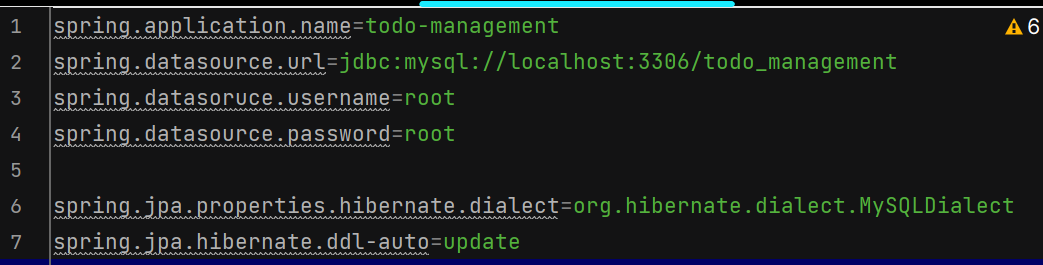
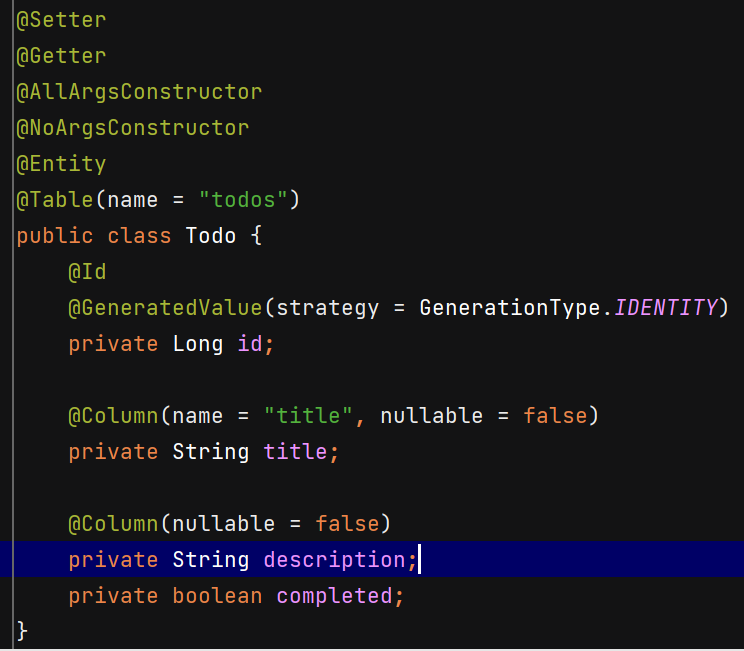




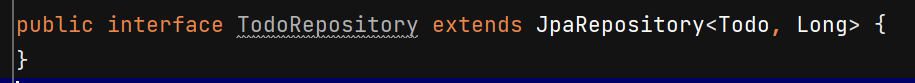
**Configure MySQL Database**

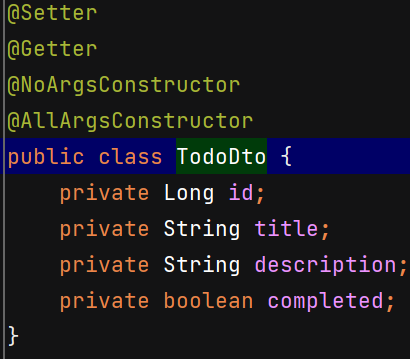


**Create Todo JPA Entity**



**TodoRepository**



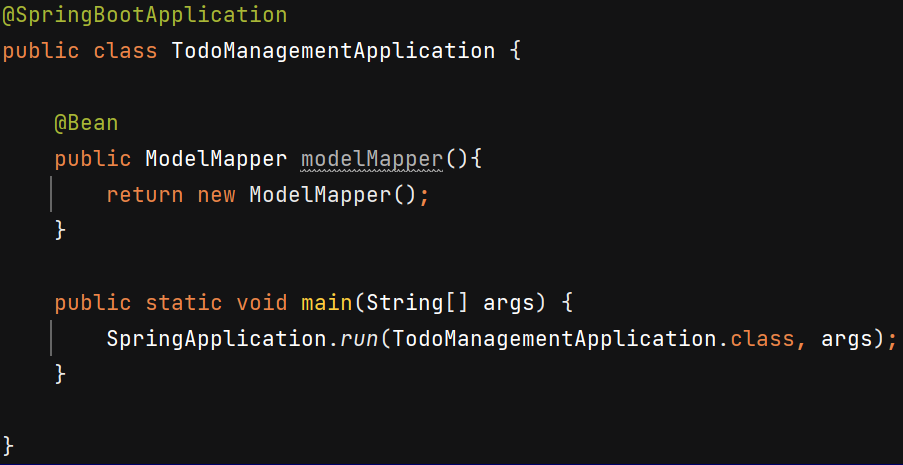
**TodoDto** 

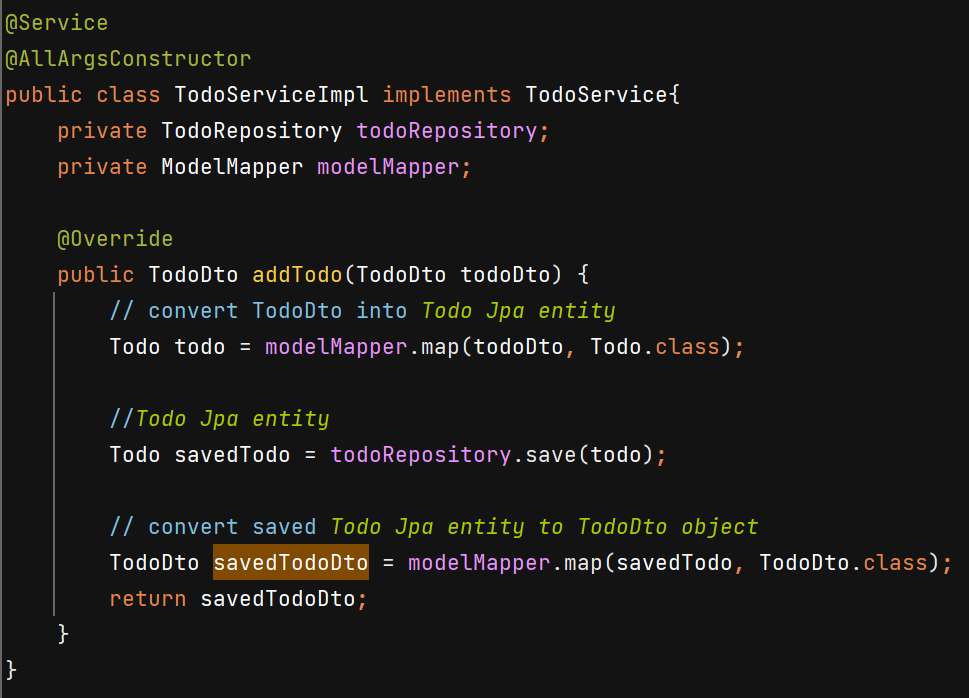
**Build Add Todo REST API**



ModelMapper library is used to convert dto into jpa entity and vice versa.

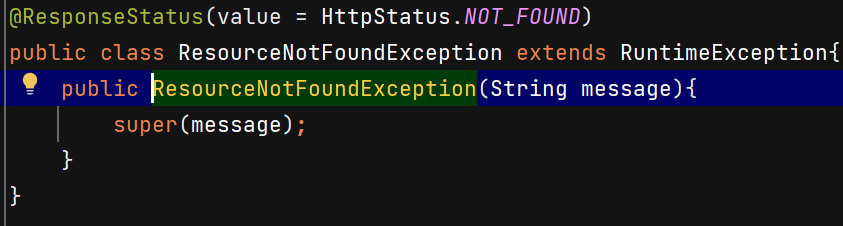
Steps – include its dependency in pom.xml, register its bean, inject it in serviceImpl class and then use it.

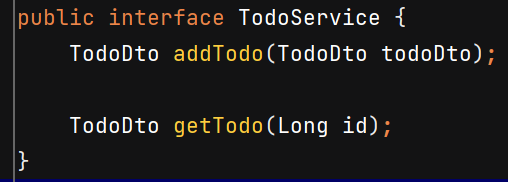




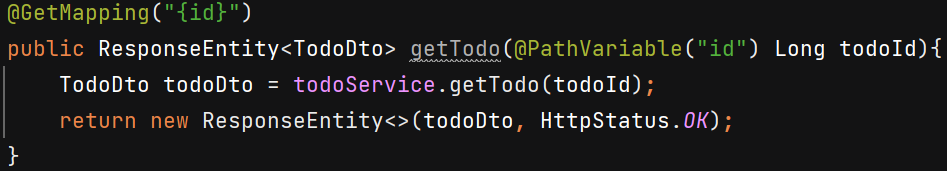


**Build Get Todo REST API**

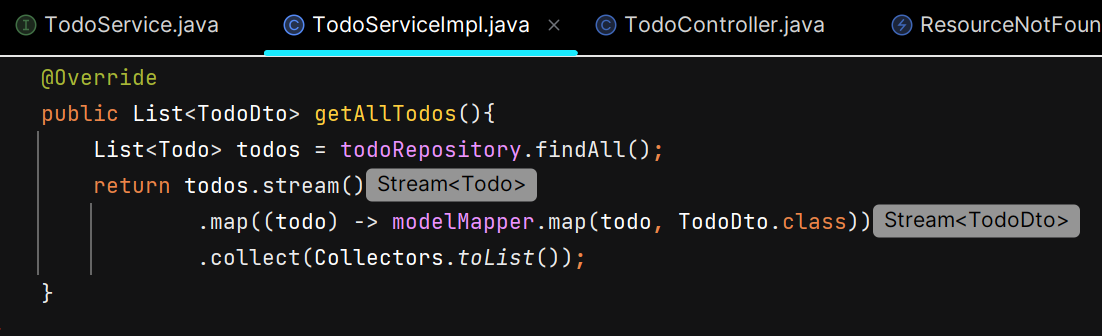


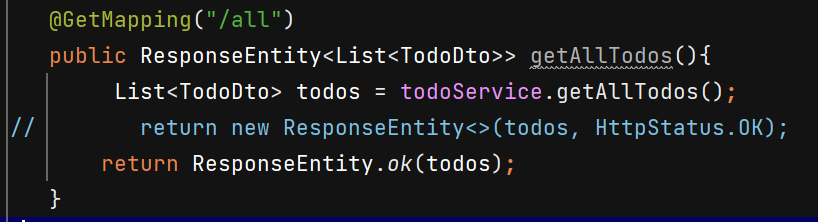






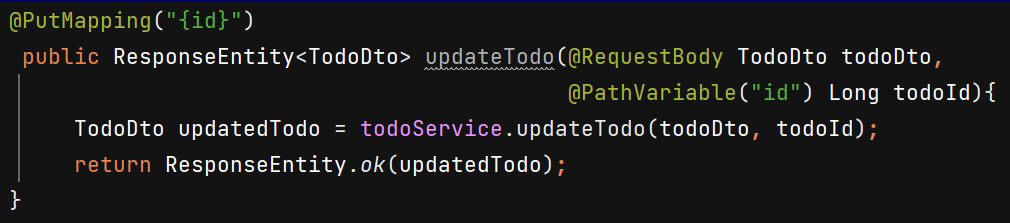
**Build Get All Todos REST API**



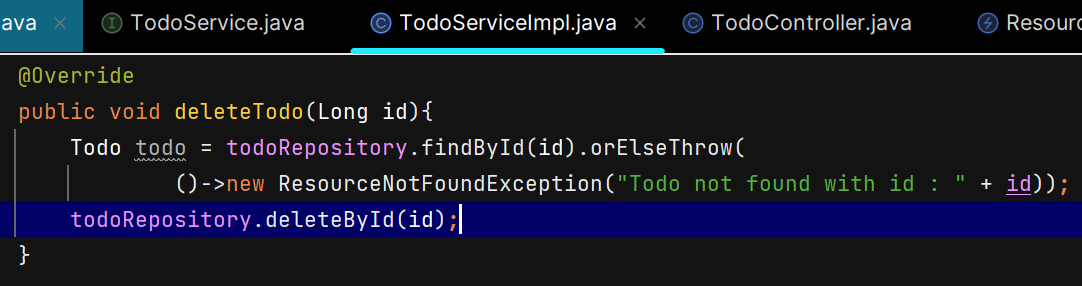


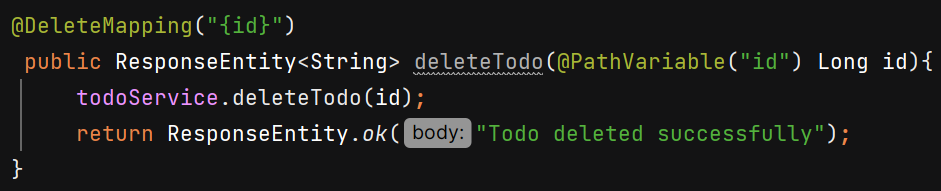
**Build Update Todo REST API**





**Build Delete Todo REST API**

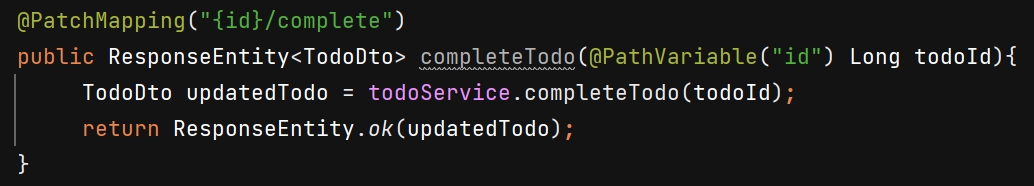




**Build Complete Todo & InComplete Todo REST API**

We use PutMapping to update the resource completely and PatchMapping to update it partially.





**Authentication**

Process of verifying the identify of a user or system attempting to access a resource. Ex- password based, biometric, multifactor authentication etc.

**Authorization**

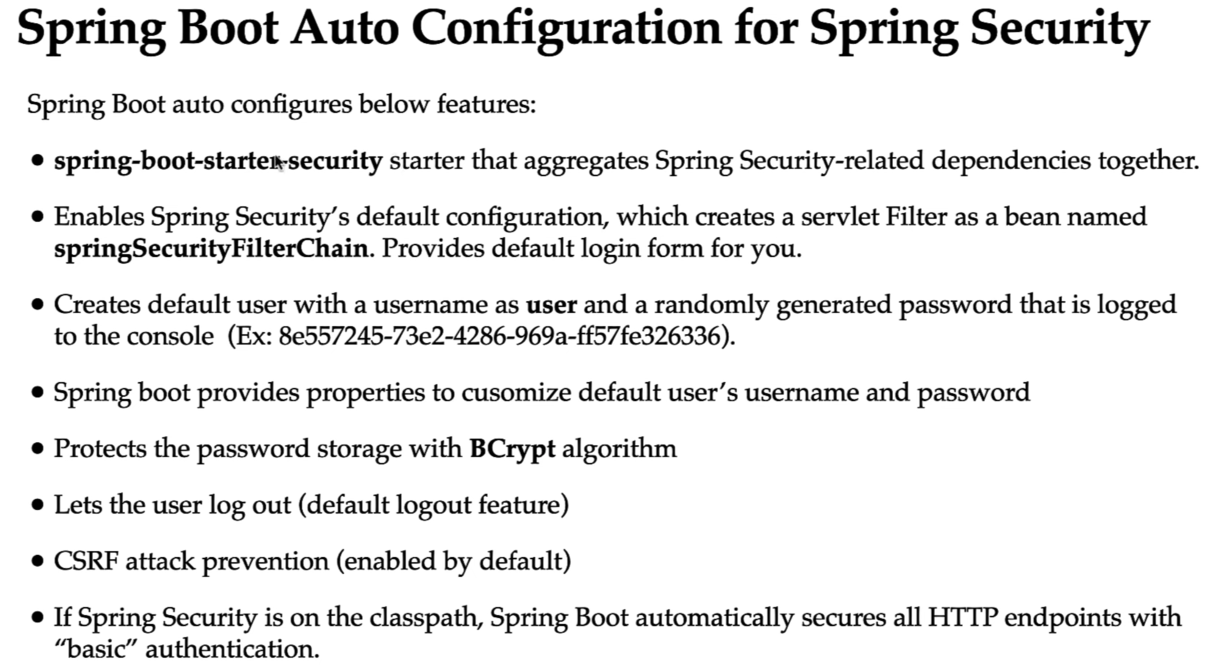
Process of determing what actions an authenticatied user or system is allowed to perform on a resource. In simple term, authorization determines what actions they are allowed to perform once their identity has been verified.

Ex- role based authentication : A user with a “manager” role can approve or reject employee leave requests, while a user with an “employee” role can only submit leave requests.

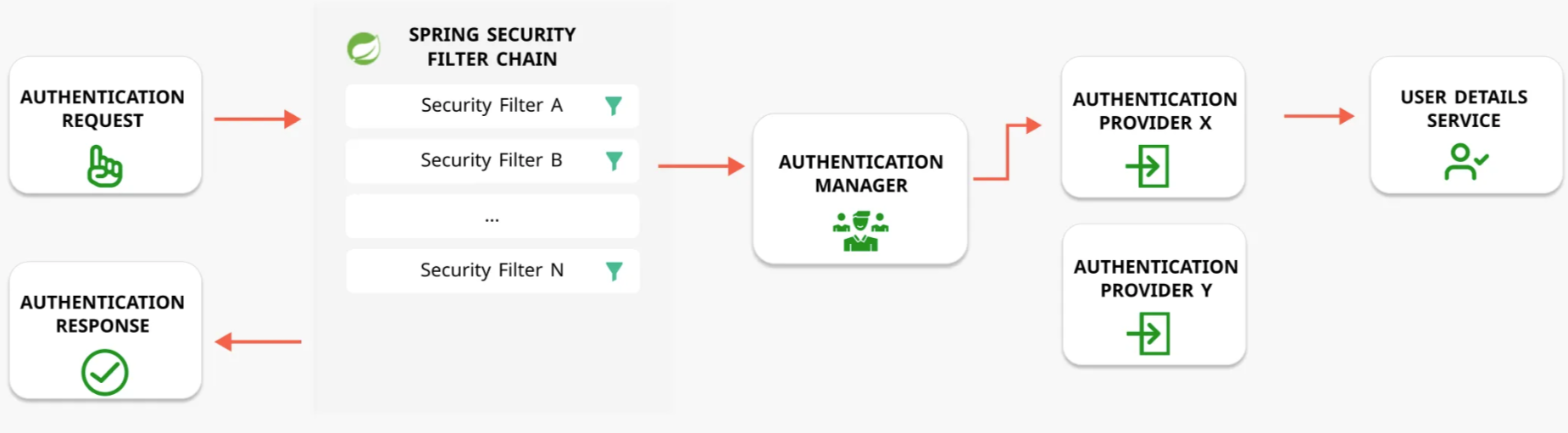
Spring security provides built-in support for authentication, authorization and projection against common attacks. To use it add spring-boot-starter-security dependency in ur project.

**springSecurityFilterChain** bean is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the login form, and so on) within your application. When an HTTP request comes in the springSecurityFilterChain delegates the request to the appropriate SecurityFilterChain based on matching conditions.

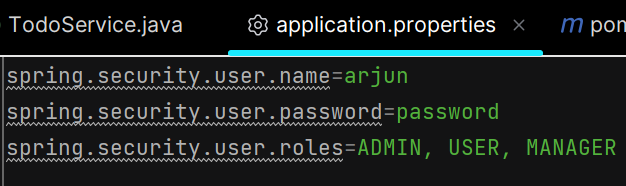
**SecurityFilterChain**: Represents individual security rules for specific request patterns.



In postman, we have to pass username & password for basic authentication.



Configure credentials like this -



**Configure Basic Authentication**

In project we don’t use form based authentication (where u get spring security default form) but we have used basic authentication.

HTTP Basic authentication involves sending a verified username and password with your request. In the request Headers, the Authorization header passes the API a Base64 encoded string representing your username:password. Browser gives us popup to enter credentials.

Inorder to configure spring security we have to create a SecurityFilterChain bean. You can define multiple SecurityFilterChain beans for different security requirements in your application. Bean name is that of method name or can be given using @Bean(name = “………”)

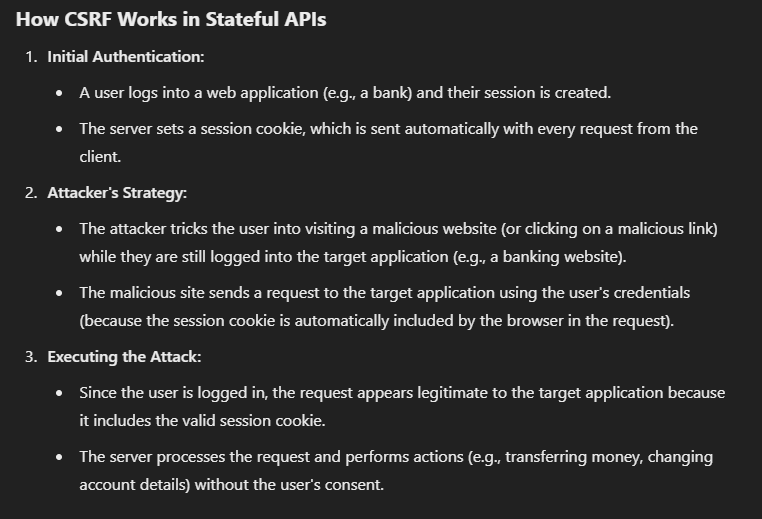
Http.build() method return DefaultSecurityFilterChain class object which is implementation of interface SecurityFilterChain.

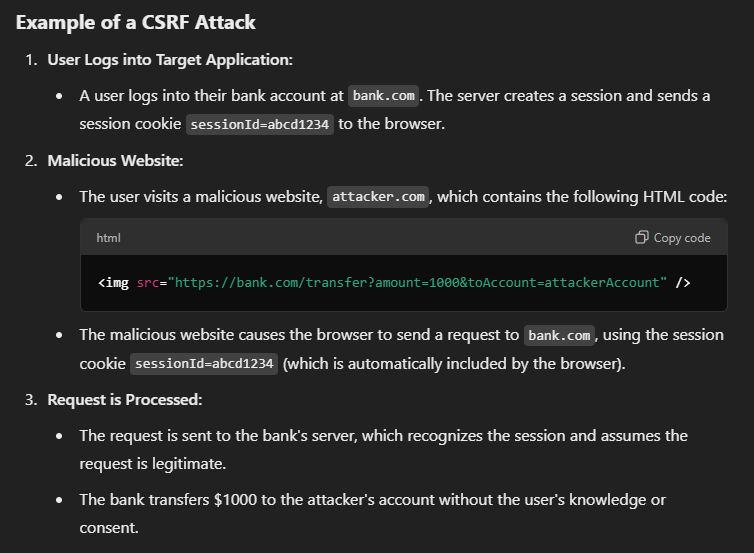
.authorizeHttpRequests() is used to authorize all the incoming http requests.

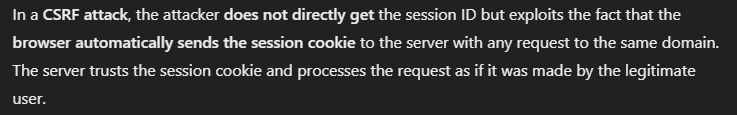


**REST API -**  specific type of web API that follow the principles of REST, an architectural style. Like its stateless which means Each request from a client to the server must contain all the information the server needs to fulfill the request. The server does not store any client state between requests. Follow standard HTTP methods such as GET, PUT, POST, etc.

**Normal API –** any api which may not adhere to REST principles. Like it can be stateful or stateless. May use any protocol – HTTP, SOAP etc. Stateful means server maintains info about client’s state across multiple requests typically using sessions. Once authenticated, a session ID (or similar identifier) is sent back to the client, typically in a cookie. Client sends the session ID with every subsequent request, usually as a cookie/header and server relies on it to authenticate requests. No need for the client to re-authenticate as long as the session is active.



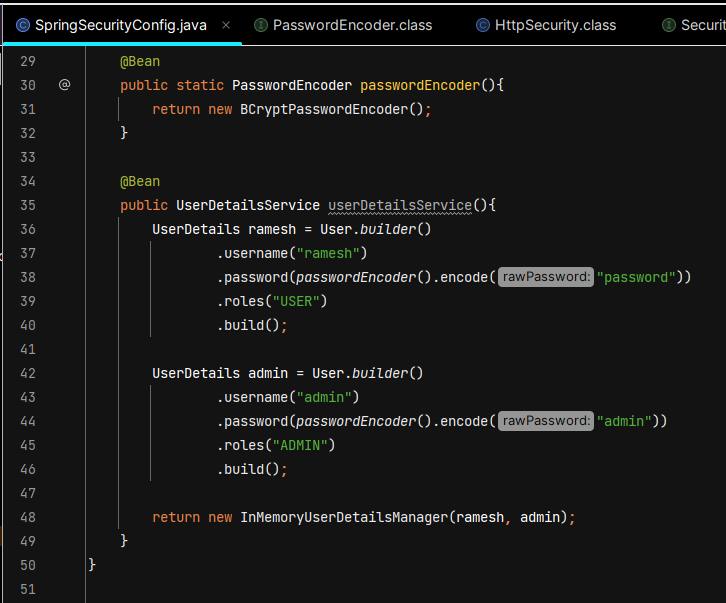




**Configure multiple users in In-Memory object**

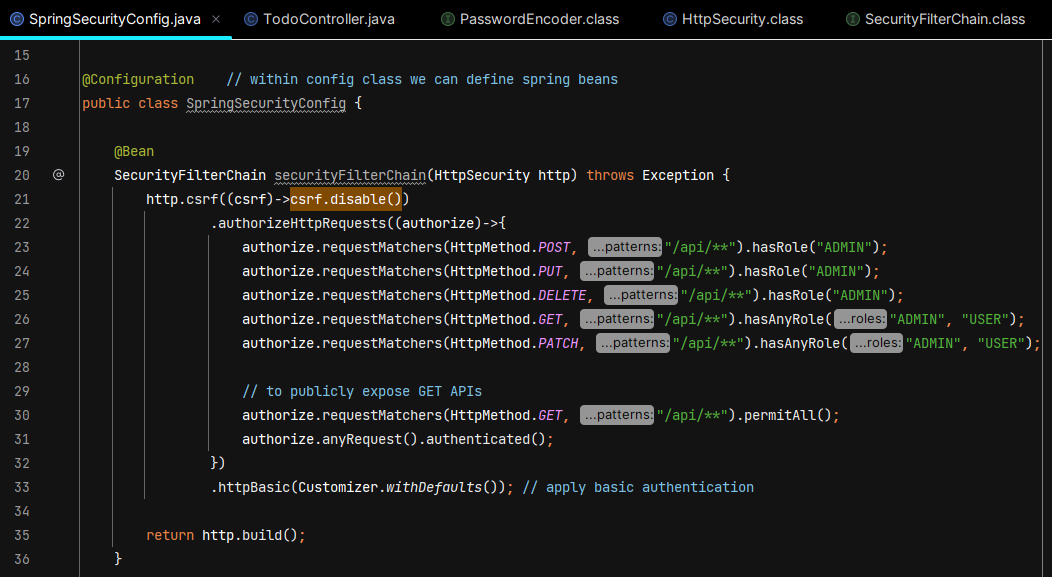
In the Spring Framework, a **Spring Bean** is an object that is managed by the Spring **Inversion of Control (IoC) container.**

Spring security expects password in encoded format so we create PasswordEncoder bean.



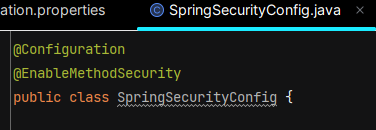
**Role-Based Authorization**

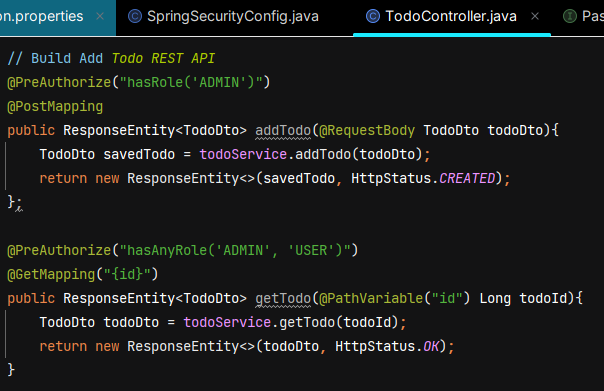
Only ADMIN can access add, update and delete todo REST API



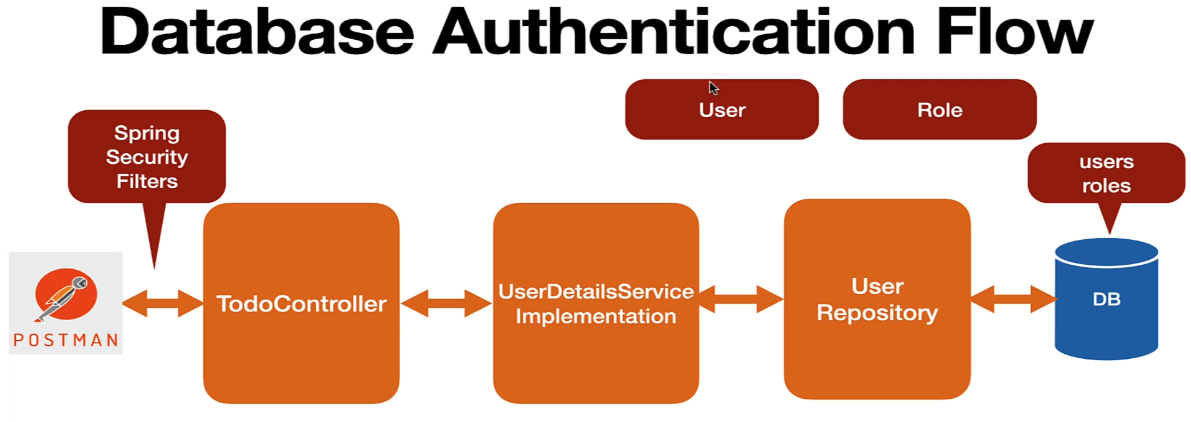
**Method Level Security**

We need to use @EnableMethodSecurity and @PreAuthorize





**Database Authentication Flow**



Request from Postman will first come to spring security filters, which extract the username and password from header and load the corresponding user object from the DB and validate username & password.

We’ll create User & Role jpa entity and hibernate will create users & roles tables in DB. We have many to many mapping between user and role as 1 user can have multiple roles and 1 roles can be assigned to multiple user.



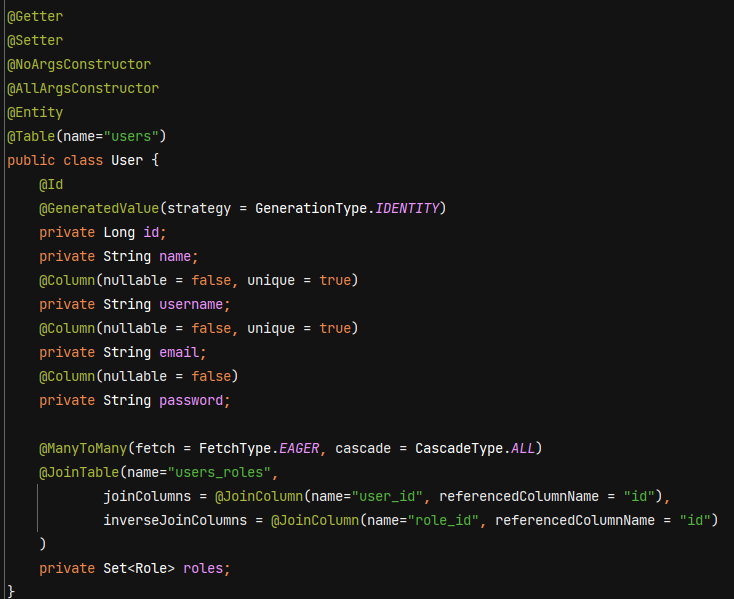
Whenever we create many to many mapping between 2 jpa entities then 3rd table is created called join table. It maintains details from users and roles table.

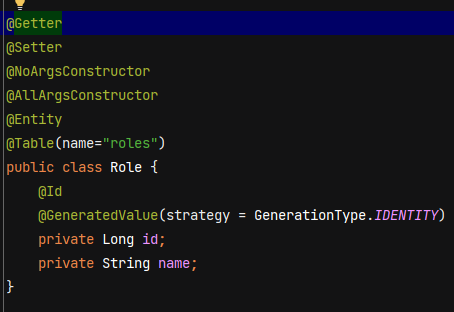
Users table contain has a primary key ID, this primary key becomes a foreign key in 3rd table. Roles table has primary key ID, this primary key ID becomes a foreign key in a third table.

FetchType.EAGER means whenever we load User entity it will also load its role.

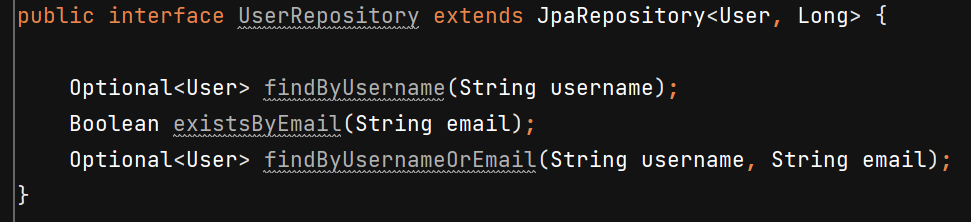
In SQL, a **cascade** is a mechanism used to ensure that changes made to one table automatically propagate to related tables, maintaining referential integrity. **CascadeType.ALL** propagates all operations (persist, merge, remove, refresh, detach) from the parent entity to the associated child entities.

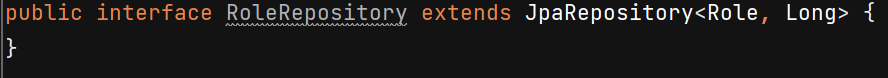
Whenever we save User it will also save Roles as User is parent and Role is its child.

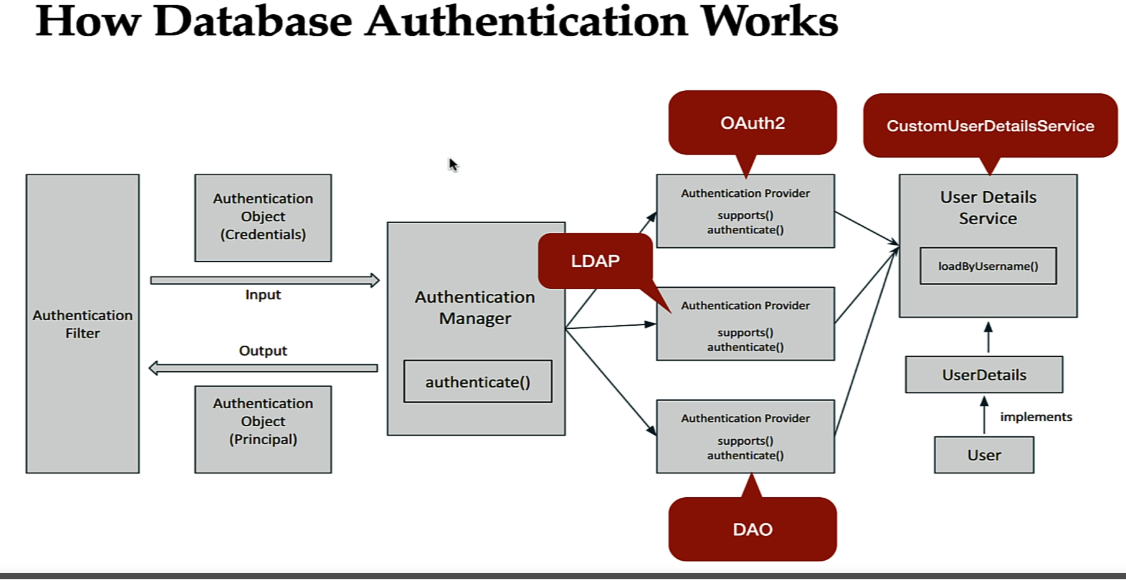




We’ll create some custom query method in UserRepository. findBy is a keyword which retreive entity by field name. existsBy is a keyword which check whether user object is present in database with given fieldname. Spring data jpa will internally create query with these method name.





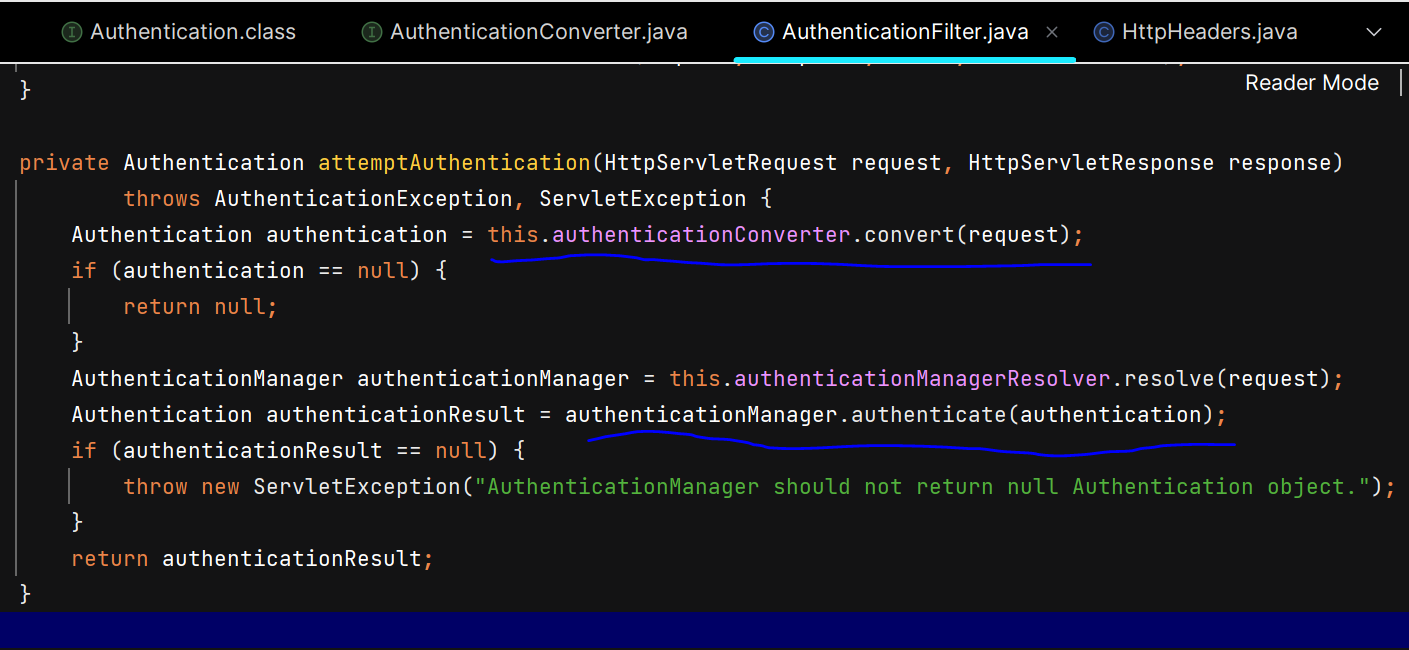


Whenever user submit a request, the username and password are stored in header and the request first comes to AuthenticationFilter, and then AuthenticationFilter will create an instance of Authentication object. This object basically contains the username and password, and then AuthenticationFilter will pass this authentication object to AuthenticationManager. This component manages different authentication providers.

AuthenticationManager’s authenticate() method internally call this supports() method of all these authentication providers to check with authentication provider provides a support. If supports() returns true, the authenticate() method of the corresponding AuthenticationProvider is invoked.

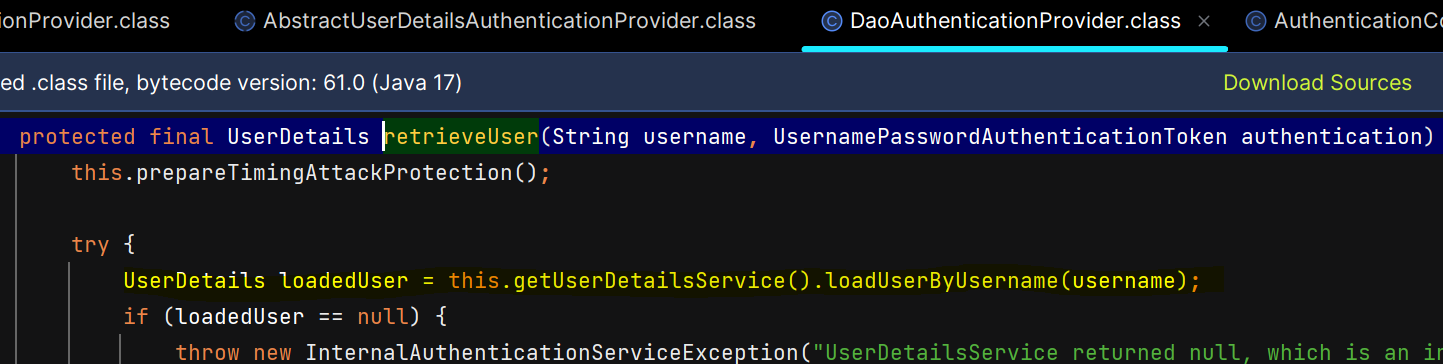
Each authentication provider uses loadByUsername method to load the user object. In case of database authentication we have to create a CustomUserDetailsService class that implements UserDetailsService interface, and then we need to provide impl for loadByUsername method.

Internal working - Request comes to doFilterInternal() method of AuthenticationFilter class. Inside this we create Authentication object by extracting the “Authorization” header which contains username:password in base64 format and finally object of UsernamePasswordAuthenticationToken is created. This class internally implements the Authentication interface.



DaoAuthenticationProvider is responsible to authenticate the requests related to database.

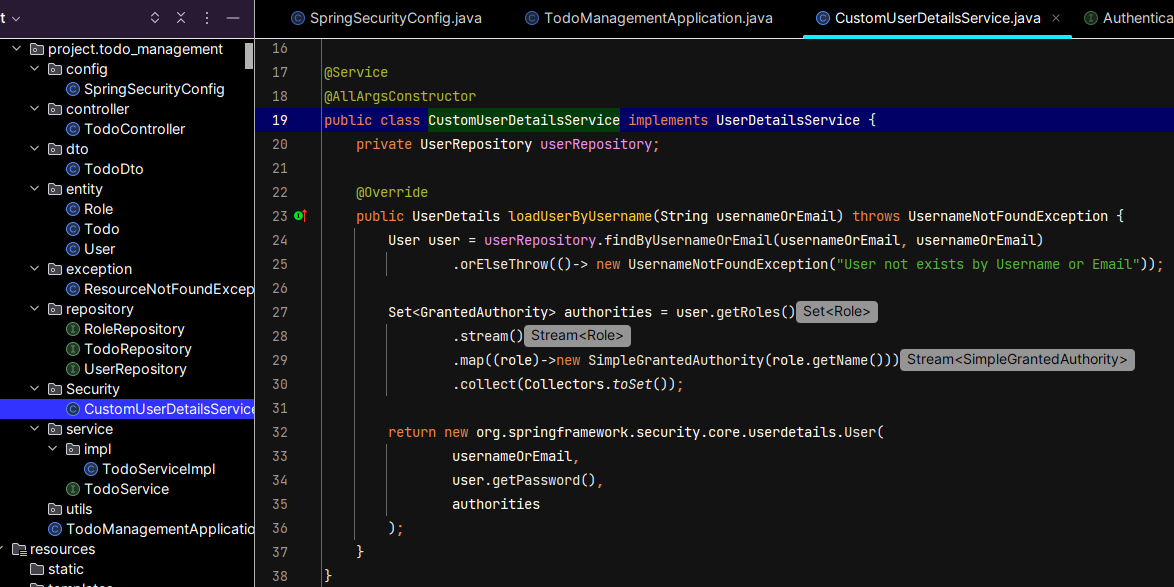




**Creating CustomUserDetailsService class**

we’ll provide implementation for loadByUsername() method to load the user from db and give it to DaoAuthenticationProvider which will provide that user to the authentication manager.

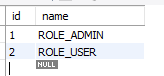
User object has a set of Role, so we convert a set of Role into a set of GrantedAuthority.



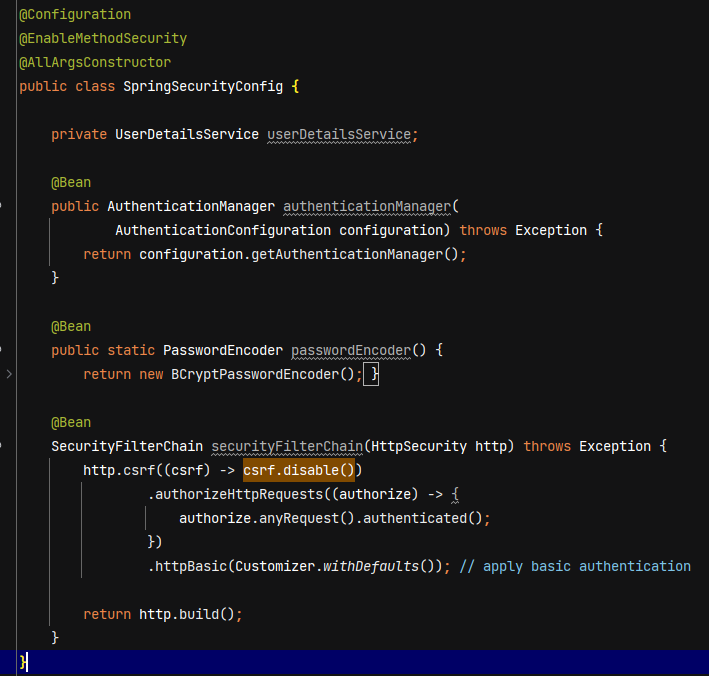
**Database Authentication**

We’ll use UserDetailsService interface to inject the dependency so to achieve loose coupling.

spring security internally add ROLE\_ prefix for roles

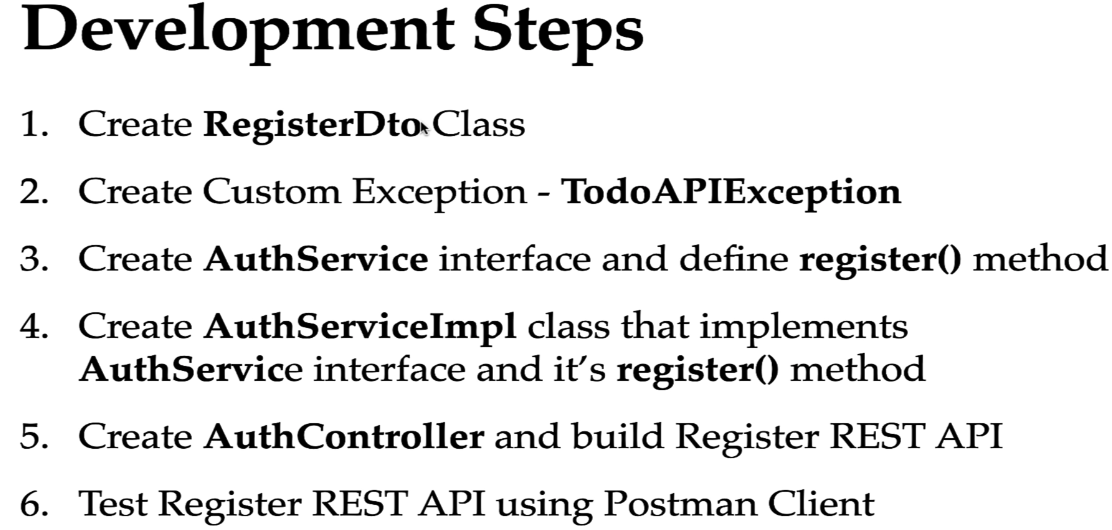
  

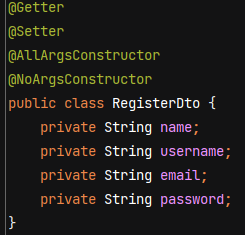
Whenever we inject UserDetailsService in a SpringSecurityConfig class, then spring security 6 will automatically use this UserDetailsService and it will call its loadUserByUsername method. We don’t have to explicitly provide this UserDetailsService instance to the authentication manager.



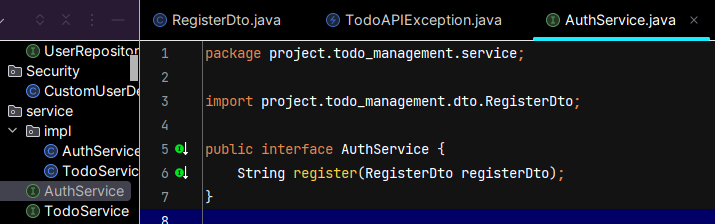
We stored password in DB using Bcrypt encoder. Spring Security uses the configured PasswordEncoder (e.g., BCryptPasswordEncoder) to hash the plain text password.

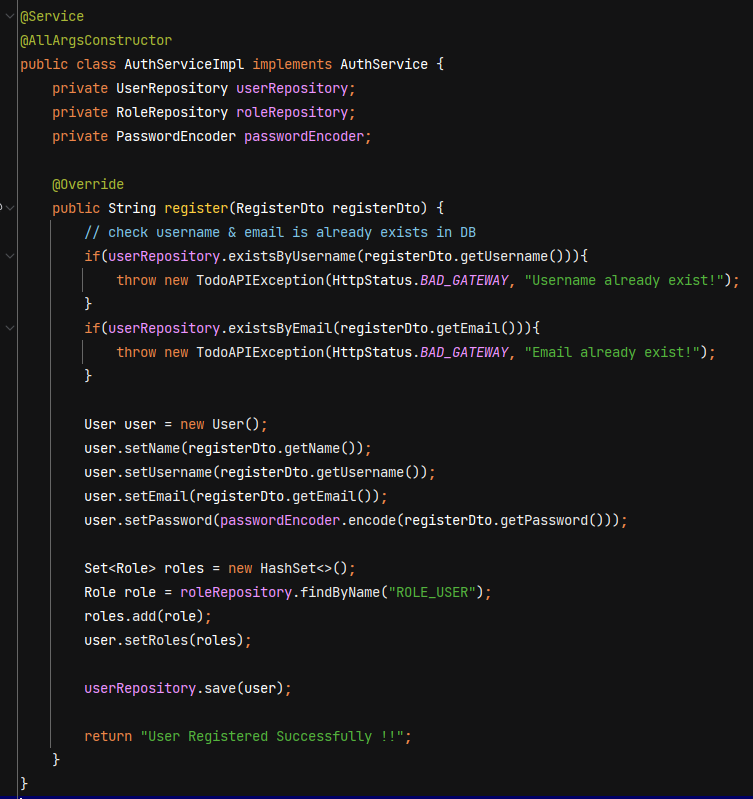
**Build Register REST API**

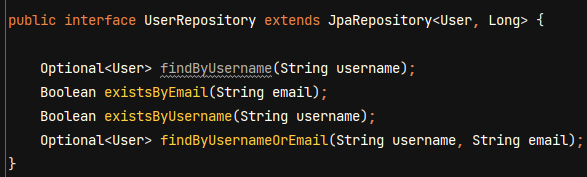
****

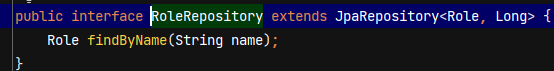


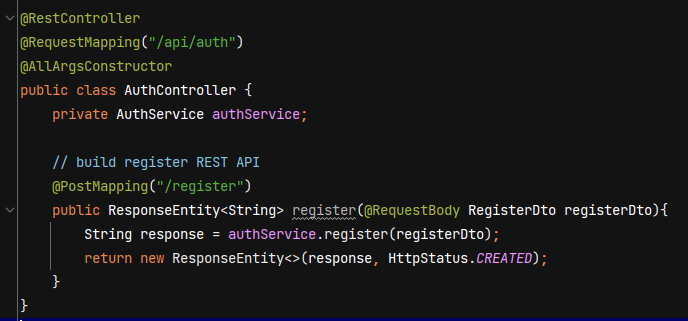












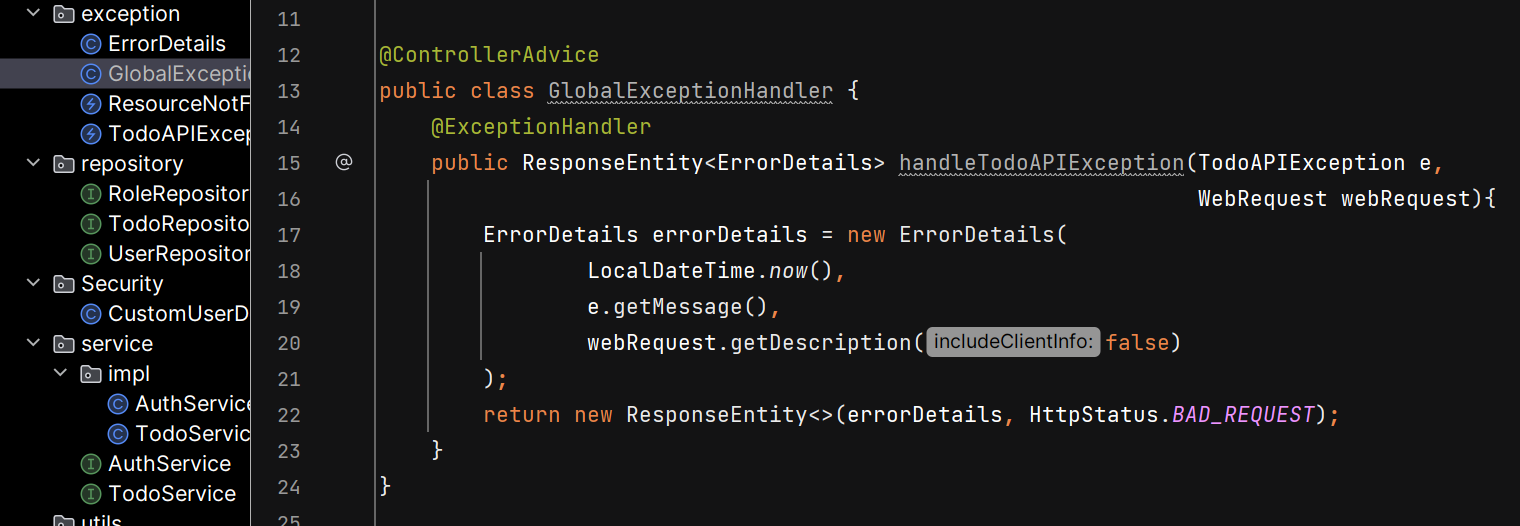


**Exception Handling in Register REST API**

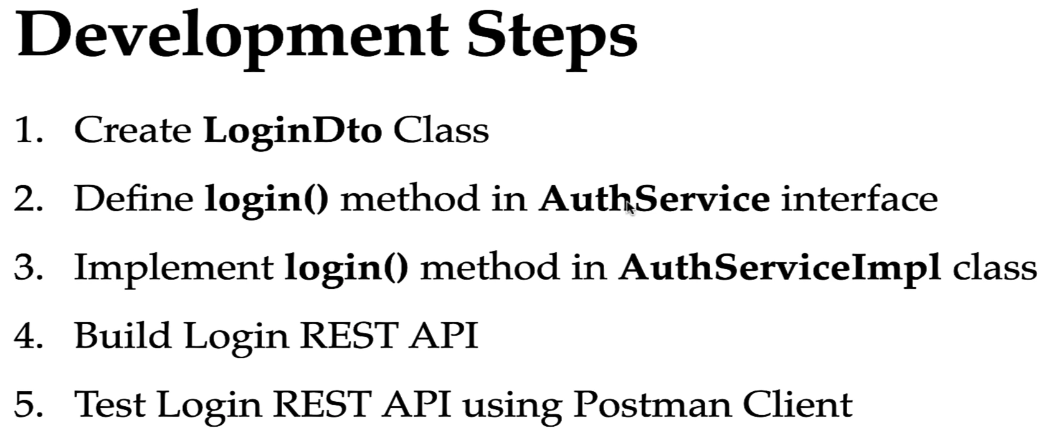
When user already exist in db and we still submit register rest API then we should get proper exception details in response. Within GlobalExceptionHandler class we’ll handle all the exception that occur in our application.

We use **@ControllerAdvice** annotation to handle the exceptions globally across all controllers.

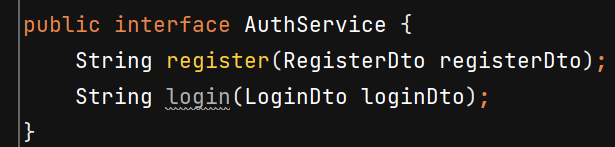
**@ExceptionHandler** used on method that handle specific exception.

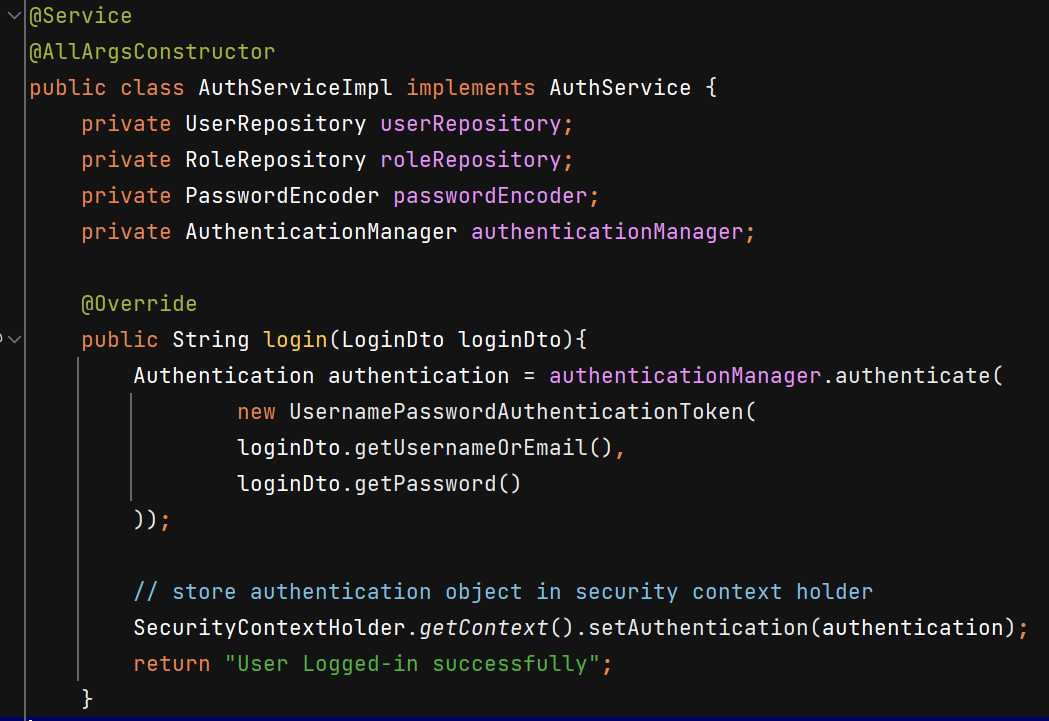


**Build Login REST API**

****

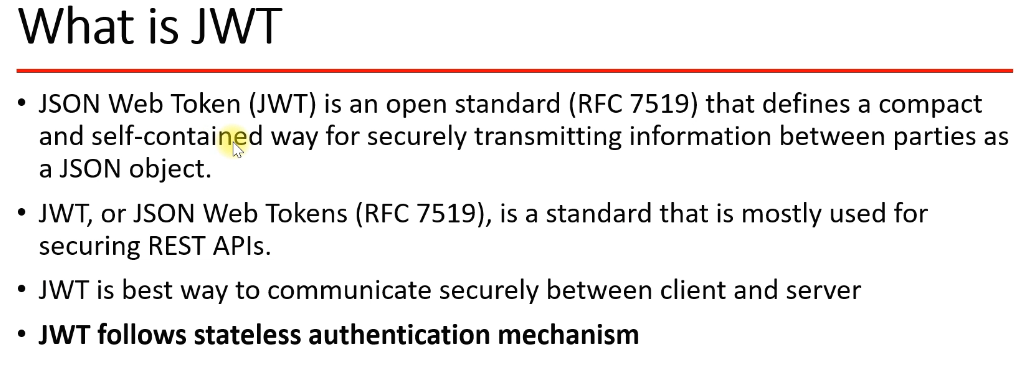


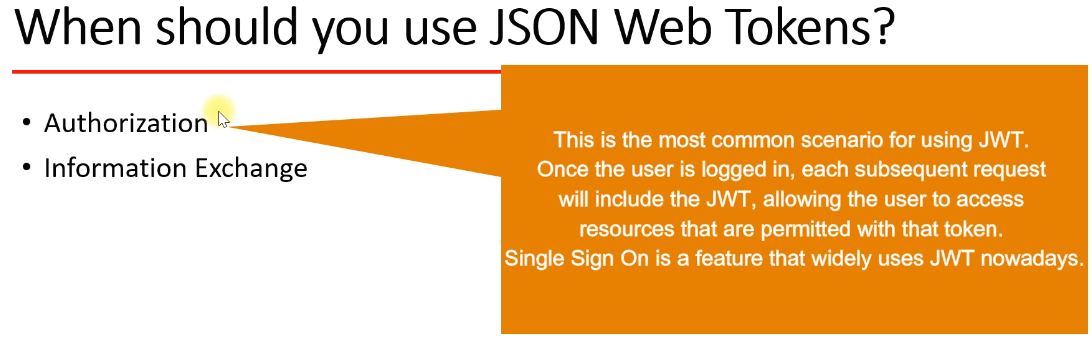


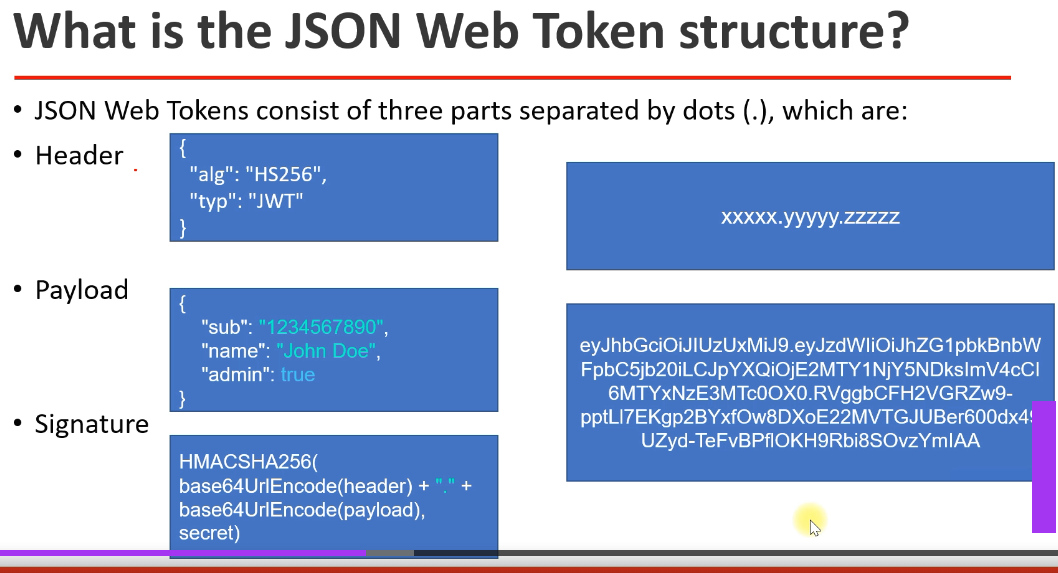




**JWT (JSON Web Token)**

****

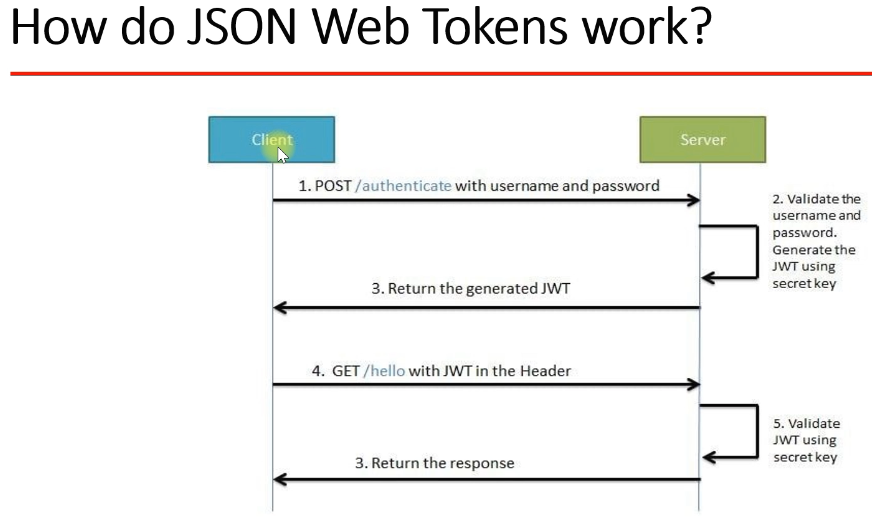
****

****

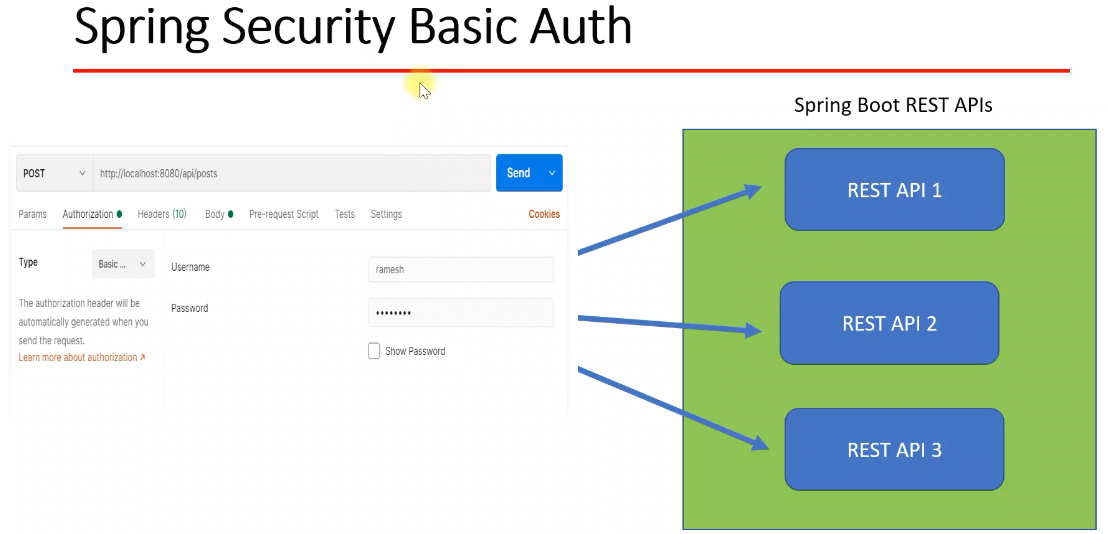
Header consists of alogorithm being used and type of token.

Payload contains the claims. It means here the statement about an entity or a user and the additional data.

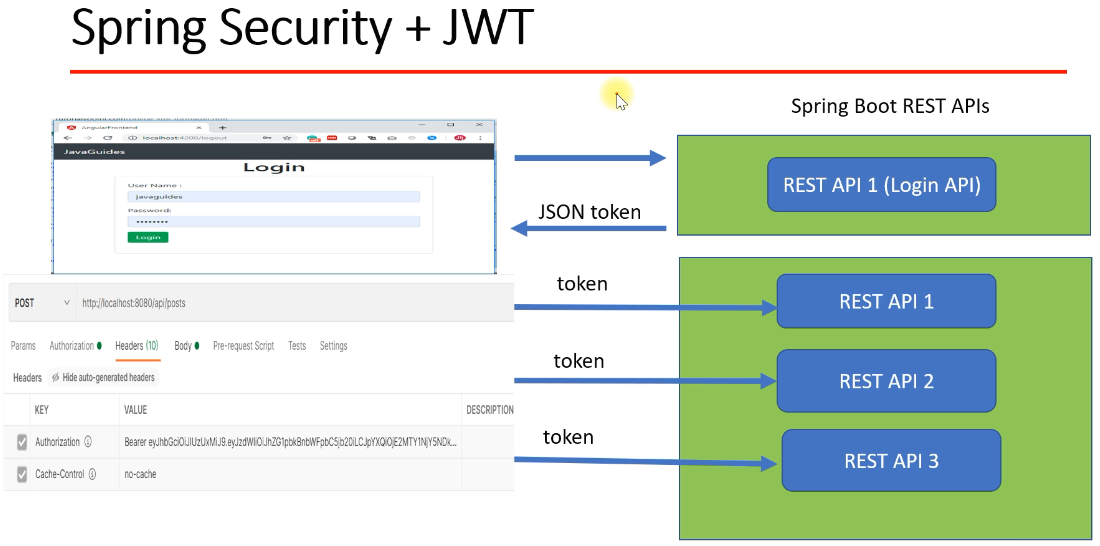
To create a signature, we have to take the encoded header and encoded payload and a secret key

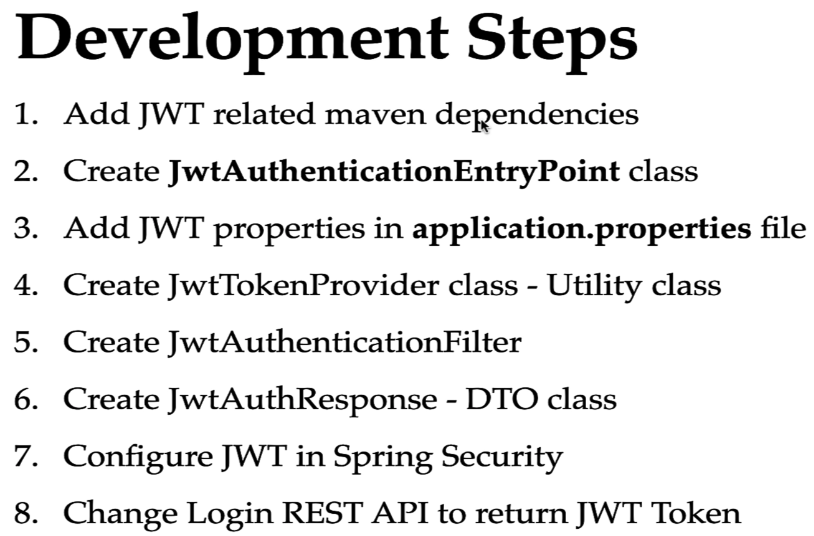
****

Disadvantage of Basic Auth - we have to pass hard coded username and passoword in header of each and every request.

****

How JWT work in spring security – client will call the login API and it will pass username and password. Server will process the request and if username & password are valid, then server will basically generate a JWT token and it will send as a part of response of login API to the client. So once client get JWT token from the server, then client should pass this JWT token in a header of each and every subsequent request.

****

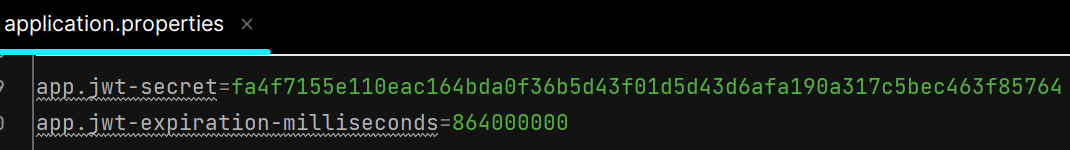
****

Add 3 maven dependencies – jjwt-api, jackson and impl in pom.xml file.

Commence() method is called whenever an exception is thrown due to an unauthorized user trying to access a resource that require an authentication.

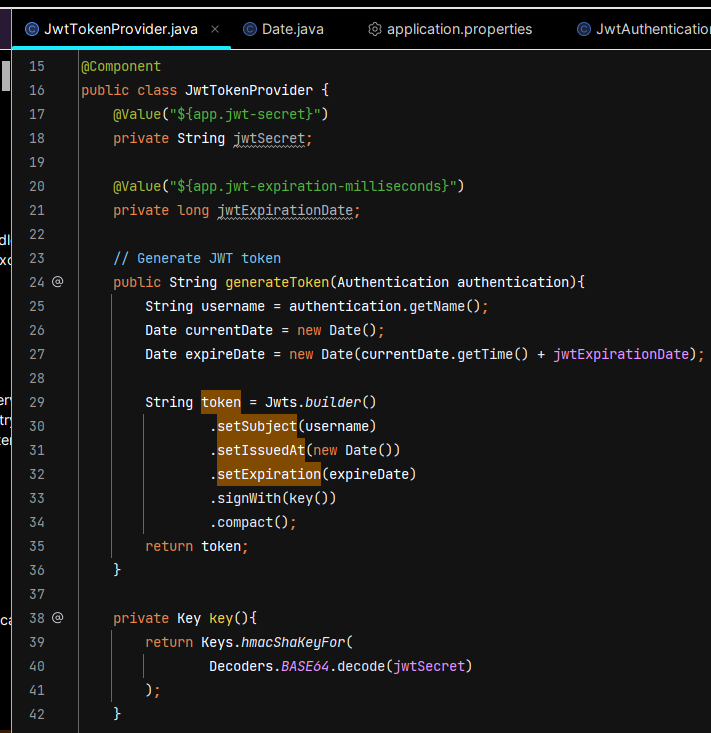
We saved secret key using SHA 256 algorithm.

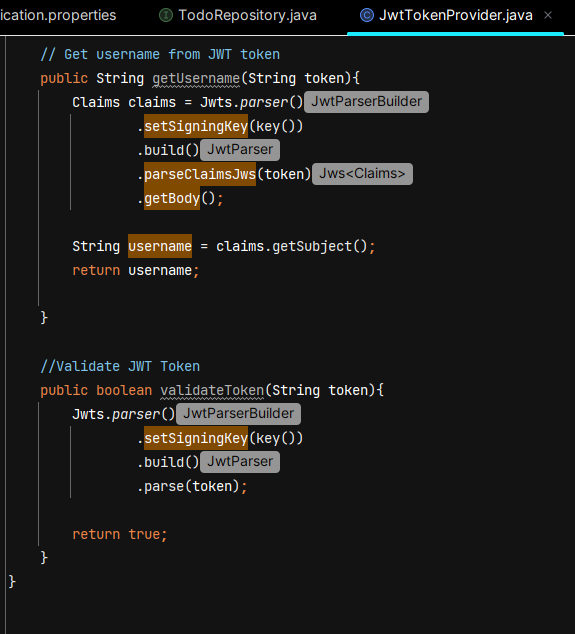




**Creating JwtTokenProvider Utility class**

@Value annotation is used to inject value into field, method parameter or constructor argument. We can give any literal value or inject properties from property file.





**Creating JwtAuthenticationFilter**

OncePerRequestFilter is a base class provided to ensure that a filter is executed only once per request.

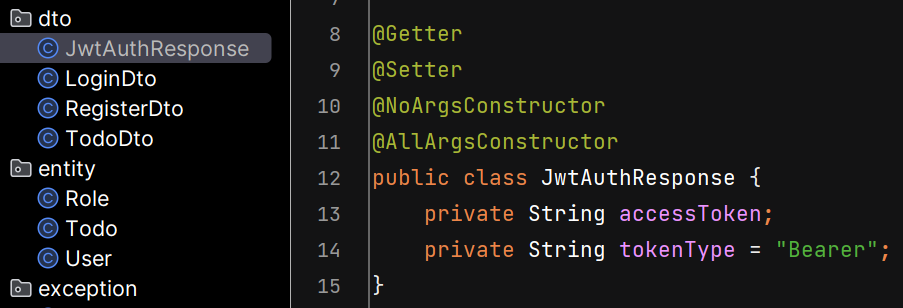
In request header, we’ll get a key as Authorization and value as “Bearer <TOKEN\_VALUE>”

JwtAuthenticationFilter will execute before executing spring security filters. It will validate the JWT token and provides user details to Spring security for Authentication.

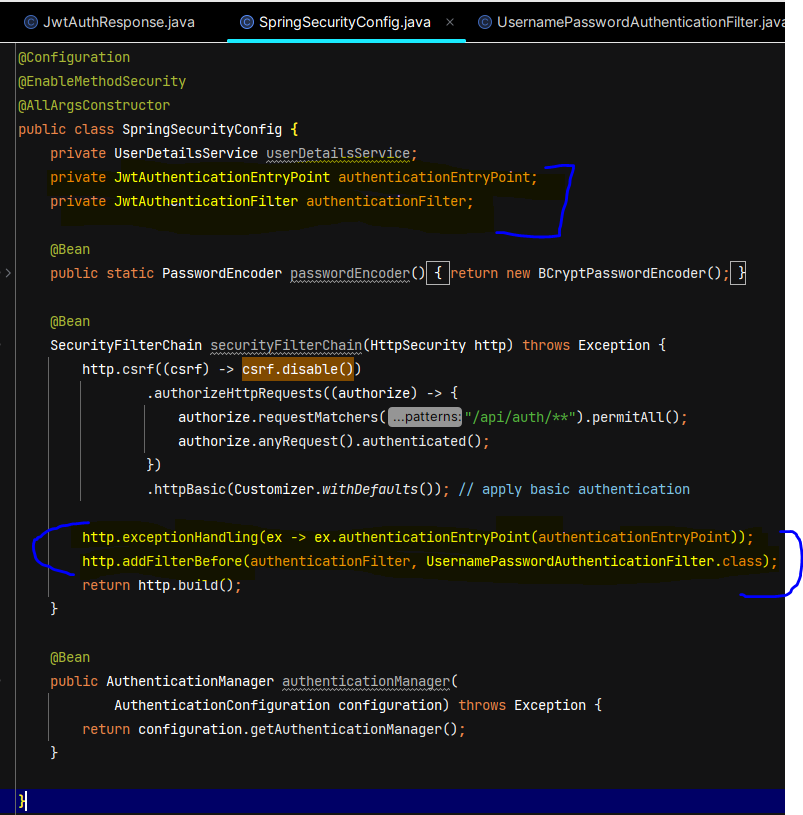




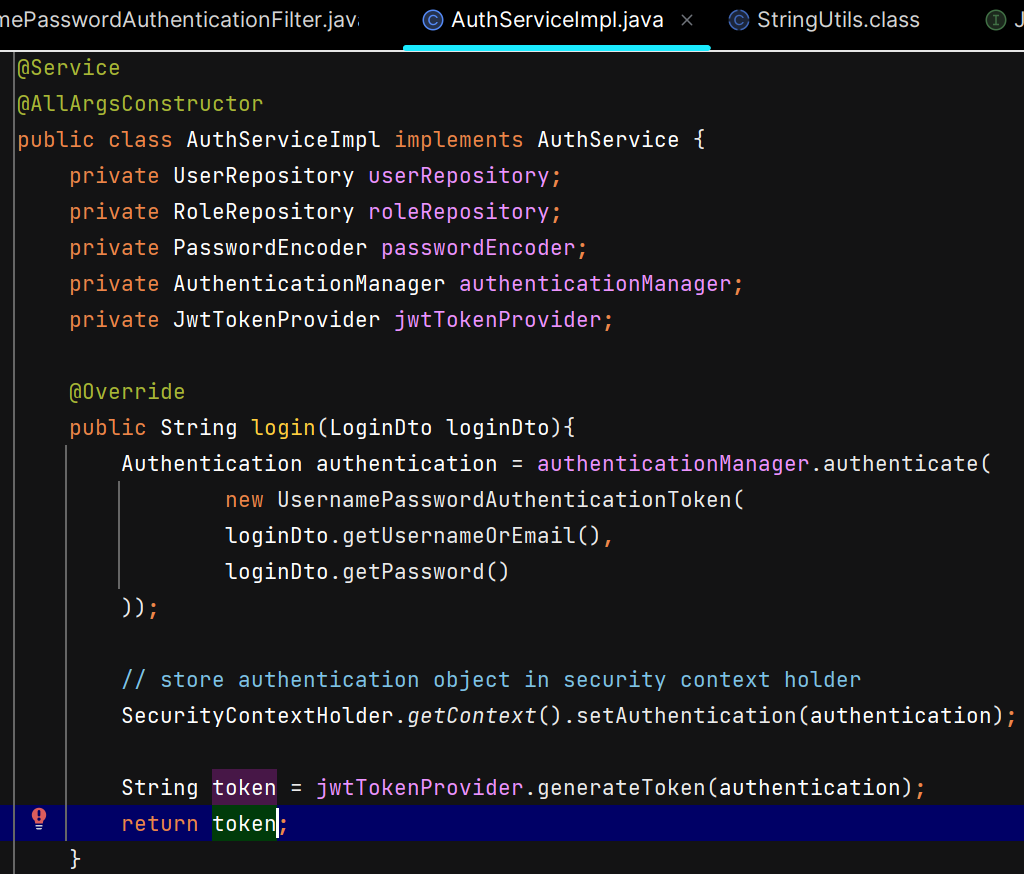
**Configure JWT in Spring Security**



Whenever unauthenticated user tried to access the resource, then spring security throws authentication exception and JwtAuthenticationEntryPoint class will catch that exception and return the error response to the client.

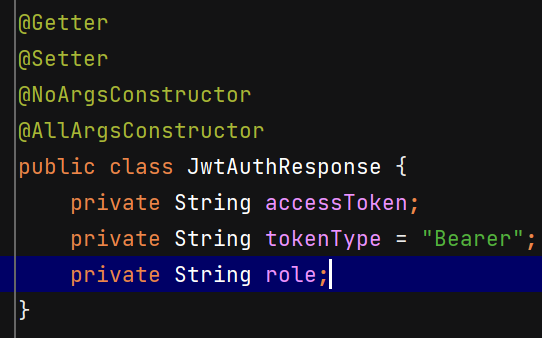


**Change Login REST API to return JWT Token**



**Change login REST API to return Role along with JWT Token**

Add role in JwtAuthResponse -



Change return type of login() method of AuthService to JwtAuthResponse from String –

