REST:

~ Postman is a REST Client which is used to send & receive data with server. In realtime app, frontend will be communicating with server. But here for learning purpose we use POSTMAN.

~REST – Representational State Transfer

State: The current data which u send is called as State. (i.e)When a client requests a data(resource), server fetches & sends the current data(State).

Representational: Ther server sends the data in a format(like json). This type of sending in a format is called Representational.

Transfer: It refers to the data being transferred btwn the client n server.

So, the REST is nothing but a API which is used to send & receives datas(resources) from client to server and viceversa.

REST uses HTTP methods to achieve this. Think in this way, we use URL’s to all actions(CRUD Operations). REST is executed via URL to perform all the CRUD operations.

Eg: To get a data, we use url “localhost:8181/jobs”.

To insert and to update a data, we use the same url, “localhost:8181/job”. This will cause confusion. For this only REST uses HTTP methods like POST,PUT,GET,DELETE etc..

With this HTTP methods, we can specify the method so that the server executes accordingly without any confusion. We shld mention the method in both server(in controller method) and client side(in url in POSTMAN).

Purpose of @Controller & @RestController: @Controller is a normal Controller which will return a jsp page defaultly and @RestController is a Rest Controller which will return a data defaultly in methods.

@RequestBody – To get the data from the client(what we type in request in Postman and send).

@ResponseBody – To get the data from the server(what we get in response in Postman).

->By default, spring returns the data from controller methods in JSON format. We can also get and return values in xml format by adding “jackson-dataformat-xml” library.

SPRING DATA JPA:

Using Spring ORM and its tool Hibernate, we can connect with DB and perform the CRUD operations. But it takes a lot of code. So we use Spring Data Jpa, by which we can perform the CRUD operations with very less no of lines of code.

->Spring ORM and Hibernate is a implementation of Spring Data JPA.

In Spring Data JPA, we can make the JPA create the table of entity class in mysql/postgresSQL by using the property, “spring.jpa.hibernate.ddl-auto=update”. This will create a table in mysql, when u run the app for first time and then from sec time, it wont create the table and just update it. If u use ‘create’ instead of update, then everytime u run the app, it will create a new table with same name in DB.

Custom query in Spring Data JPA:

**public** **interface** ProductRepository **extends** JpaRepository<Product, Integer> {

@Query("select p from Product p where p.name=?1") //In @Query, instead of table name, \*(star) and row names, u shld u give Class name(Product) and obj name(p) and Class variable name(name). This query is called JPQL(JPA query language). ?1 means the argument in first place(String name) in getByName method. So, When u pass a name in this method it will take n put in the query in place of ?1.

List<Product> getByName(String name); //Return type is List, bcoz name is not Primary key and many same names can be present in the table.

}

SPRING DATA REST:

Spring Data Rest is advanced than Spring Data JPA. We don’t need Controller and Service Layer in Spring Data Rest. It will automatically takes care of all the http functions. Give the endpoint in Http request path as Model class name(jobPosts). Just we need to change the methods in postman. Endpoint(jobPosts) will be same for all core methods like POST,GET,PUT,DELETE etc..

Spring Data REST is widely used in the industry where rapid development and standard CRUD operations are needed. However, for applications requiring extensive customization and complex business logic, @Controller based approaches are still more prevalent.

Many organizations use a hybrid approach, leveraging Spring Data REST for straightforward CRUD operations and @Controller for more complex use cases.

Use Cases of Spring Data Rest:

**-**Applications that primarily perform CRUD operations on entities and require rapid development.

**-**Quick prototyping of RESTful APIs to validate ideas or gather early feedback.

**-**Simple microservices where each service is responsible for managing a single entity type.

Custom queries in Spring Data Rest:

Adding custom methods to a Spring Data REST repository involves creating a custom repository and defining the custom methods in a service or repository class.

- Create methods within your repository interface that extend CrudRepository .

- Annotate these methods with @Query if they involve complex JPQL queries. Otherwise, Spring Data REST might infer the query based on method name.

- Use the @RestResource annotation on your custom methods to control their exposure as REST endpoints.

By default, custom methods inherit the HTTP method constraints from standard CRUD operations (GET, POST, PUT, DELETE).

You can't modify core CRUD methods exposed by Spring Data REST.

In your application.properties, ensure that Spring Data REST is configured properly:

spring.data.rest.base-path=/api

**SPRING SECURITY:**

By adding spring security dependency in pom.xml, our app will become secure in a basic level. When we run the app, before taking any request, it will ask for credentials in login form.

OWASP is the website which shows top 10 security loopholes in current, that we should take care in our applications.

Spring security works in as filters, the same that we saw in Servlet concept. There will be layers of filters through which the security checks will be done.

A screenshot of a computer

Description automatically generated

All the filters will be in a order and it will be executed in a sequential order one by one. These filters are connected by Filter chain.

Note: Spring framework in behind the scenes is implemented by servlet.

Once you login, you can access all the pages till you logout (i.e) the Session ID will be same throughout the Session till you logout. Once you logout, session will get expired.

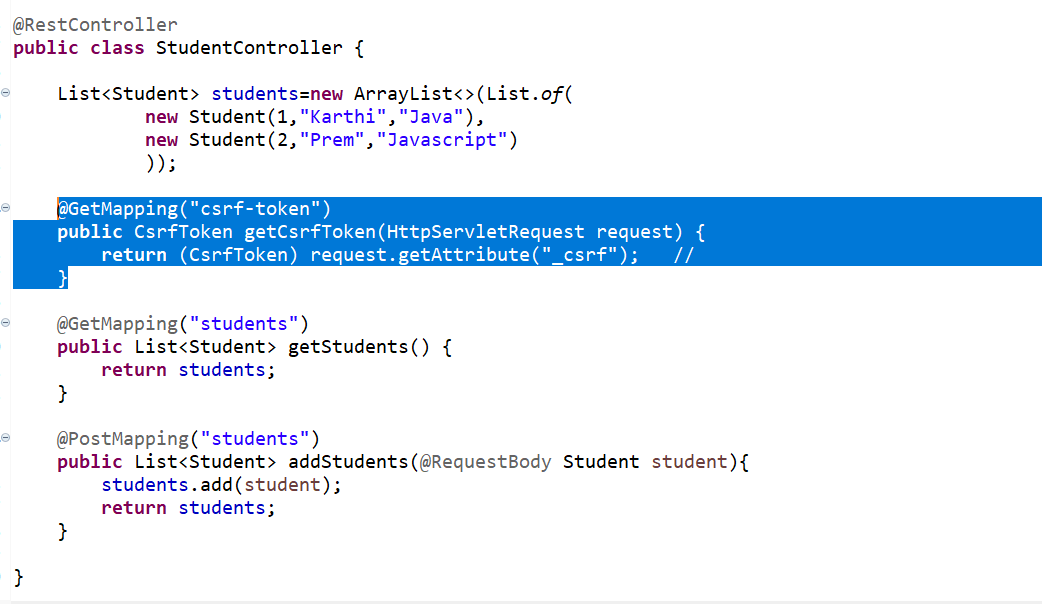
You can set ur own login credentials in application properties also. And through browser, you can secure using login form and through REST Clients like Postman, we can access using Basic Auth by giving the username & password in Authorization.

**CSRF(Cross Site Request Forgery) - Session & Cookie:**

When you login, a sessionID will be created and till you log out you can able to access every pages using that sessionID. Once you logout, it will be closed. This sessionID will be stored in Cookie of the browser.

But attackers can easily access the sessionID and rob the information. To avoid this, we should get and use CSRF tokens for every requests like POST,PUT,DELETE etc, except GET as they can’t make changes in DB using GET. This CSRF tokens will be inbuiltly given by spring security.

CSRF tokens are unique ID’s and it is highly difficult to hack it. We can get the CSRF ID by using the method:



Once we get the CSRF tokens, then we shld give it in the Headers in POSTMAN,

A screenshot of a computer

Description automatically generated

Only after giving this CSRF token here, we will be able to do any requests except GET. Else it will show 401 error.

2 other ways:

1. server.servlet.session.cookie.same-site=strict : Using this property, we can restrict the cookie only to our app site and it can’t be accessed by any other external & third party sites. But the drawback is , it will affect some functionalities (i.e) we may have to redirect to some other sites and by using this same-site property, we will restrict some functionalities in our destined site.
2. Using custom security configuration:
3. @Configuration
4. @EnableWebSecurity
5. **public** **class** SecurityConfig {
6. @Bean
7. **public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {
9. http.csrf(customizer -> customizer.disable()) //Disable CSRF Token; u can perform any crud operation without giving CSRF Token in Headers in Postman
10. .authorizeHttpRequests(request -> request.anyRequest().authenticated()) //provides security/authentication to all requests; only if u give Basic Auth in Authorization in Postman, it will give results.
11. .httpBasic(Customizer.*withDefaults*()) //Only with this method you can access any request
12. .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.***STATELESS***)); //make the session stateless, (i.e)session id gets changed for every requests; Provides more security; csrf tokens are not necessary as session id is stateless; FormLogin is not needed when we use session stateless
13. // .formLogin(Customizer.withDefaults()); //creates form login in beginning after running the app, like in default; when u r making the session stateless, then u dont have to give formlogin
14. **return** http.build();
15. }

Using this class, we can stop the default spring security and use our own given security. When you define this class and return ‘http.build()’ , default spring security will get disabled and whatever u give in this class will only be in action.

Mostly in realtime application, we use custom spring security and only once we set them up. So just grasp the concept alone and don’t worry much abt syntax.

**Builder Pattern:**



In java, for calling multiple methods we can use like as below,



Spring Security Authentication:

To secure your app with credentials that is saved in DB you shld add the bean AuthenticationProvider and Instantiate DaoAuthenticationProvider to connect with DB,

A computer screen shot of a program

Description automatically generated

When you declare this bean and return provider, you app will be secured with popup login.

provider.setUserDetailsService(userDetailsService) will do all the functionalities to check if the uname & pswrd thay you give matches with the one in DB.

UserDetailsService is a interface. So, you have to implement it with a class so that class will get autowired in UserDetailsService.

MyUserDetailsService fetches the row of uname that we give using UserRepo interface.

If the uname is not present in DB, it will throw error.

If it uname present, then it will send that entity obj to UserPrincipal and that will return uname, pswrd and authorities. We return that in MyUserDetailsService class’s method ‘loadUserByUsername()’.

We can also directly return the user obj in MyUserDetailsService class, but the return type is UserDetails. So we have to call that and from there we have to return. Also UserDetails interface contains many other methods to authorize like, if the uname is expired or locked etc..

Provider obj in AuthenticationProvider will get the credentials from MyUserDetailsService and it will return it. Then spring checks if the uname & pswrd & authority we gave matches with the row in DB.

BCryptPasswordEncoder:

Till now, we have used password as a simple text. But in real time application we will encoded passwords.

For this, spring has BCryptPasswordEncoder.

2 Steps:

(1)Registering the user password in encoded form:

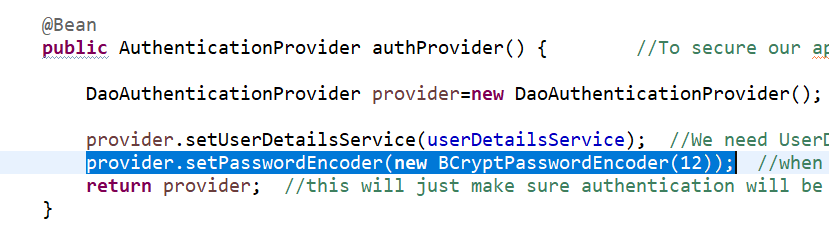
When you register a user, in service method you have to encode the password using BCryptPasswordEncoder and then save it into DB.

A computer code with text

Description automatically generated with medium confidence

(2)Accessing password in encoded form:

While you enter uname & password, we have to convert that password into encoded password, bcoz in DB, password is saved in encoded form. So that spring will check both the given password and encoded password in DB.



Setting passwordEncoder here will make sure whatever password you enter to login, will gets converted to encrypted form.

Note: the strength you pass in BCryptPasswordEncoder**(12)**, should be same in both steps.