API Security using Spring Security Framework

standard chartered

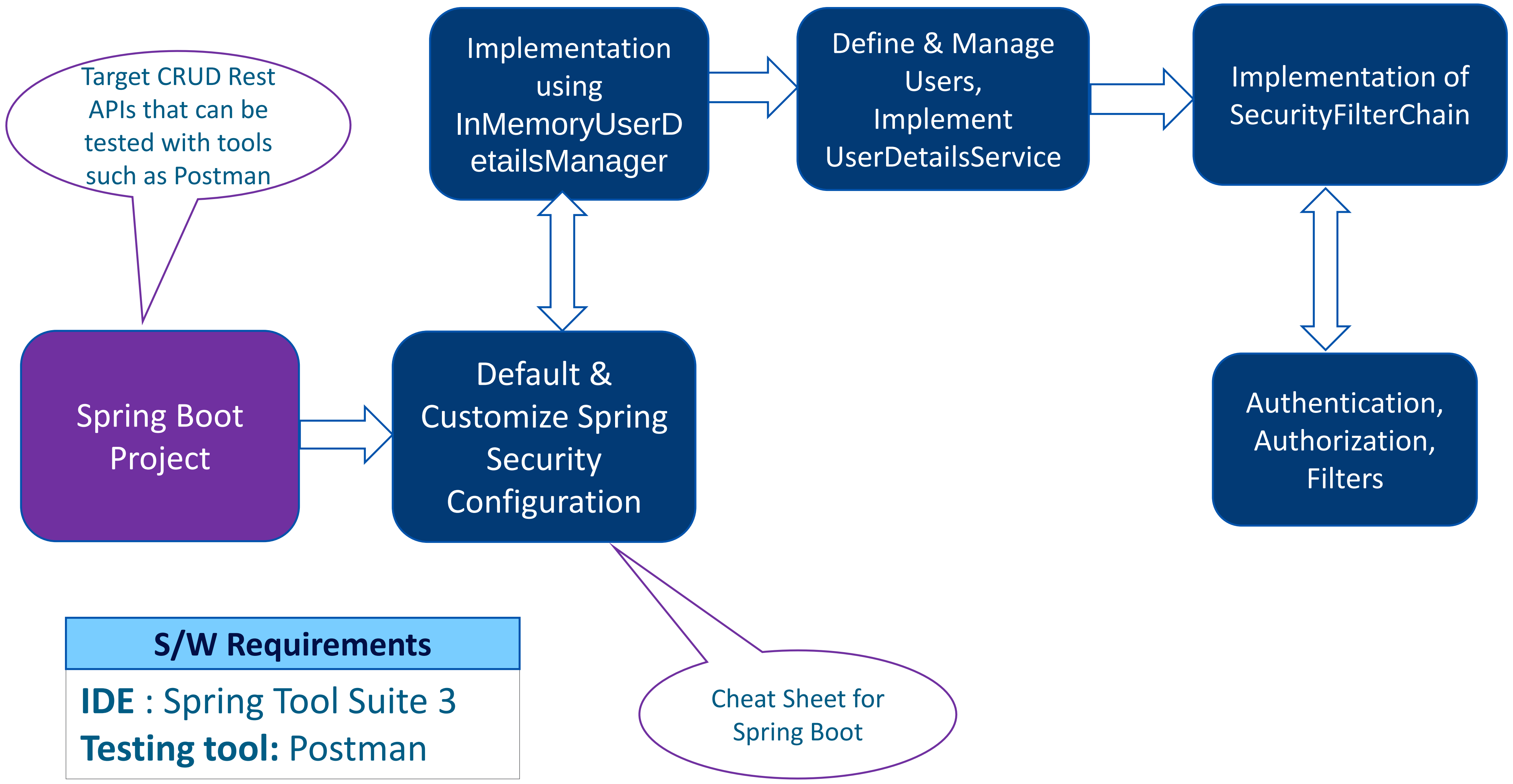
axess academy

INTERNAL

Content

axess academy

Module Design


Target CRUD Rest APIs that can be tested with tools such as Postman
Implementation using InMemoryUserDetailsManager
Define & Manage Users, Implement UserDetailsService
Implementation of SecurityFilterChain
Spring Boot Project
Default & Customize Spring Security Configuration
Authentication, Authorization, Filters
Cheat Sheet for Spring Boot

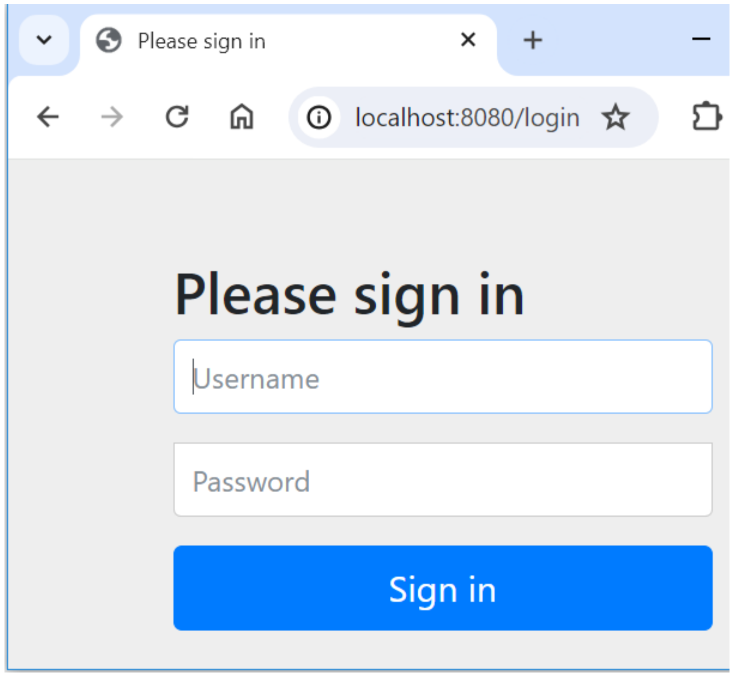
S/W Requirements
IDE : Spring Tool Suite 3
Testing tool: Postman

Use Case 1 on Spring Security

axess academy

Use Case 1

- Check basic authentication and authorization using InMemoryUserDetailsManager

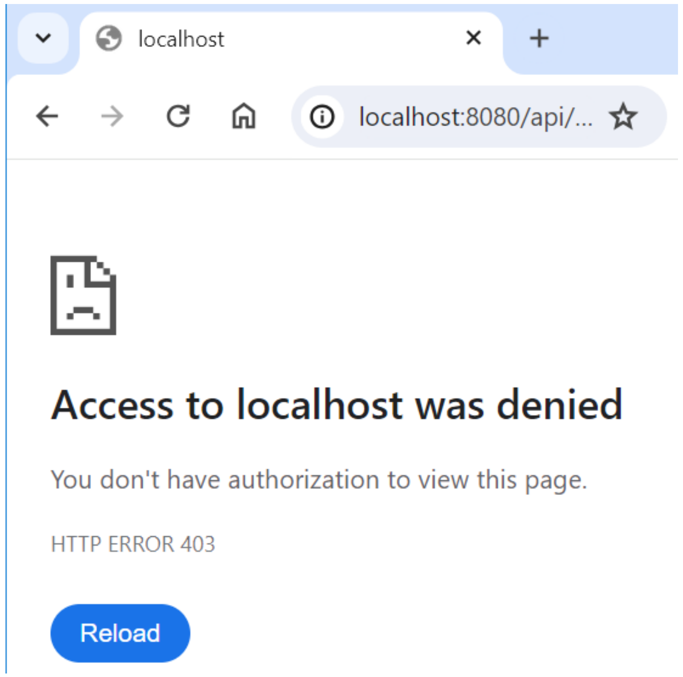


Please sign in

Username

Password

Sign in



Access to localhost was denied

You don't have authorization to view this page.

HTTP ERROR 403

Reload

Spring Security

- Powerful and Customizable authentication and authorization framework
- De-facto standard for securing Spring-based applications

Terminologies

Authentication	Process of ***verifying the identity of a user***, based on provided credentials
Authorization	Process of ***determining if a user has proper permissions*** to perform a particular action or read data
Principle	Refers to ***currently Authenticated user***
Granted Authority	Refers to ***permission of the authenticated us***er
Role	Refers to ***group of permissions of the authenticated user***

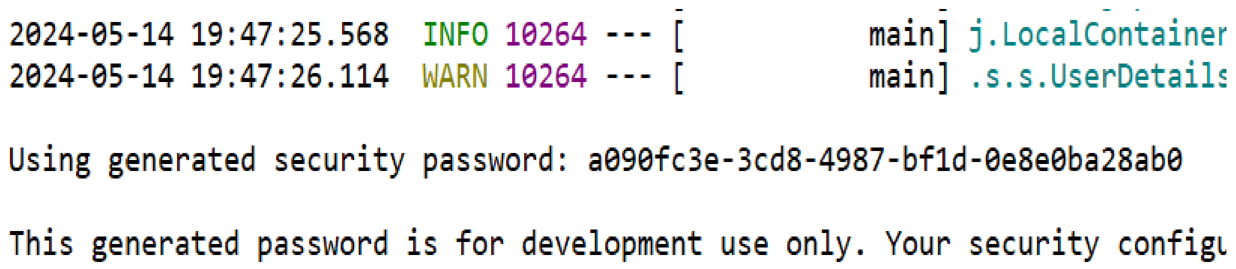
axess academy

Basic security with random password

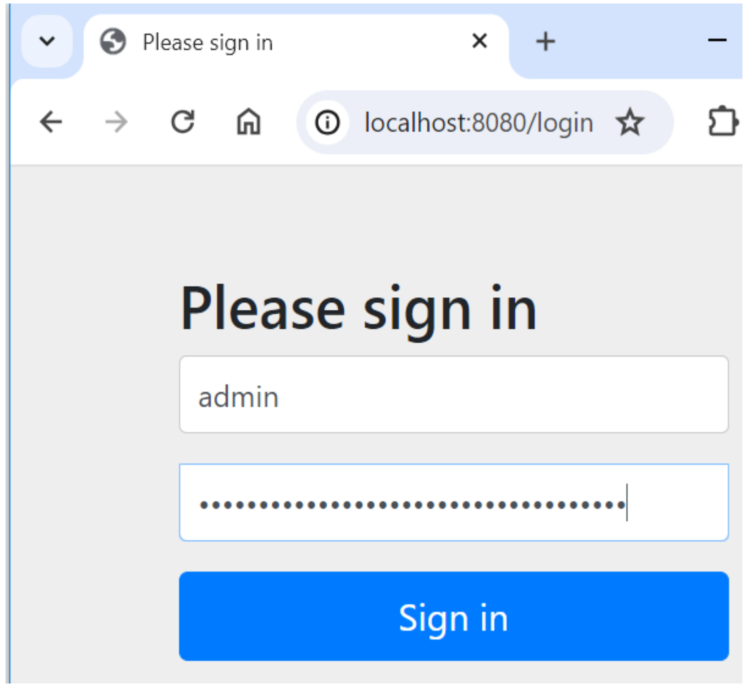
- Basic security is provided to the application by adding the below dependency

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

- Spring Security generates a random password every time when we execute the Spring Application.



```
2024-05-14 19:47:25.568  INFO 10264 --- [           main] j.LocalContainer
2024-05-14 19:47:26.114  WARN 10264 --- [           main] .s.s.UserDetails

Using generated security password: a090fc3e-3cd8-4987-bf1d-0e8e0ba28ab0

This generated password is for development use only. Your security config
```



Please sign in

admin

••••••••••••••••••••••••••••••••••••

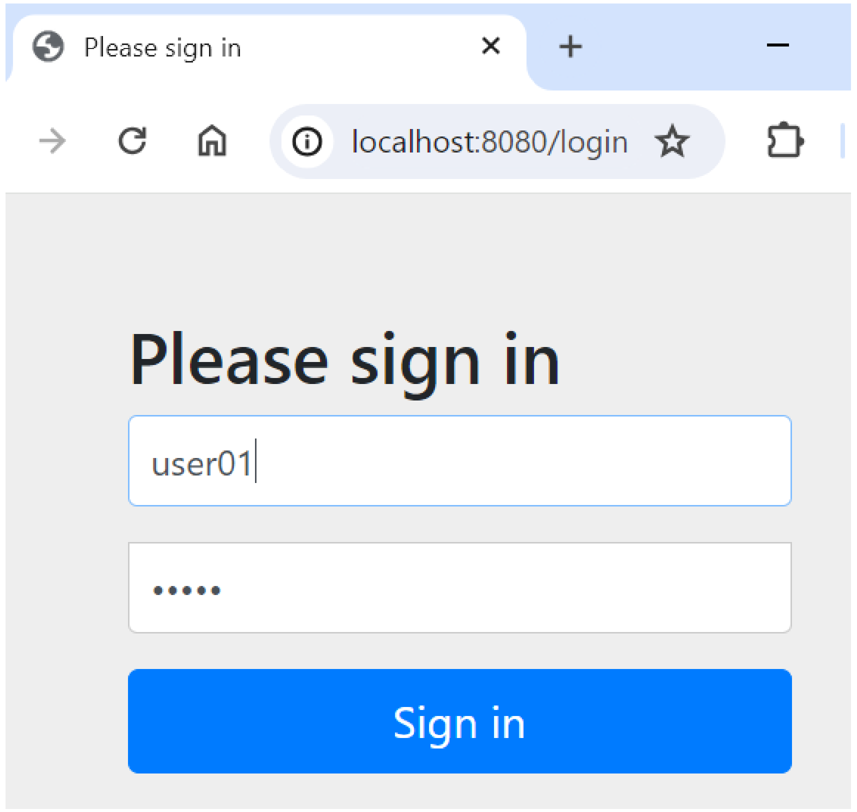
Sign in

axess academy

Adding credentials using application.properties

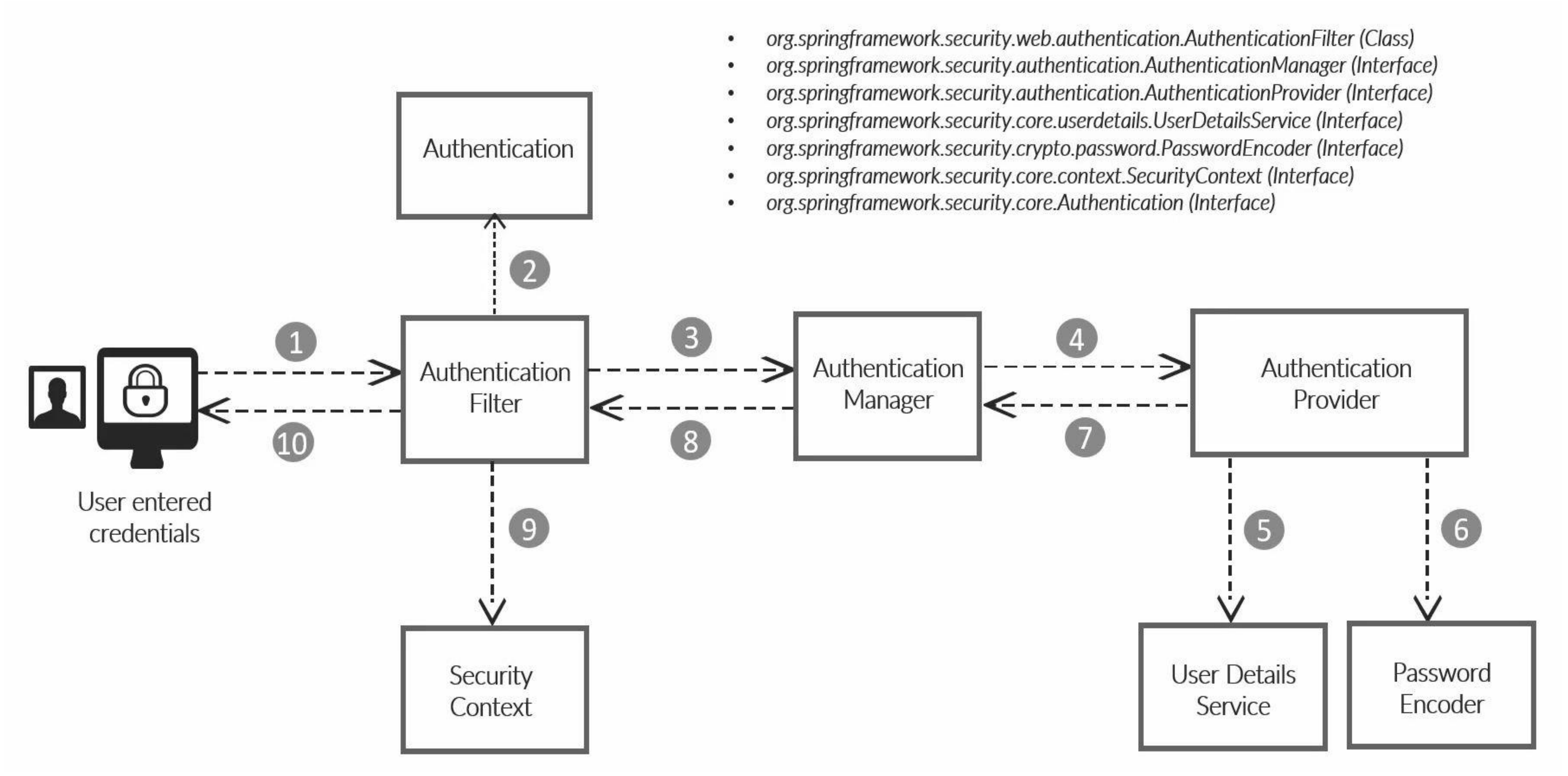
- If we want to add a custom user name and password in the Spring application for authentication we can add it easily using application.properties

```
spring.security.user.name= user01
spring.security.user.password=12345
```



Spring Security Architecture

axess academy



- *org.springframework.security.web.authentication.AuthenticationFilter (Class)*
- *org.springframework.security.authentication.AuthenticationManager (Interface)*
- *org.springframework.security.authentication.AuthenticationProvider (Interface)*
- *org.springframework.security.core.userdetails.UserDetailsService (Interface)*
- *org.springframework.security.crypto.password.PasswordEncoder (Interface)*
- *org.springframework.security.core.context.SecurityContext (Interface)*
- *org.springframework.security.core.Authentication (Interface)*

Spring Security Architecture Terms

Filter Chain

Intercepts all incoming requests

- Framework automatically registers a filters chain that intercepts all incoming requests. Each filter handles a particular use case.

UserDetailsService

Interface

- Core Interface that loads user-specific data
- Exposes single function: ***loadUserByUsername*** **-** accepts username as a parameter and returns the user identity object**.**

Authentication manager

Manages the providers

- Acts a coordinator where multiple providers are registered and based on the request type, it will deliver an authentication request to the correct provider.

Password Encoder

Manages the password encoding

- Different implementation of password encoding, and encryption technique is available.
- Most widely used technique - ***BCryptPasswordEncoder***

Authentication Provider

Processes specific types of authentication.

Its interface exposes two functions.

- ***authenticate*** function authentication with the request
- ***supports*** function checks if this provider supports the indicated authentication type
- Providers like **DaoAuthenticationProvider** which retrieves user details from **UserDetailsService**

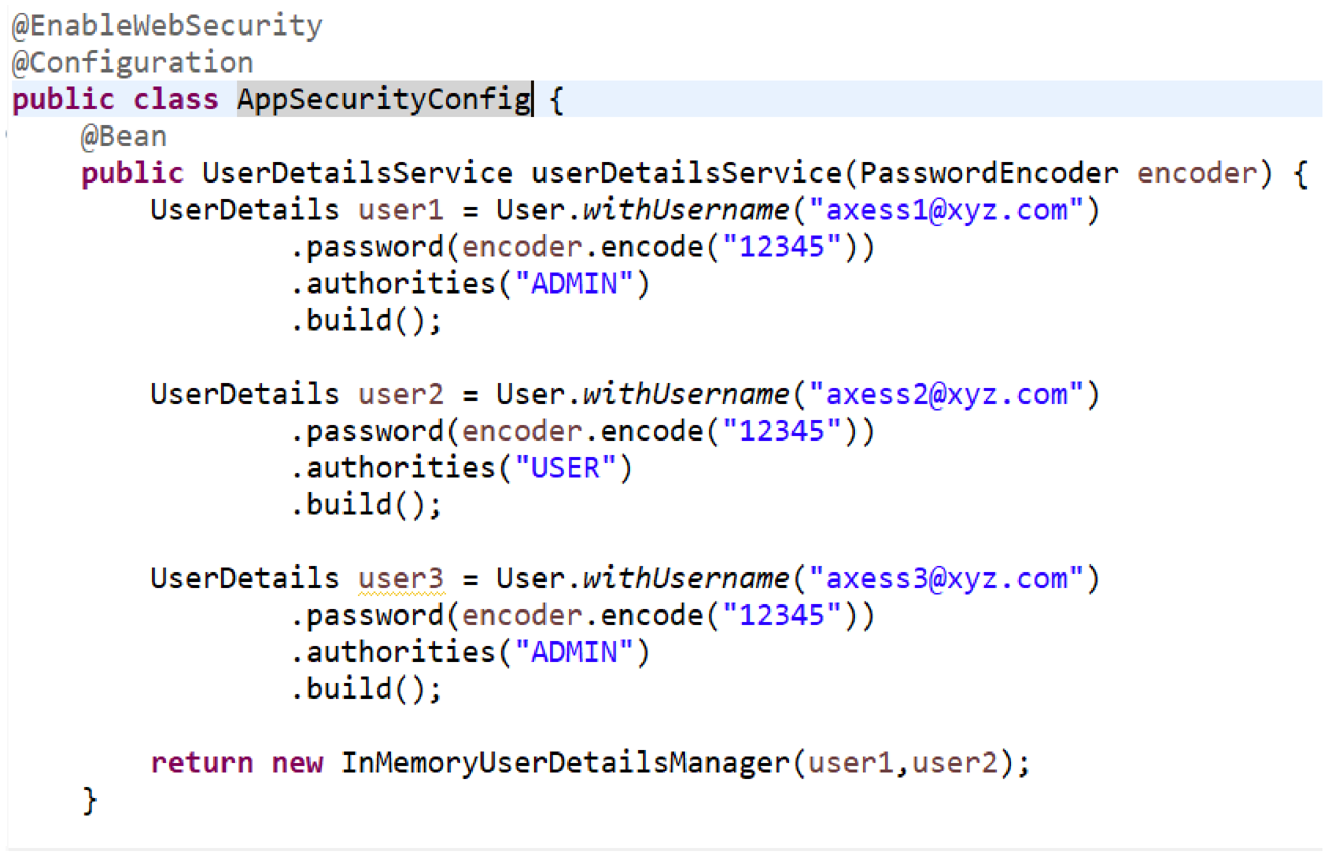
Configuration

Customize the Spring security configuration

- ***@EnableWebSecurity*** configuration class need to be annotated
- Override the methods to configure the authentication manager and web security.

UserDetailsService in Configuration file

- We implement Role Based Access by creating a java class and applying @EnableWebSecurity and @Configuration on top of it.
- @EnableWebSecurity enables Spring Security features in the application, whereas @Configuration represents that this class is a configuration class.
- UserDetailsService is Core interface which loads user-specific data.
- InMemoryUserDetailsManager is Non-persistent implementation of UserDetailsManager which is backed by an in-memory map.

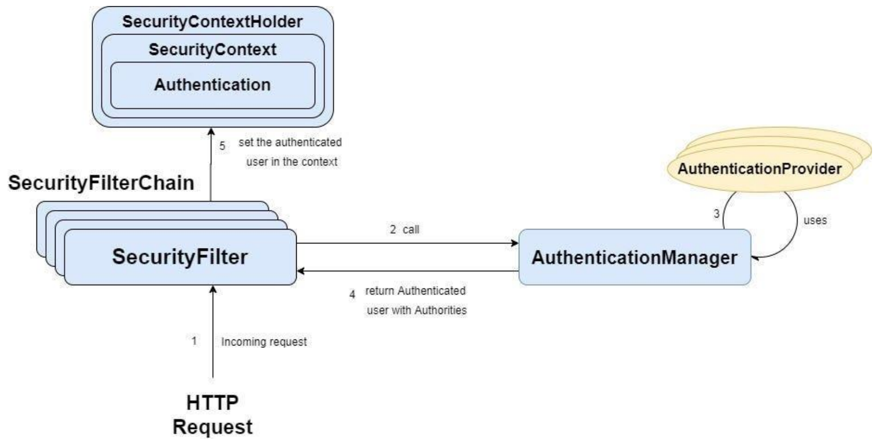


```
@EnableWebSecurity
@Configuration
public class AppSecurityConfig {
    @Bean
    public UserDetailsService userDetailsService(PasswordEncoder encoder) {
        UserDetails user1 = User.withUsername("axess1@xyz.com")
                .password(encoder.encode("12345"))
                .authorities("ADMIN")
                .build();

        UserDetails user2 = User.withUsername("axess2@xyz.com")
                .password(encoder.encode("12345"))
                .authorities("USER")
                .build();

        UserDetails user3 = User.withUsername("axess3@xyz.com")
                .password(encoder.encode("12345"))
                .authorities("ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user1,user2);
    }
```

axess academy

SecurityFilterChain in configuration file

- SecurityFilterChain is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, and so on) within your application.



```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.authorizeHttpRequests(auth -> {
        auth.requestMatchers("/api/customer/**").hasAuthority("ADMIN");
        auth.requestMatchers("/api/account/**").hasAuthority("USER");
    })
    .formLogin(withDefaults())
    .build();
}
```

PasswordEncoder in configuration file

- Service interface for encoding passwords.
- Implemented classes are
  - NoOpPasswordEncoder
  - StandardPasswordEncoder
  - Pbkdf2PasswordEncoder
  - BCryptPasswordEncoder
  - ScryptPasswordEncoder
- The preferred implementation is BCryptPasswordEncoder.

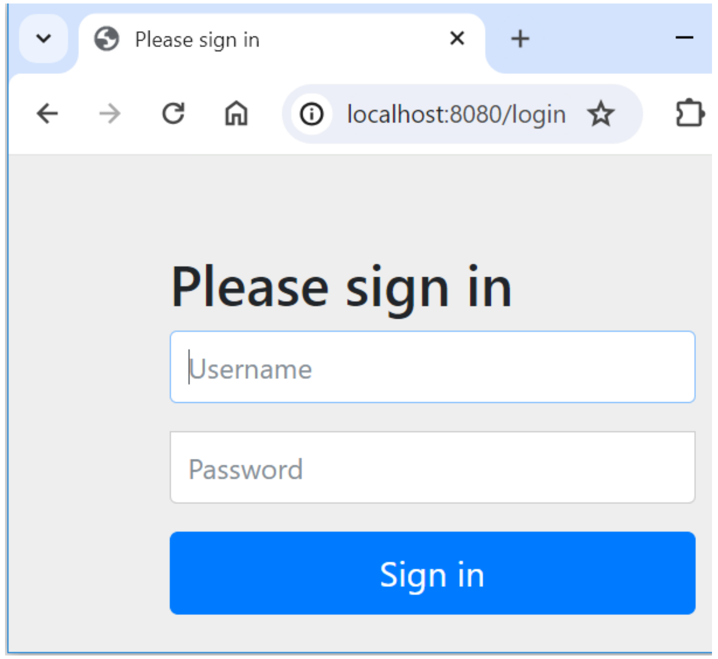
```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Use Case 2 on Spring Security

axess academy

Use Case 2

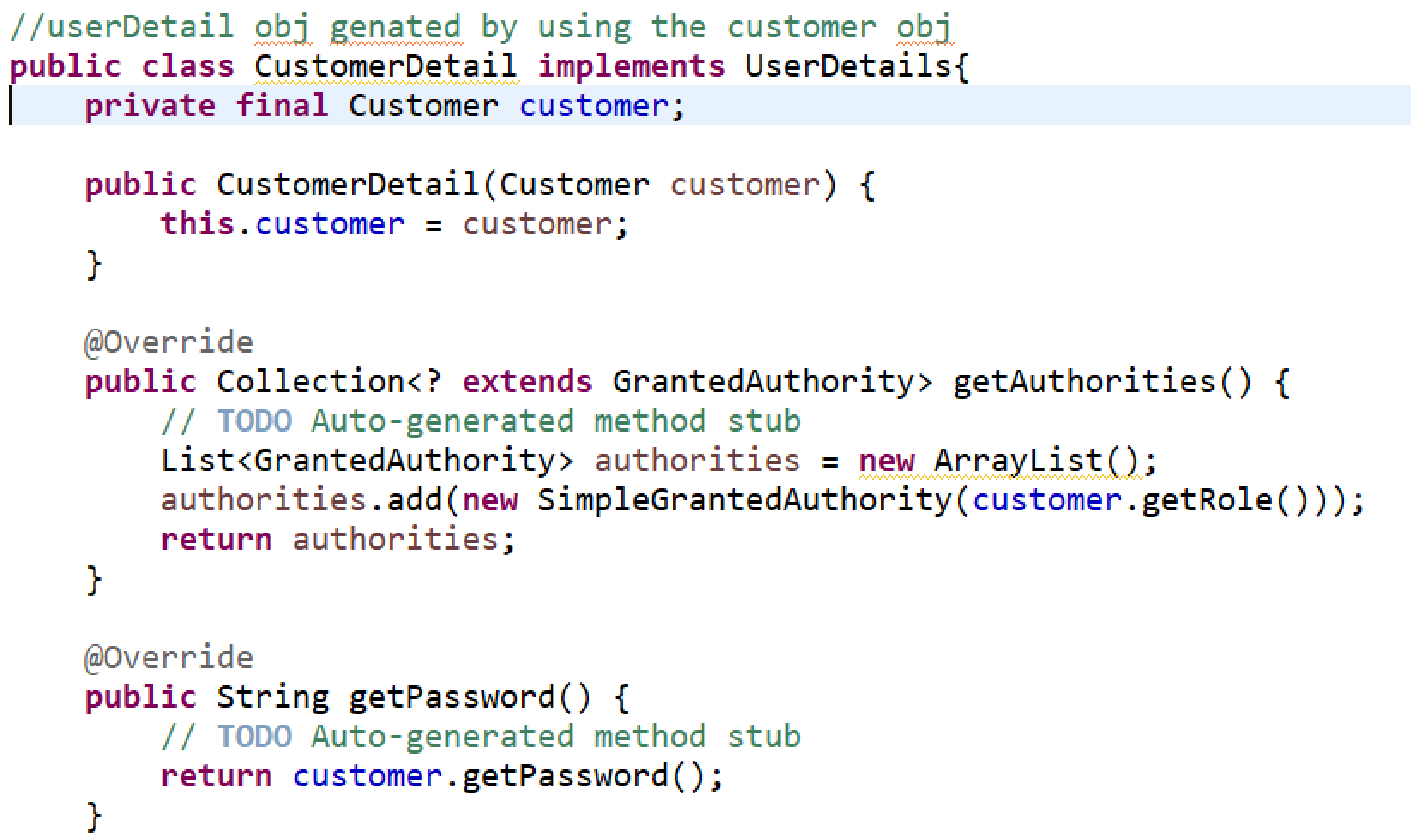
- Authenticate and Authorize customers based on their roles from data base.
- Implement by using UserDetailService.



axess academy

Security Configuration from customer table (DB)

- Create customerDetail class which implements UserDetails



```java
//userDetail obj genated by using the customer obj
public class CustomerDetail implements UserDetails{
    private final Customer customer;

    public CustomerDetail(Customer customer) {
        this.customer = customer;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        // TODO Auto-generated method stub
        List<GrantedAuthority> authorities = new ArrayList();
        authorities.add(new SimpleGrantedAuthority(customer.getRole()));
        return authorities;
    }

    @Override
    public String getPassword() {
        // TODO Auto-generated method stub
        return customer.getPassword();
    }
```

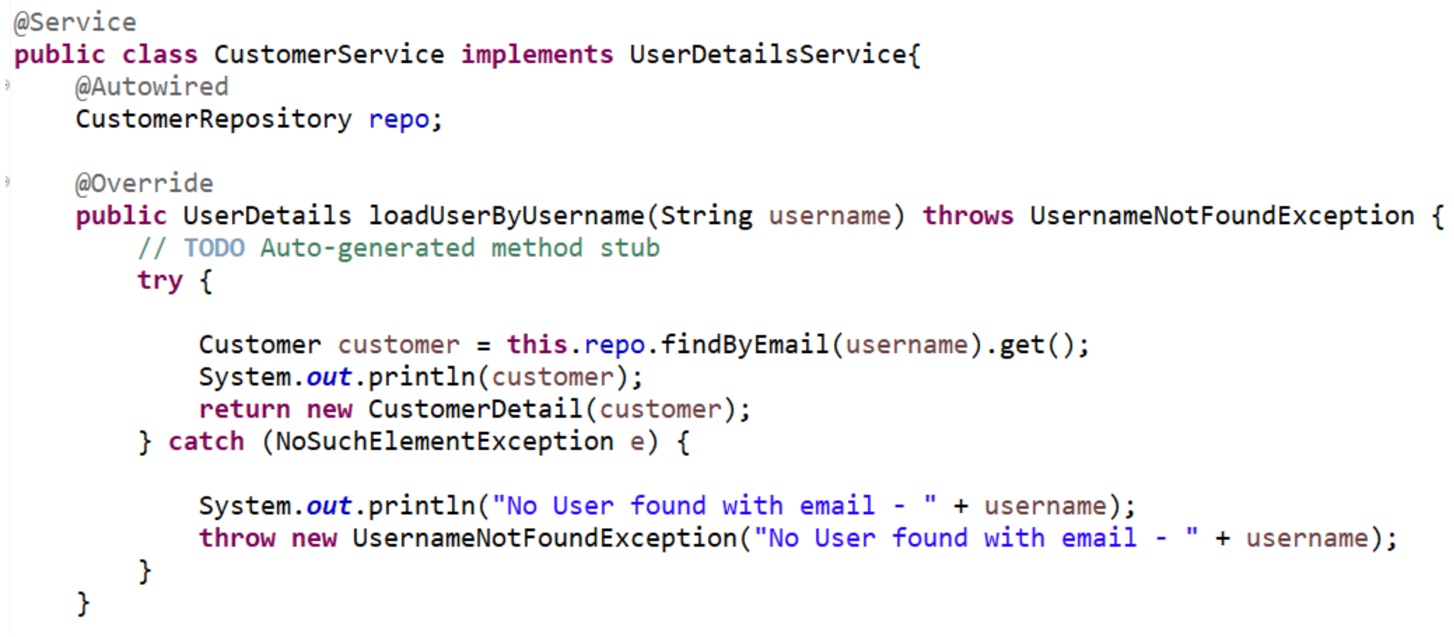
Repository

```java
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer>{
    Optional<Customer> findByEmail(String email);
}
```

axess academy

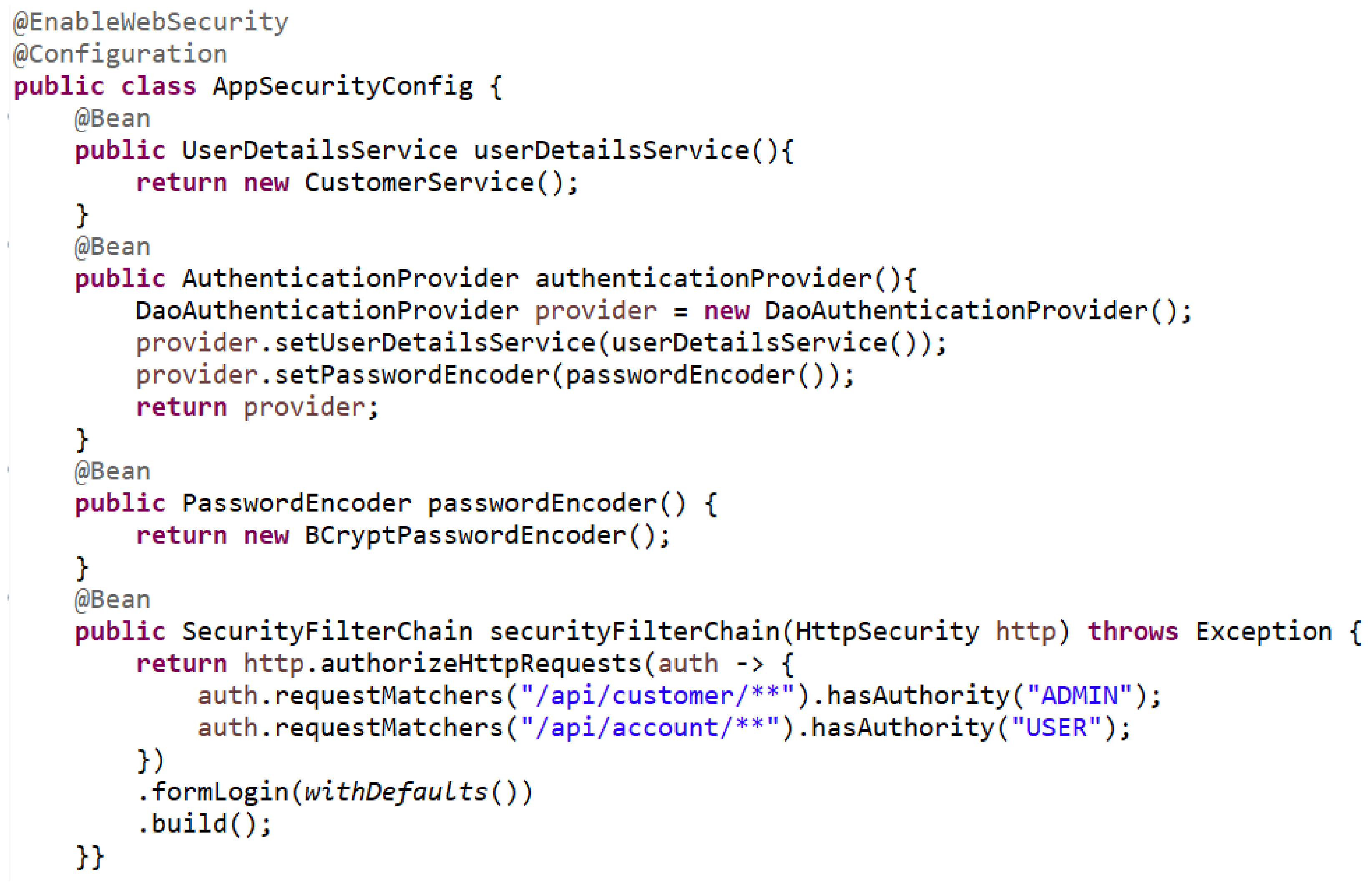
Implementation of UserDetailsService

- Create customerService class should implement interface UserDetailsService (Provided by Spring)
- Equally important, Override loadUserByUsername(String username) method of interface UserDetailsService in Customer class.
- As part of implementation,
  - Get your Customer Object with the help of username/email from CustomerRepository.
  - Convert your Customer Object into Spring's predefined User object (org.springframework.security.core.userdetails.User) accordingly.
  - Return Spring defined User object which is an implementation of UserDetails(method's return type).



```
@Service
public class CustomerService implements UserDetailsService{
    @Autowired
    CustomerRepository repo;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // TODO Auto-generated method stub
        try {

            Customer customer = this.repo.findByEmail(username).get();
            System.out.println(customer);
            return new CustomerDetail(customer);
        } catch (NoSuchElementException e) {

            System.out.println("No User found with email - " + username);
            throw new UsernameNotFoundException("No User found with email - " + username);
        }
    }
```

Configuration file



```
@EnableWebSecurity
@Configuration
public class AppSecurityConfig {
    @Bean
    public UserDetailsService userDetailsService(){
        return new CustomerService();
    }
    @Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService());
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http.authorizeHttpRequests(auth -> {
            auth.requestMatchers("/api/customer/**").hasAuthority("ADMIN");
            auth.requestMatchers("/api/account/**").hasAuthority("USER");
        })
        .formLogin(withDefaults())
        .build();
    }}
```