———————JWT (JSON WEB TOKEN)———————-

#What is JWT?

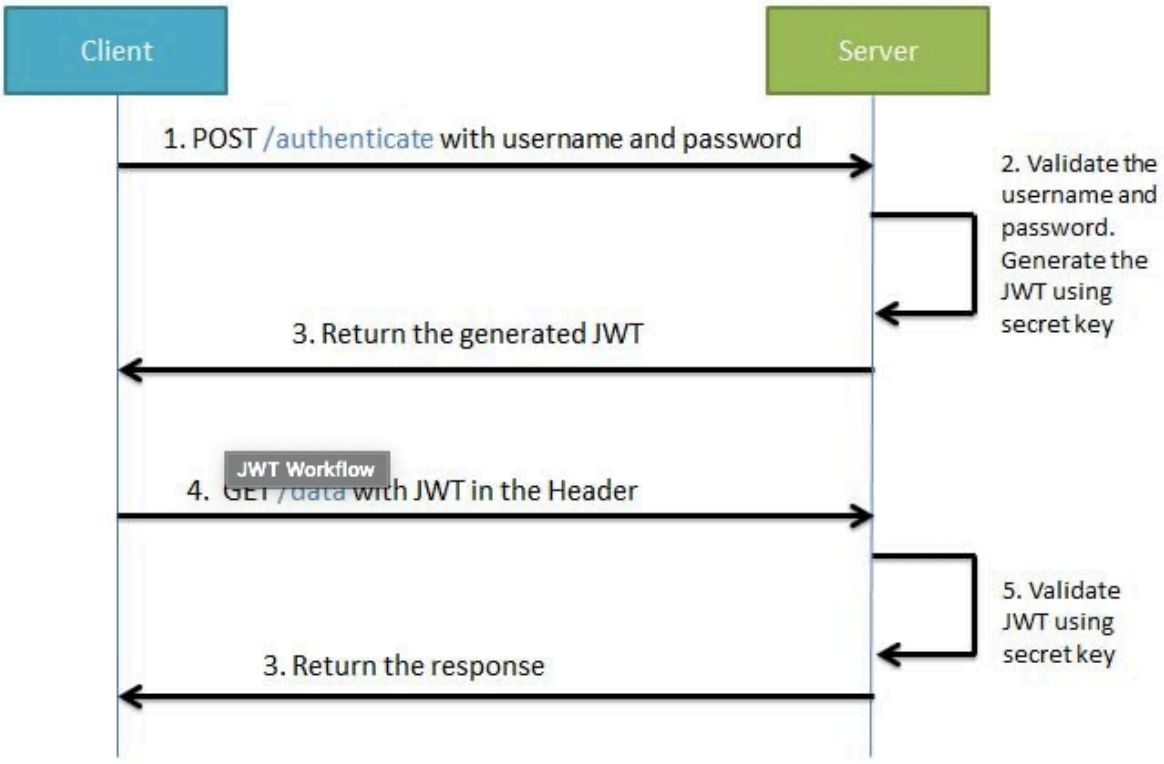
- Open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

"An open standard is a standard that is publicly available and has various rights to use associated with it and may also have various properties of how it was designed. There is no single definition, and interpretations vary with usage."

- The information can be verified and trusted because it is digitally signed.

- The client will need to authenticate with the server using the credentials only once. During this time the server validates the credentials and returns the client a JSON Web Token(JWT). For all future requests the client can authenticate itself to the server using this JSON Web Token(JWT) and so does not need to send the credentials like username and password.

#Workflow of how JWT is used:-



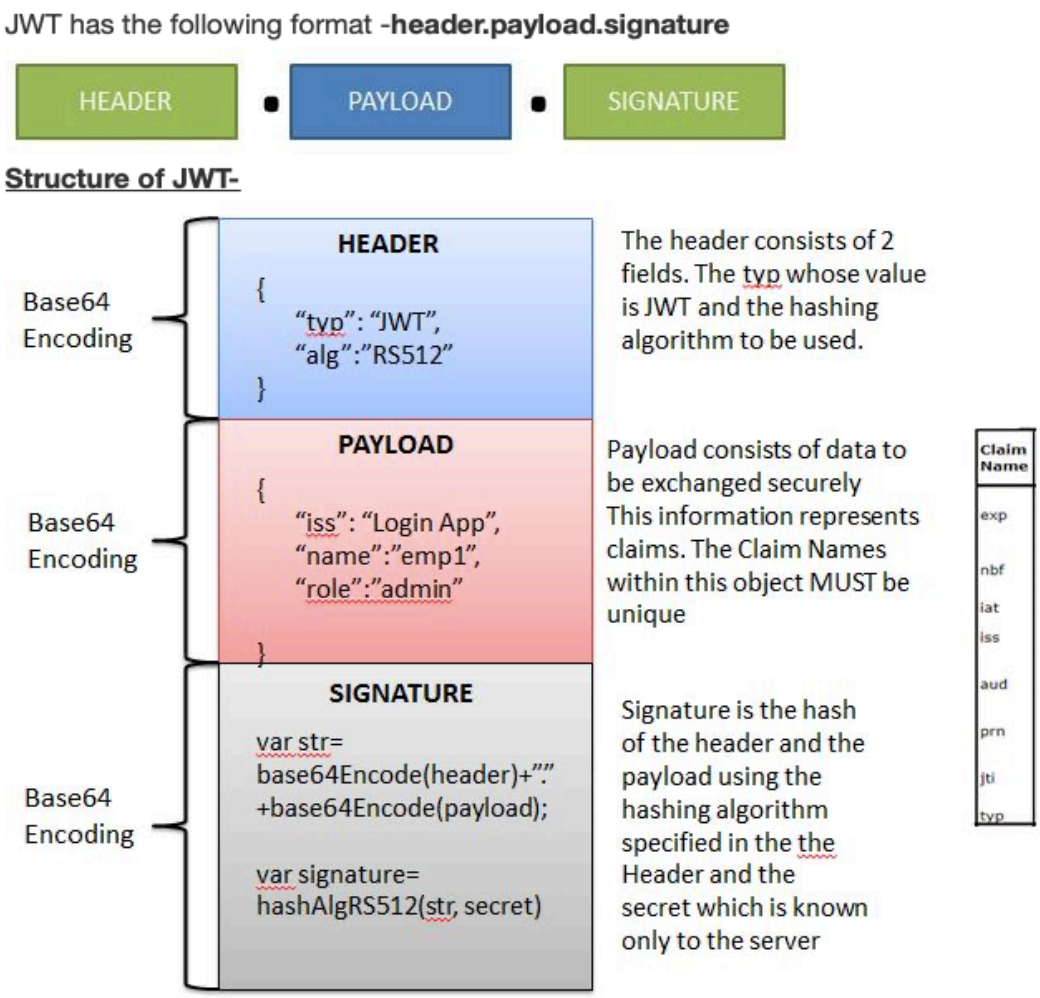
- During the first request the client sends a POST request with username and password.

- Upon successful authentication the server generates the JWT & sends this JWT to the client. This JWT can contain a ***payload*** of data.

- On all subsequent requests the client sends this JWT token in the header. Using this token, the server authenticates the user. So we don't need the client to send the user name and password to the server during each request for authentication, but only once after which the server issues a JWT to the client.

  *"A **JWT payload** can contain things like user ID so that when the client again sends the JWT, you can be sure that it is issued by you, and you can see to whom it was issued."*

#Structure of JWT :-

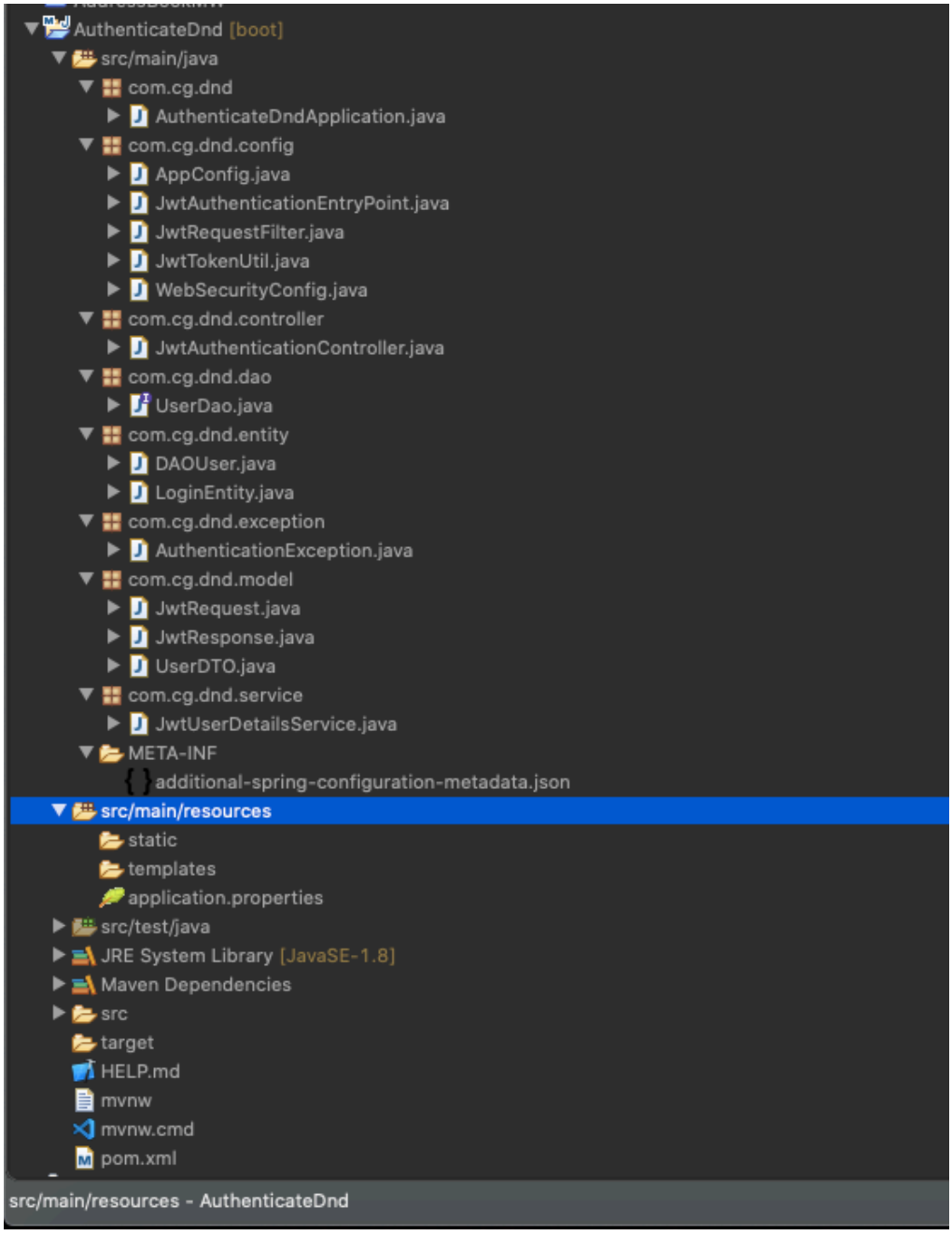


**********NOTE**********

The information in the payload of the JWT is visible to everyone. So we should not pass any sensitive information like passwords in the payload. We can encrypt the payload data if we want to make it more secure. However we can be sure that no one can tamper and change the payload information. If this is done the server will recognise it.

****

#Stage flow of JWT Project in stages :-



•Develop a Spring Boot Application to expose a Simple REST GET API with mapping /data.

•Configure Spring Security for JWT.

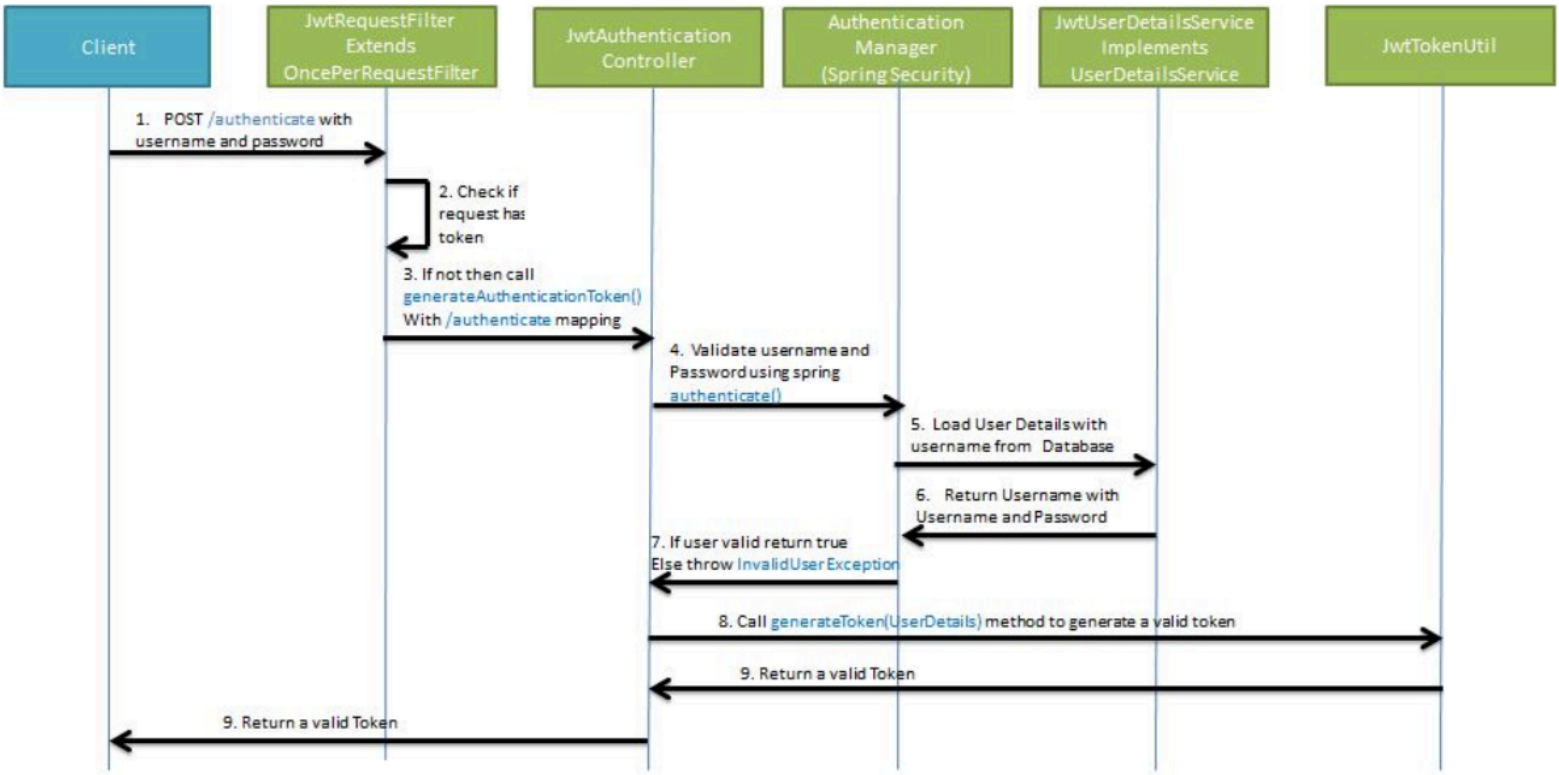
•Expose REST POST API with mapping / authenticate using which User will get a valid JSON Web Token. And then allow the user access to the api / data only if it has a valid token.

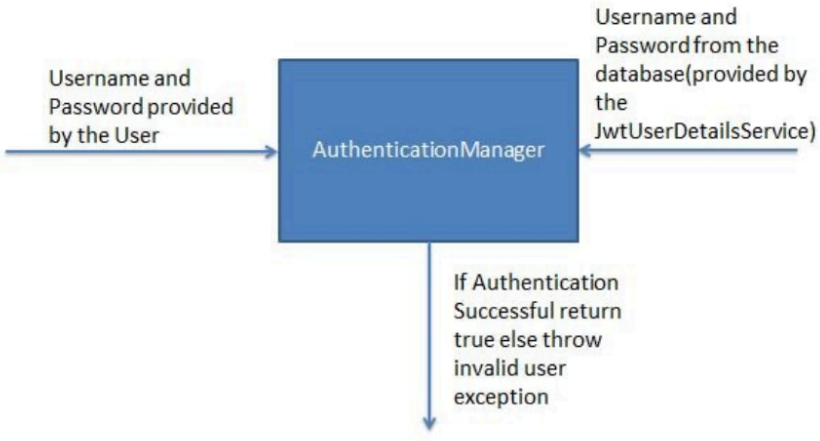
(Project Structure for my Spring JWT project)

#Spring Security and JWT Configuration :-

We will be configuring Spring Security and JWT for performing 2 operations-

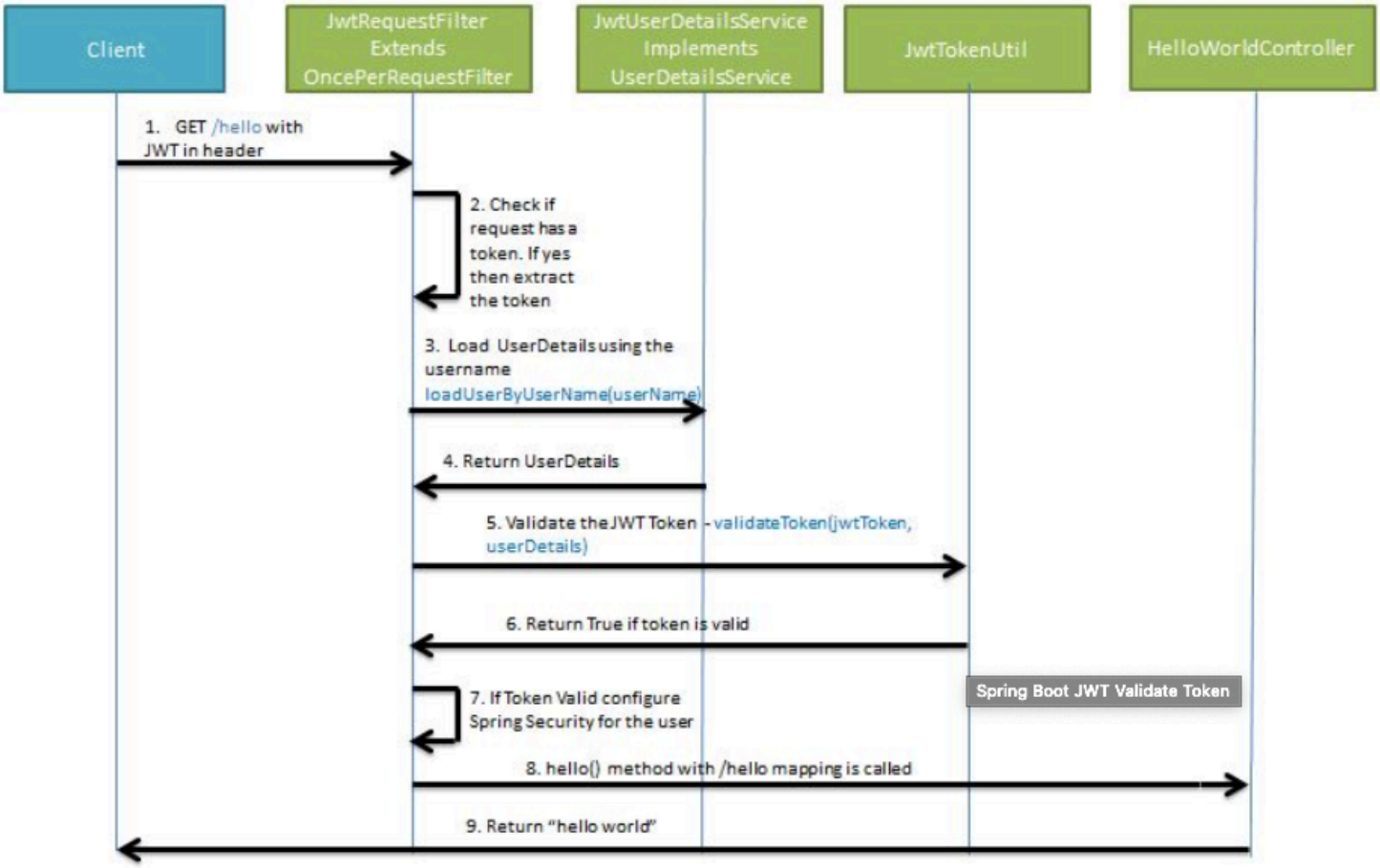
- **Generating JWT** - Expose a POST API with mapping **/authenticate**. On passing correct username and password it will generate a JSON Web Token(JWT)





- **Validating JWT** - If user tries to access GET API with mapping **/login.** It will allow access only if request has a valid JSON Web Token(JWT).

Validating JWT



*****imagine hello=login******

#Step-to-step Procedure :

- Add the Spring Security and JWT dependencies along with standard spring starter dependencies and other dependencies as per required.

- Define the **application.properties**.

  We specify database configurations, port on which it would run and the secret key which we will be using for hashing algorithm. **The secret key is combined with the header and the payload to create a unique hash.** We are only able to verify this hash if we have the secret key.

```
jwt.secret=capgemini@sanskar
server.port=7003
```

```
db MySQL config
spring.datasource.url = jdbc:mysql://localhost:3306/mydb
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username = root
spring.datasource.password = sanskar@123
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
```

- **DAOUser :** (as named in my project AuthenticateDnd)
Creates the Entity class and will be used while performing database operations.

- **UserDTO :** (as named in my project AuthenticateDnd)
Creates model class and is responsible for getting values from user and passing it to the DAO layer for inserting in database.

**********NOTE**********
DTO is an abbreviation for **Data Transfer Object**, so it is used to transfer the data between classes and modules of your application. DTO should only contain private fields for your data, getters, setters and constructors. It is not recommended to add business logic methods to such classes, but it is OK to add some util methods.
It's basically a **value object used for passing structured data between tiers / layers.**

DAO is an abbreviation for **Data Access Object**, so it should encapsulate the logic for retrieving, saving and updating data in your data storage (a database, a file-system, whatever).
It is **responsible for hiding implementation details about how your data is stored and how it is retrieved.**
****

- **UserDao :** (as named in my project AuthenticateDnd)
  An interface that **extends the Spring Framework class CrudRepository and defines the DAO class.**
  The required operation of inserting user details in DB will now be handled.

  "*CrudRepository class is a generics and takes the following two parameters as arguments- What type of Object will this repository be*

working with- In our case DAOUser and Id will be what type of object- Integer(since id defined in the UserDao class is Integer) Thats the only configuration required for the repository class."

**********NOTE**********

JPA: *The Java Persistence API provides* ***a specification for persisting, reading, and managing data from your Java object to relational tables in the database.***

Hibernate:

An ***object-relational mapping solution for Java environments.*** *Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is a Java-based ORM tool that provides a framework for mapping application domain objects to the relational database tables and vice versa.*

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as ORM tool with benefits of loose coupling.

Spring Data:

A ***part of the Spring Framework.*** *The goal of Spring Data repository abstraction is to significantly* ***reduce the amount of boilerplate code required to implement data access layers for various persistence stores.***

Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).

Spring Data offers a solution to `GenericDao` *custom implementations. It can also generate JPA queries on your behalf through method name conventions.*

With Spring Data, you may use Hibernate, Eclipse Link, or any other JPA provider. A very interesting benefit is that you can control transaction boundaries declaratively using the @Transactional annotation.

It is a system to create "automatic" **Data** Access Objects (DAOs) for you at compile time, and uses an **ORM** (like Hibernate) in these DAOs. **Spring Data JPA** will then create a real repository class at compile-time that you can use to select, insert, update, and delete your objects.

"JPA is a specification, Hibernate is a JPA provider or implementation & Spring Data JPA is a JPA Data Access Abstraction."

****

- **JwtTokenUtil :**

Responsible for performing JWT operations like **creation and validation of tokens**.It makes use of the **io.jsonwebtoken.Jwts** for achieving this.

- **JWTUserDetailsService :**

(Here, we autowired the UserDao bean and the BcryptEncoder bean. Also defined the saveUser function for inserting user details)

* Implements the **Spring Security UserDetailsService interface**.
* It overrides the **loadUserByUsername** for fetching user details from the database using the username.
* The **Spring Security Authentication Manager** calls this method for getting the user details from the database when authenticating the user details provided by the user.
* We have added the DAO implementation for fetching User Details from the Database.
* Also the password for a user is stored in encrypted format using **BCrypt**.

***This class acts like a provider for the user; meaning it loads the user from the database (or any data source). It doesn't do authentication. It just loads the user given his username.

- **JwtRequest :**

This class is required for storing the username and password we receive from the client.

- **JwtResponse :**

This class is required for creating a response containing the JWT to be returned to the user.

- **JwtRequestFilter :**

The JwtRequestFilter **extends the Spring Web Filter OncePerRequestFilter class**. For any incoming request this Filter class gets executed. It checks if the request has a valid JWT token. If it has a valid JWT Token then it sets the Authentication in the context, to specify that the current user is authenticated.

`OncePerRequestFilter` guarantees a single execution per request (since you can have a filter on the filter chain more than once).

- **JwtAuthenticationEntryPoint :**

This class will **extend Spring's AuthenticationEntryPoint class** and **override its method commence**. It **rejects every unauthenticated request and send error code 401**.

- WebSecurityConfig :
This class extends the WebSecurityConfigurerAdapter & is a convenience class that **allows customization to both WebSecurity and HttpSecurity.**

- **JwtAuthenticationController :**

* Exposes a POST API /authenticate using the JwtAuthenticationController.
* The POST API gets username and password in the body- Using Spring Authentication Manager we authenticate the username and password.
* If the credentials are valid, a JWT token is created using the JWTTokenUtil and provided to the client.

********Controller is added normally for adding a POST request for adding any user details to database.