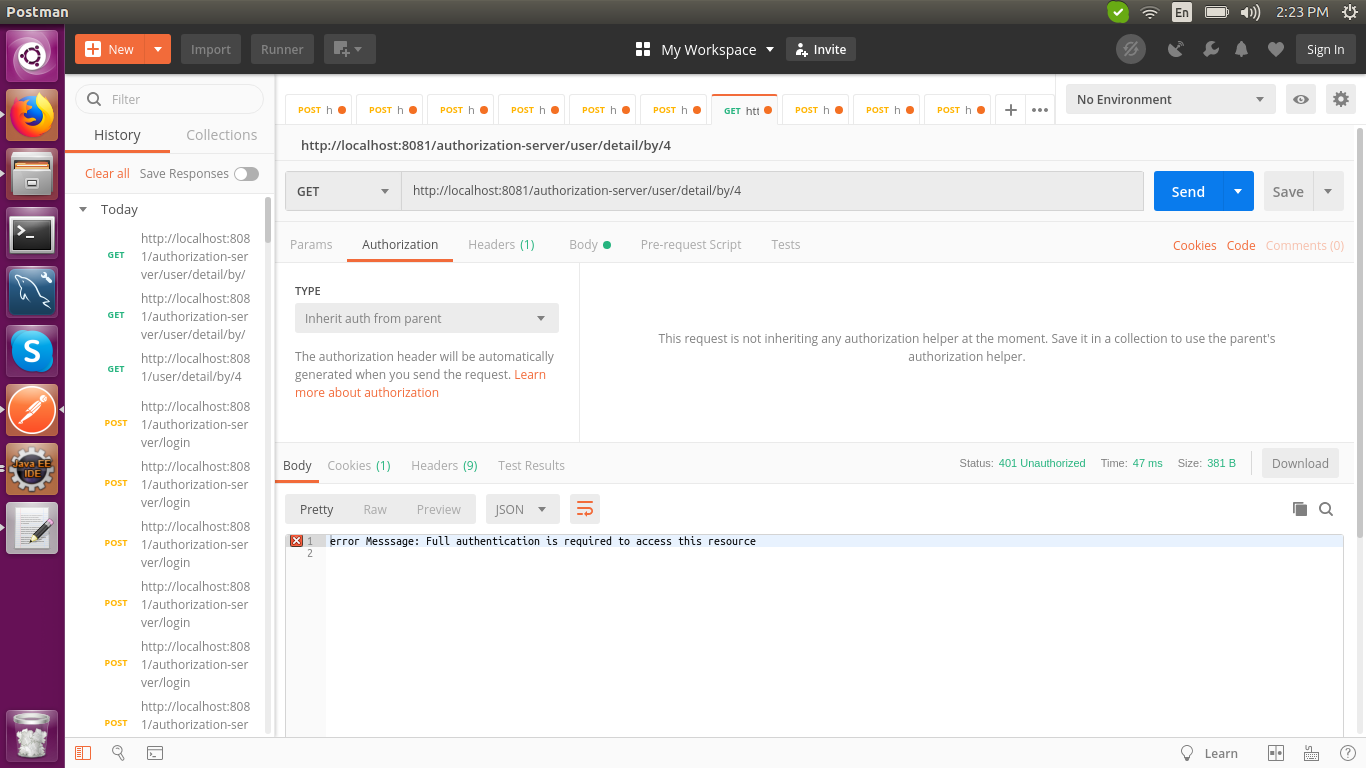
The first thing you’re going to do is protect the user service so that it can only be accessed by an authenticated user.

The following listing shows how you can build this rule into the *ResourceServerConfiguration.java* class.



* All access rules are going to be defined inside the configure() method. You’ll usethe HttpSecurity class passed in by Spring to define your rules.
* In this example,you’re going to restrict all access to any URL in the organization service to authenticated users only.
* If you were to access the user service without an OAuth 2 access token present in the HTTP header, you’d get a 401 HTTP response code along with a message indicating that a full authentication to the service is required.

*Step3 This API is not allowed to be executed without authentication (access\_token)*

****

* when you call the user service,you need to add an HTTP header called Authorization with the value Bearer access\_token value .

*Step4 Get user by Id API with access\_token*

