Spring Security

Application Security

Application security refers to the protective measures built into the software applications to protect ***data***, ***resources*** and ***functionalities*** from *unauthorized access*, *malicious attacks* and other threats.

Organizations building digital products/services where the users will be accessing these services via HTTP protocol ex: Flipkart, Netflix, Youtube, Gmail

These products/services contain two critical resources

1. Customer data
2. Application functionality

The above can be further classified into unsecure and secure features

Insecure features/functionalities

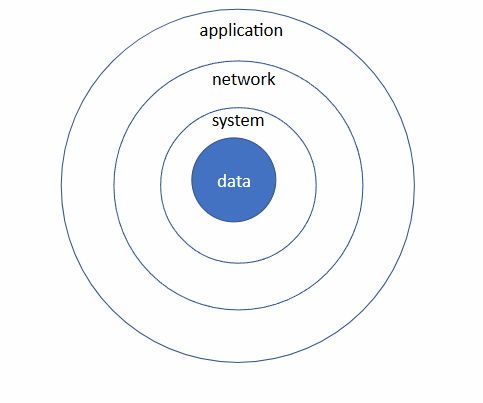
These features do not need security and are often intended for unauthorized access

* User registration
* Login/Logout
* About-us/portfolio/home page
* Contact-us

The features that needs security

* User profiles
* User preferences
* User data/resources

Different levels of security



1. Data Security:
   1. It is at the core of security layer
   2. Data should be stored in encrypted format
   3. Data should be store using a *one-way hash function*
   4. Sensitive information should be stored in the log files
2. System Security
   1. System/server where the data resides should also be secured
   2. *Rule of minimum privilege* should be followed to provide access to the server
   3. All activities should be logged (AWS - Cloudtrail)
   4. Root account should be disabled
   5. All the OS updates should be done periodically to ensure security vulnerabilities

are patched

1. Network Security
   1. Ensure that the firewall rules are configured correctly
   2. Only the necessary ports should be open and from a specific subnet
   3. All the backend non customer facing applications should be provisioned inside private subnets
   4. Follow HTPS protocol and certificates should be installed.
   5. External facing application, certificates from CA should be installed and for internal communication we can use self-signed certificates.
2. Application Security
   1. Two important things to be considered for application security – *Authentication and Authorization*
   2. Authentication refers to checking if you are the same person whom you claim to be.
   3. Authorization refers to checking if you are having the permissions to carry out the tasks.
   4. In HTTP world, the Authentication failure results in Http Status code – ***401***
   5. In HTTP world, the Authorization failure results in Http Status code – ***403***

Implementing security comes with a cost, at the same time, if there is a vulnerability, the advisories will exploit the vulnerability.

If the application is compromised, then the ***TRUST*** of the organization is at stake.

Application Security

There are broadly two ways in taking the approach to implement security:

Customer Centric application (Banking domain, Financial, Healthcare)

1. The applications should be less user friendly
2. Strict measures for login activity

Social Centric applications (Facebook, Google, Twitter)

1. The applications should be easier for the users to use the application
2. Force the users to authenticate for critical tasks (profile updating/ deletion)

Security should be implemented using *Defense in Depth* technique. i.e, even if one of the layers is compromised the next layer should be even more strong. And in case of worst case scenario where all the layers get compromised the data should be meaningless to the attacker.

Security domain

Principal:

A Principal is the entity that can be authenticated by a computer security system. It represents

1. An authenticated identity that can perform actions in the system. i.e, a logged in entity
2. Any entity that can be granted access rights and hold security permissions

Examples:

1. Individual users
2. Processes
3. Services/Applications
4. System/Machines
5. API/Service-Accounts
6. Roles/Groups

Anonymous Entity

1. A Non-logged in entity is referred to as Anonymous. An entity without identity. An anonymous entity after the login process becomes a Principal.

Authentication

Different types of Authentication

1. Username/password
2. Multi-Factor
3. Biometric
4. Single-Sign-On
5. OpenID/OAuth 2.0
6. SAML
7. Certificates
8. Token bases
9. API keys
10. Active directory
11. Hardware token
12. OTP/TOTP
13. Grid
14. Passkeys
15. Captcha

Authorization

1. Role-based Authorization
2. Attribute based Authorization
3. OAuth 2.0
4. Path based Authorization
5. Method level Authorization
6. Policy based Authorization
7. Claim based Authorization
8. Group based Authtorization

Frameworks to implement Application Security

1. JAAS API - Java API for implementing Application Security
   1. Java Authentication and Authorization Service
2. Spring Security
   1. Wide community adoption and support
   2. Natively integrates with Spring ecosystem
   3. Provides extensive supports for different types of authentication providers
   4. Very well documented and maintained
3. Apache Shiro

Spring Security

Spring Security is a comprehensive Application security framework built into Spring Ecosystem.

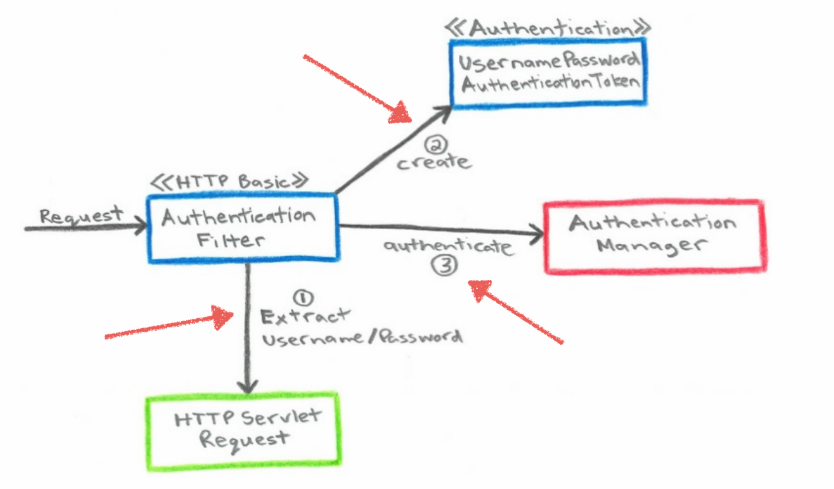
1. It integrates with many different Authentication providers, provides comprehensive solution to implement application security and can be extended.
2. All the heavy lifting is taken care by the Spring Security framework behind the scenes.
3. It lets developers write the application logic and takes care of seamlessly integrating the security of the applications.
4. It also handles application vulnerabilities like CSRF, CORS with less efforts from the developers.
5. It is implemented based on Filter pattern. It intercepts the requests, performs Authentication and Authorization before hitting the endpoints.
6. The application does not even know that the requests are being intercepted.

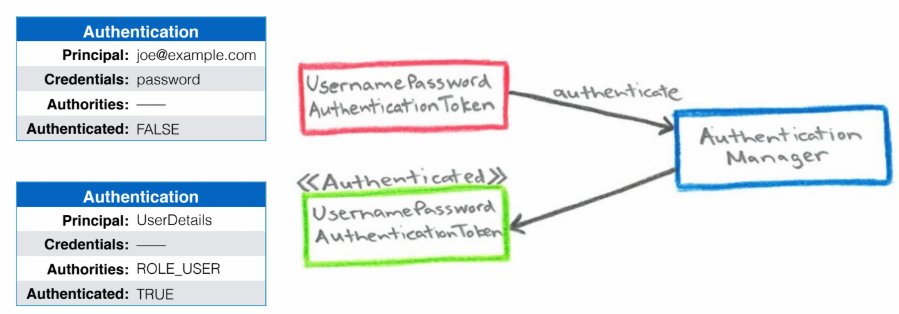
Authentication flow

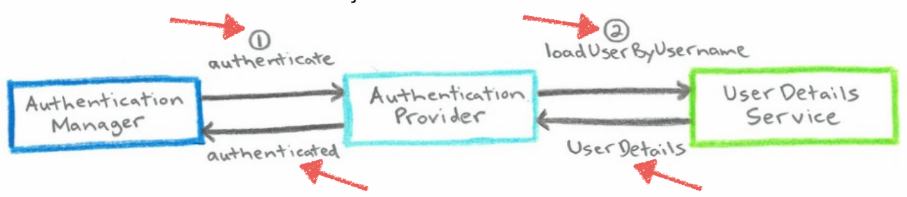
1. Anonymous user login to the application
2. Passes his credentials (*username and password*) using POST request
3. *UsernamePasswordAuthenticationFilter* intercepts the request
4. Extracts the username and password and create a *UsernamePasswordAuthenticationToken* object
5. The Filter then passes the authentication token object to *AuthenicationManager’s authenticate* method
6. The default implementation of AuthenticationManager is *ProviderManager* which provides the implementation for the *authenticate* method
7. The *authenticate* method takes the *Authentication* object and sends back *Authentication* object in response.
8. The ProviderManager calls the authenticate method on the list of AuthenticationProviders. From the list, the *AuthenticationProvider* which supports the type of authentication will be used for authentication.
9. The AuthenticationProvider implementation will implement the authentication and will be specific to the type of Authentication i.e, *DAOAuthenticationProvider* can authenticate against DBs, *OIDCAuthenicationProvider* can authenticate oidc login.

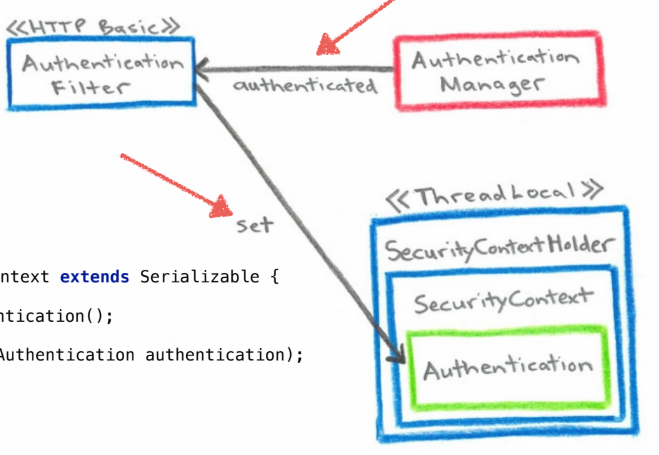
*Specific to Database authentication*

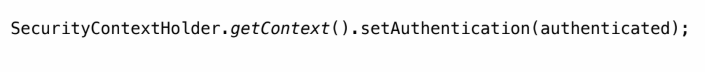
1. In case of Database authentication, we use *DAOAuthenticationProvider* which sends the *username* to *UserDetailsService loadUserByUsername* method.
2. We have to provide custom implementation for the *UserDetailsService loadUserByUsername* method and fetch the User from the DB corresponding to the *username* provided.
3. If the username is not present, we should throw an exception (*UsernameNotFoundException*) else, need to map the domain user to *UserDetails* object using the *adaptor design pattern*.
4. The *DAOAuthenticationProvider* verifies the user, matches the passwords, erases the credentials and return back the *UserDetails* object to the *AuthenticationManager.*
5. The *UsernamePasswordAuthenticationFilter* lastly stores the authentication object inside a *SecurityContext* which is ThreadLocal.
6. The security context will be available within any method where the requests flow.

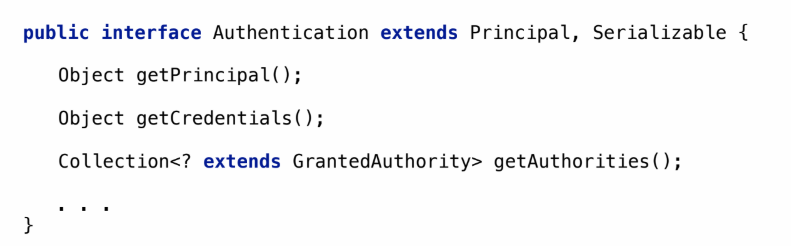
**

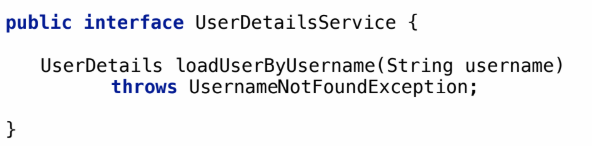
**

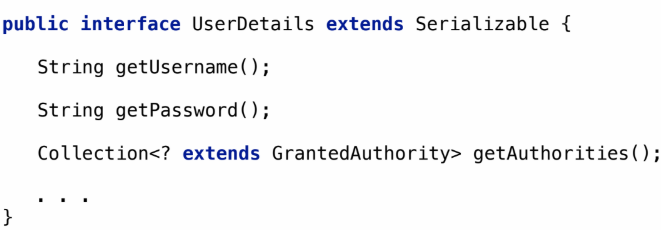
**

**

**

**

**

**

Lab: Setting up a Spring Boot project

Required starter dependencies

* *spring-boot-starter-web*
* *spring-boot-starter-data-jpa*
* *H2*
* *spring-boot-starter-validation*

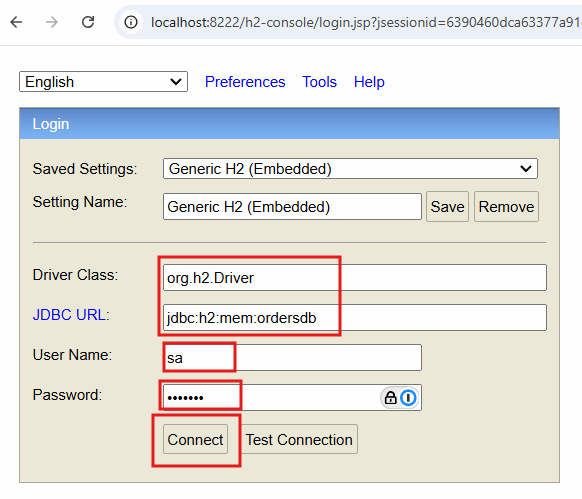
Steps to clone the repository

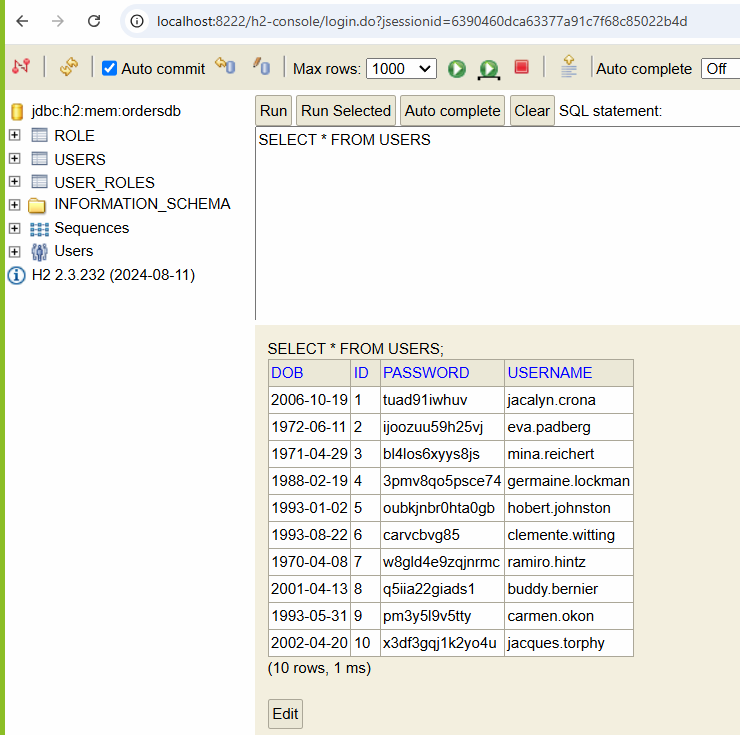
*git clone* [*https://github.com/pradeepkl/orders-application.git*](https://github.com/pradeepkl/orders-application.git)

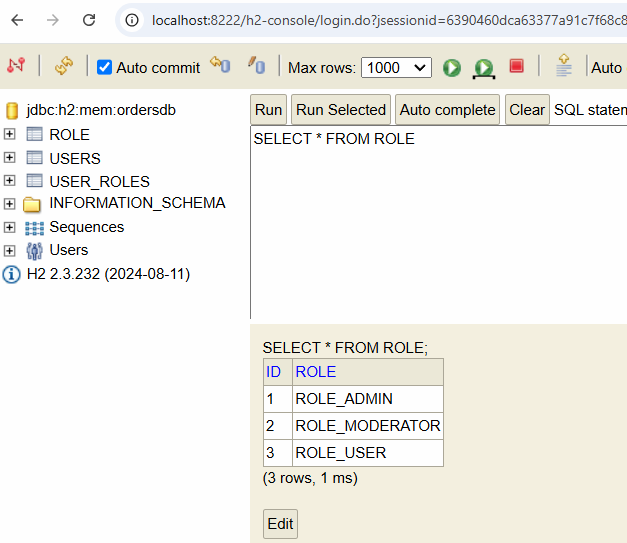
Import the project as maven project inside STS/Eclipse IDE

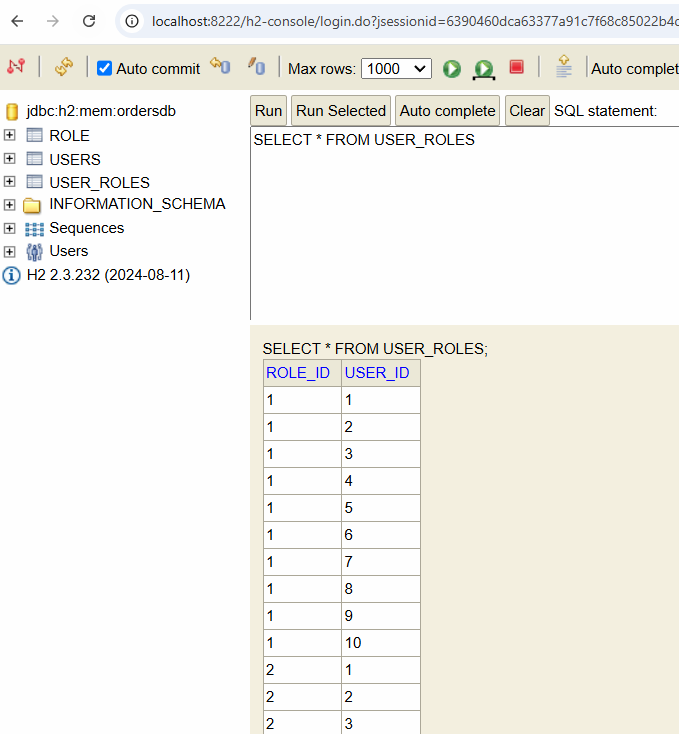
Open project in case of IntelliJ/VSCode editor

Run the application and visit the url in the browser: <http://localhost:8222/h2-console>









Day-2

Building a REST API: CRUD implemenations

Domain model: Order and LineItems

Relationships: One to Many (Bidirectional)

Endpoints for OrderRestController

* *GET - /api/orders*
* *GET - /api/orders/<id>*
* *POST - /api/orders*
* *PUT - /api/orders/<id>*
* *DELETE - /api/orders/<id>*

Order Payload:

{

    "customerName": "Lowell Moore",

    "totalAmount": 130.9,

    "orderDate": "1982-02-03",

    "lineItems": [

        {

            "itemName": "Rustic Granite Bag",

            "price": 18.18,

            "quantity": 2

        },

        {

            "itemName": "Small Copper Gloves",

            "price": 94.54,

            "quantity": 1

        }

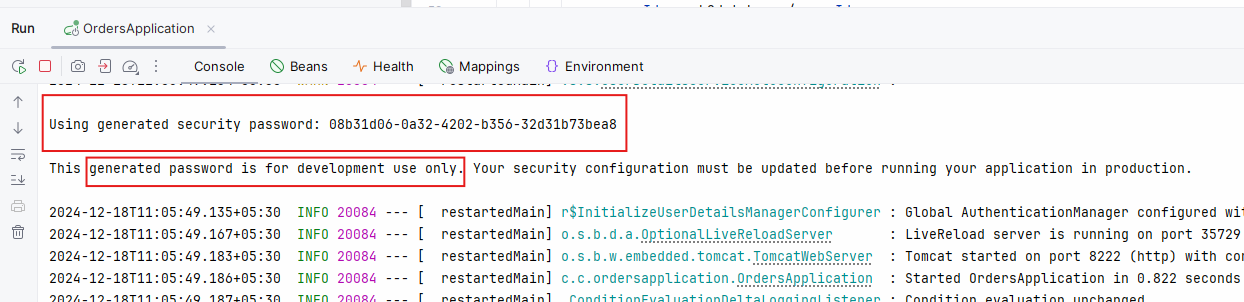
    ]

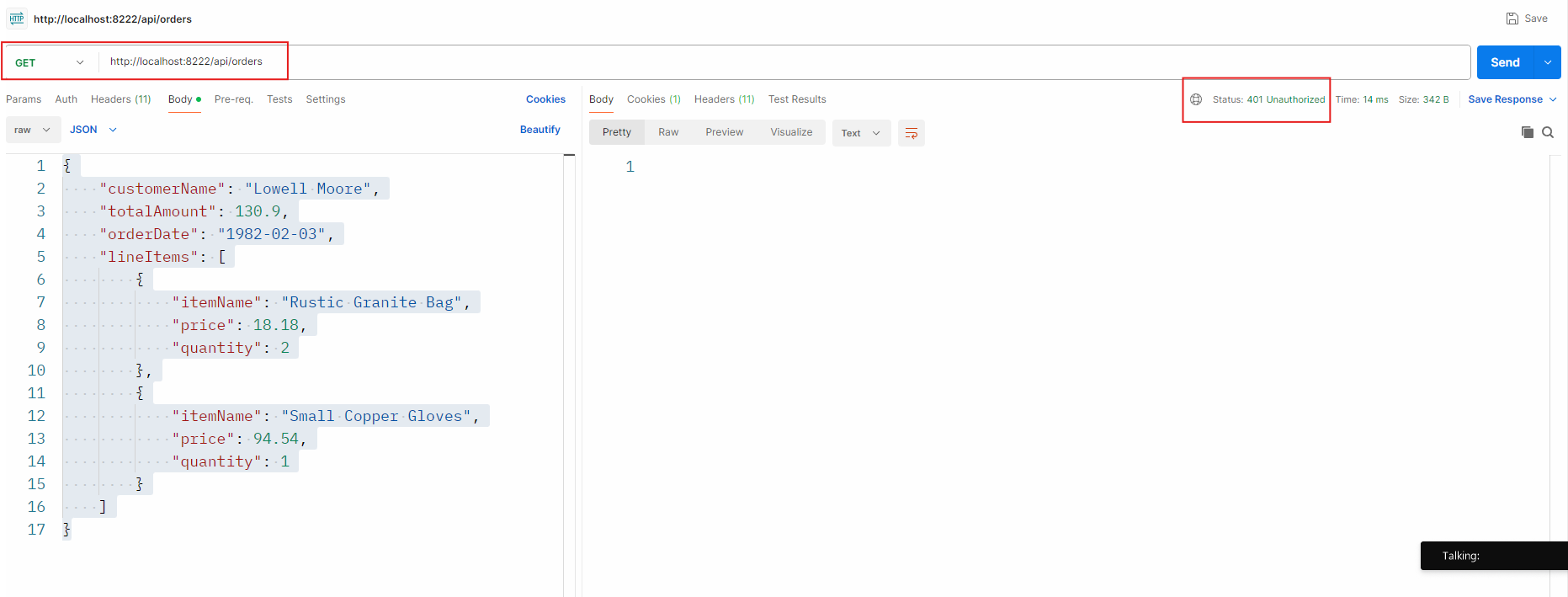
}

Securing the REST API’s using Spring Security (6.4.1)

pom.xml

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>





Spring Security Configuration

@Configuration  
@EnableMethodSecurity  
@RequiredArgsConstructor  
public class SecurityConfiguration {  
  
 private final UserDetailsService userDetailsService;  
 private final PasswordEncoder passwordEncoder;

SecurityFilterChain

@Bean  
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 *//use the builder patern to configure the HttpSecurity object* http.csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(  
 authz -> authz  
 .requestMatchers("/h2-console/\*\*", "/login\*\*", "/logout", "/about-us").permitAll()  
 .requestMatchers(HttpMethod.*GET*, "/api/orders/\*\*").hasAnyRole("USER", "ADMIN", "MODERATOR")  
 .requestMatchers(HttpMethod.*POST*, "/api/orders/\*\*").hasAnyRole("ADMIN", "MODERATOR")  
 .requestMatchers(HttpMethod.*DELETE*, "/api/orders/\*\*").hasRole("SUPER\_ADMIN")  
 .anyRequest().fullyAuthenticated())  
 .headers( header -> header.frameOptions(frame -> frame.disable())) *// for h2-console* .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.*STATELESS*))  
 .httpBasic(httpBasic -> {}) *//use basic authentication.i.e username:password base64 encoded* .formLogin(form -> form *// form based authentication* .loginPage("/login")  
 .loginProcessingUrl("/login")  
 .defaultSuccessUrl("/home", true)  
 .failureUrl("/login?error=true")  
 .permitAll());  
 return http.build();  
}

DAOAuthenticationProvider

@Bean  
public DaoAuthenticationProvider authenticationProvider(){  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService);  
 provider.setPasswordEncoder(passwordEncoder);  
 return provider;  
}

PasswordEncoder

@Bean  
public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
}

Requirement for building User management

* Onboarding user and managing users is an integral part of secure application development
* There are challenges when building a comprehensive user management module

Challenges in building a user management module

1. User Registration/Signup workflow
2. Secure Storage of Password
3. Password management (complexity, strength, policy)
4. Password rotation
5. Password reset
6. Account lock/unlock

Spring Security