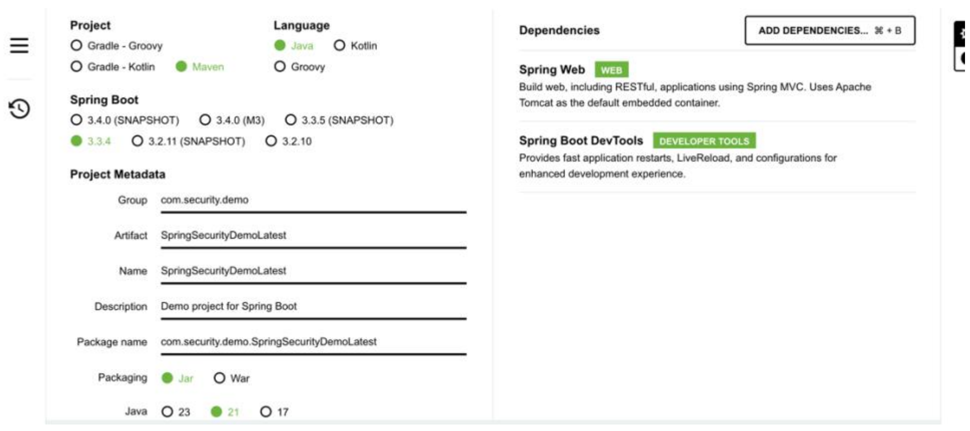
## Table of Contents

## 1. Spring Boot Maven Project

1.1. Create a Spring boot maven project with the following dependencies from start.spring.io



**Project**
Gradle - Groovy
Gradle - Kotlin
[✓ Maven]

**Language**
[✓ Java]
Kotlin
Groovy

**Spring Boot**
3.4.0 (SNAPSHOT)
3.4.0 (M3)
3.3.5 (SNAPSHOT)
[✓ 3.3.4]
3.2.11 (SNAPSHOT)
3.2.10

**Project Metadata**

| | |
|---|---|
| Group | com.security.demo |
| Artifact | SpringSecurityDemoLatest |
| Name | SpringSecurityDemoLatest |
| Description | Demo project for Spring Boot |
| Package name | com.security.demo.SpringSecurityDemoLatest |
| Packaging | [✓ Jar] War |
| Java | 23 [✓ 21] 17 |

**Dependencies**

**Spring Web** WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot DevTools** DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

1.2. Unzip the project and open in the editor
1.3. The embedded tomcat with spring boot web includes a light weight server which is the tomcat core and is capable of processing HTTP requests and send JSON as a response
1.4. Applications that use devtools will automatically restart whenever files on the classpath change
1.5. Add below in properties file :

```
spring.application.name=${SPRING_APP_NAME:SpringSecurityDemoLatest}

logging.pattern.console = ${LOGPATTERN_CONSOLE:%green(%d{HH:mm:ss.SSS}) %blue(%-5level)
%red([%thread]) %yellow(%logger{15}) - %msg%n}
```

## 2. Create a Rest Controller as follows:

```
@RestController
public class WelcomeController {
    @GetMapping("/welcome")
    public String sayWelcome(){
        return "Welcome to spring application for security;
    }
}
```

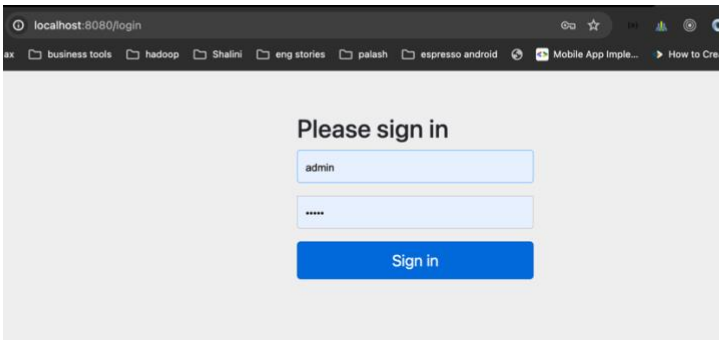
Start the application and open a browser. Pasting the below url should display the message returned by the method.
http://localhost:8080/welcome

## 3. Add Security

3.1. Add below dependency in pom.xml file:

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

3.2. Restart the server and hitting the same url this time should take you to a login page as shown in the screenshot



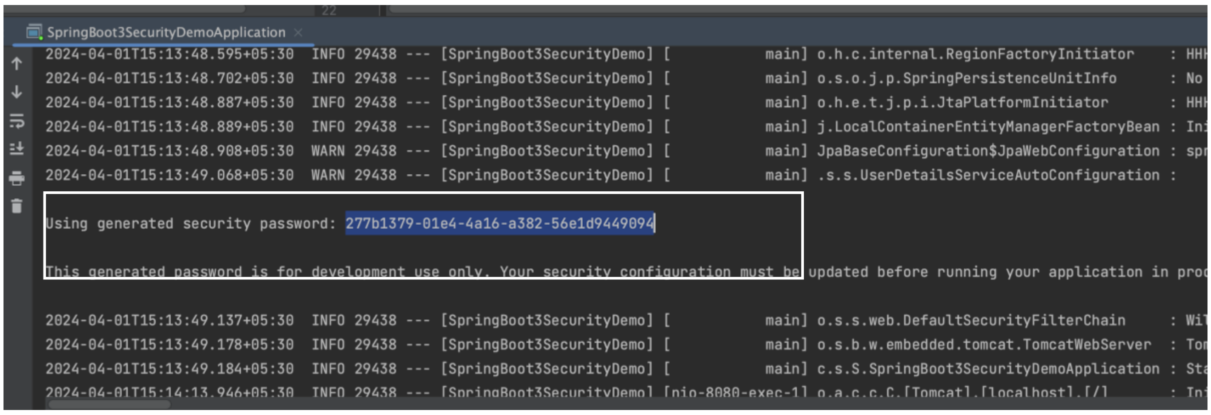
Please sign in

admin

•••••

Sign in

3.3. By default all the urls are secured by spring security, the default username is user and password is generated in the console as follows:



```
SpringBoot3SecurityDemoApplication
2024-04-01T15:13:48.595+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] o.h.c.internal.RegionFactoryInitiator    : HHH
2024-04-01T15:13:48.702+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No
2024-04-01T15:13:48.887+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH
2024-04-01T15:13:48.889+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] j.LocalContainerEntityManagerFactoryBean : Ini
2024-04-01T15:13:48.908+05:30  WARN 29438 --- [SpringBoot3SecurityDemo] [           main] JpaBaseConfiguration$JpaWebConfiguration : spr
2024-04-01T15:13:49.068+05:30  WARN 29438 --- [SpringBoot3SecurityDemo] [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 277b1379-01e4-4a16-a382-56e1d9449094

This generated password is for development use only. Your security configuration must be updated before running your application in proc

2024-04-01T15:13:49.137+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] o.s.s.web.DefaultSecurityFilterChain     : Wit
2024-04-01T15:13:49.178+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
2024-04-01T15:13:49.184+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [           main] c.s.S.SpringBoot3SecurityDemoApplication : Sta
2024-04-01T15:14:13.946+05:30  INFO 29438 --- [SpringBoot3SecurityDemo] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Ini
```

Type in user and paste the password , you should be able to login and test the urls
http://localhost:8080/welcome

3.4. Either Click Navigate->Class or Cmd-O(MAC), Cntrl-O(Windows), in the pop up box type SecurityProperties and click Download Sources. Should see the default credentials used to login.

## 4. Add Custom credentials

4.1. Add below for custom credentials in properties file:
spring.security.user.name=${SECURITY_USERNAME:demo}
spring.security.user.password=${SECURITY_PASSWORD:12345}

4.2. Now the credentials will be as defined above.

4.3. Add below property in properties file to understand the flow of spring security classes
logging.level.org.springframework.security=${SPRING_SECURITY_LOG_LEVEL:TRACE}

4.4. Look at cookies within dev tools of browser and there is JSESSIONID. Modifying this id after successful login and refresh the page will redirect to the login page

## 5. Prepare Controllers

5.1. Create below controllers in the controller package

```
package com.security.demo.SpringSecurityDemoLatest.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AccountController {

    @GetMapping("/myAccount")
    public  String getAccountDetails () {
        return "Here are the account details from the DB";
    }
```

```java
}

package com.security.demo.SpringSecurityDemoLatest.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class BalanceController {

    @GetMapping("/myBalance")
    public  String getBalanceDetails () {
        return "Here are the balance details from the DB";
    }

}

package com.security.demo.SpringSecurityDemoLatest.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CardsController {

    @GetMapping("/myCards")
    public  String getCardsDetails () {
        return "Here are the card details from the DB";
    }

}

package com.security.demo.SpringSecurityDemoLatest.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ContactController {

    @GetMapping("/contact")
    public  String saveContactInquiryDetails () {
        return "Inquiry details are saved to the DB";
    }

}

package com.security.demo.SpringSecurityDemoLatest.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class LoansController {

    @GetMapping("/myLoans")
    public  String getLoansDetails () {
        return "Here are the loans details from the DB";
    }

}

package com.security.demo.SpringSecurityDemoLatest.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class NoticesController {

   @GetMapping("/notices")
   public  String getNotices () {
      return "Here are the notices details from the DB";
   }

}
```

## 6. Security Config

6.1. Create a class ProjectSecurityConfig within config package as follows:

```
@Configuration
public class ProjectSecurityConfig {
 @Bean
  SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
      http.authorizeHttpRequests((requests) -> requests.anyRequest().permitAll());        Line 1
      http.formLogin(withDefaults());                                                     Line 2
      http.httpBasic(withDefaults());                                                     Line 3
      return http.build();
   }
}
```

6.2. The above will allow access to all the routes.
6.3. To deny you can use following by replacing **Line 1:**

```
http.authorizeHttpRequests((requests) -> requests.anyRequest().denyAll());
```

6.4. To allow some urls and authenticate others, update the above method as follows by replacing **Line 1:**

```
http.authorizeHttpRequests((requests) -> requests
        .requestMatchers("/myAccount", "/myBalance", "/myLoans", "/myCards").authenticated()
        .requestMatchers("/notices", "/contact", "/error").permitAll());
```

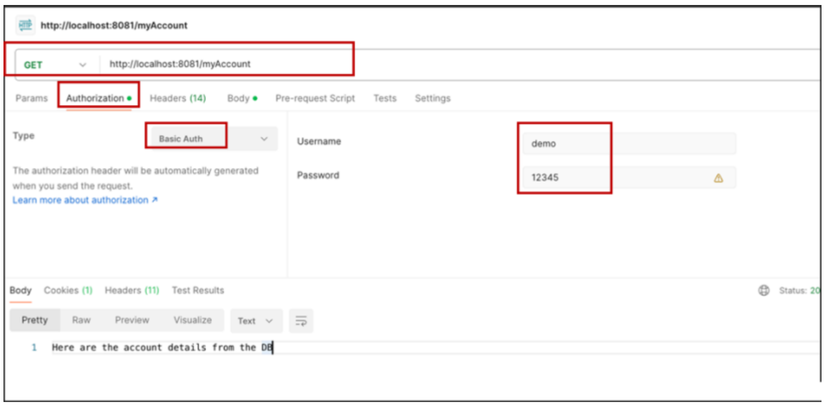
6.5. Above will allow access to routes with permit all but will have to login for other urls

## 7. Disable form login and http basic auth

7.1. To disable http form login add the below code by replacing **Line 2**:
`http.formLogin(AbstractHttpConfigurer::disable);`
7.2. Now restart the server and you should see browser default http basic form and not the spring login page
7.3. If you disable both the forms then none of the authenticated url's will be accessible

## 8. POSTMAN

8.1. Update the code to have form login and http basic auth enabled. Open Postman and to test authenticated url's type in the details as shown below:



GET http://localhost:8081/myAccount

Authorization

Type: Basic Auth

The authorization header will be automatically generated when you send the request.

Username: demo

Password: 12345

Body

```
Here are the account details from the DB
```

## 9. Using In memory details Manager

9.1. Adding credentials in properties file is not the correct way. So lets modify the Security Config to provide user details information and also get the flexibility to add authority

```
@Bean
    public UserDetailsService userDetailsService() {
        UserDetails user = User.withUsername("user").password("12345").authorities("read").build();
        UserDetails admin = User.withUsername("admin")
                .password("{54321")
                .authorities("admin").build();
        return new InMemoryUserDetailsManager(user, admin);
    }
```

9.2. Running the server and entering credentials for authenticated page, will give 500 error for password not being encoded.

9.3. Spring security makes it mandatory to encode the password or use the code as follows to bypass the encoder:

```
@Bean
    public UserDetailsService userDetailsService() {
        UserDetails user = User.withUsername("user").password("{noop}12345").authorities("read").build();
        UserDetails admin = User.withUsername("{noop}admin")
                .password("{54321")
                .authorities("admin").build();
        return new InMemoryUserDetailsManager(user, admin);
    }
```

9.4. Now restarting the server allows to login successfully

9.5. Hard coding credentials is not a good practice. Also passwords should be encoded. Add the below code in Security Config:

```
@Bean
    public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
```

9.6. Update the userDetailsService method of encoding password as follows:

```
UserDetails user = User.withUsername("user")
        .password(passwordEncoder().encode("12345"))
        .authorities("read").build();
```

9.7. Restart the server and should be able to login successfully

9.8. Alternatively go to below website and get the encrypted code:
https://bcrypt-generator.com/

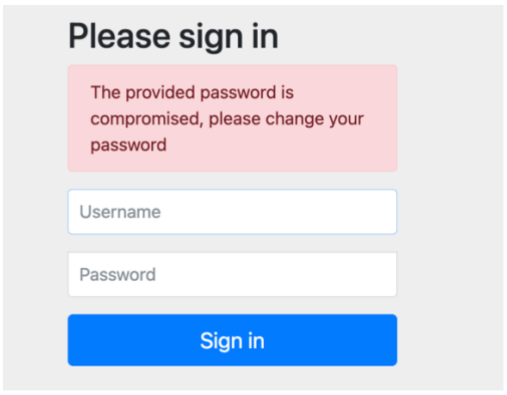
9.9. Update as below to use encrypted code instead of hard-coded passwords:

```
UserDetails admin = User.withUsername("admin")
            .password("{bcrypt}$2a$12$28hTdBcrUf7xzTwpPKvnLOlOcNqXaVsIu/A76pjE.//p7arHpRL1m")
                .authorities("admin").build();
```

9.10. To stop the users from using compromised passwords add the below in Security Config:

```
@Bean
public CompromisedPasswordChecker compromisedPasswordChecker() {
    return new HaveIBeenPwnedRestApiPasswordChecker();
}
```

9.11. Now restarting the server try to login it gives an error for compromised passwords:


Please sign in

The provided password is compromised, please change your password

9.12. Change the password in the code for a strong password and should be able to login

## 10. Using database to manage users

10.1. Create a database bank in MYSQL database. Execute below queries to create tables and insert data:

```
create table users(username varchar(50) not null primary key,password varchar(500) not null,enabled boolean not null);
create table authorities (username varchar(50) not null,authority varchar(50) not null,constraint fk_authorities_users foreign key(username) references users(username));
create unique index ix_auth_username on authorities (username,authority);

INSERT IGNORE INTO `users` VALUES ('user', '{noop}shalini.techgatha@12345', '1');
INSERT IGNORE INTO `authorities` VALUES ('user', 'read');

INSERT IGNORE INTO `users` VALUES ('admin', '{bcrypt}$2a$12$28hTdBcrUf7xzTwpPKvnLOlOcNqXaVsIu/A76pjE.//p7arHpRL1m', '1');
INSERT IGNORE INTO `authorities` VALUES ('admin', 'admin');
```

10.2. Add below dependencies in pom.xml file:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
</dependency>
```

10.3. Update the properties file as follows to provide database connection parameters:

```
spring.datasource.url=jdbc:mysql://${DATABASE_HOST:localhost}:${DATABASE_PORT:8889}/${DATABASE_NAME:bank}
spring.datasource.username=${DATABASE_USERNAME:root}
spring.datasource.password=${DATABASE_PASSWORD:root}
spring.jpa.show-sql=${JPA_SHOW_SQL:true}
spring.jpa.properties.hibernate.format_sql=${HIBERNATE_FORMAT_SQL:true}
```

10.4. Update the userDetailsService method of Security Config to connect with database

```
@Bean
public UserDetailsService userDetailsService(DataSource dataSource) {

    return new JdbcUserDetailsManager(dataSource);
}
```

10.5. With these changes restart the server and should be able to login with the credentials created in the database

## 11. Create custom table and user details manager

11.1. Till now we just followed the spring defaults. There may be scenarios where we will need to create our own custom tables as well.

11.2. Create table as follows and insert some dummy data:

```
CREATE TABLE `customer` (
  `id` int NOT NULL AUTO_INCREMENT,
  `email` varchar(45) NOT NULL,
  `pwd` varchar(200) NOT NULL,
  `role` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
);
```

```
INSERT  INTO `customer` (`email`, `pwd`, `role`) VALUES ('happy@example.com',
'{noop}shalini.techgatha@12345', 'read');
INSERT  INTO `customer` (`email`, `pwd`, `role`) VALUES ('admin@example.com',
'{{bcrypt}$2a$12$28hTdBcrUf7xzTwpPKvnLOlOcNqXaVsIu/A76pjE.//p7arHpRL1m', 'admin');
```

11.3. Create JPA Entities to map with the above table. [ ADD LOMBOK DEPENDENCY ]

```
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>annotationProcessor</scope>
</dependency>
```

```
package com.security.demo.SpringSecurityDemoLatest.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Table(name = "customer")
@Getter @Setter
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String email;
    private String pwd;
    @Column(name = "role")
    private String role;


}
```

```
package com.security.demo.SpringSecurityDemoLatest.repository;


import com.security.demo.SpringSecurityDemoLatest.model.Customer;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface CustomerRepository extends CrudRepository<Customer,Long> {

    Optional<Customer> findByEmail(String email);

}
```

11.4. Since we have our custom entity for authentication, we need to create our own AuthenticationProvider.  Add below class in the config folder:

```
package com.security.demo.SpringSecurityDemoLatest.config;


import com.security.demo.SpringSecurityDemoLatest.model.Customer;
import com.security.demo.SpringSecurityDemoLatest.repository.CustomerRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class CustomerUserDetailsService implements UserDetailsService {

    private final CustomerRepository customerRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Customer customer = customerRepository.findByEmail(username).orElseThrow(() -> new
                UsernameNotFoundException("User details not found for the user: " + username));
        List<GrantedAuthority> authorities = List.of(new SimpleGrantedAuthority(customer.getRole()));
        return new User(customer.getEmail(), customer.getPwd(), authorities);
    }
}
```

**11.5. NOTE: Comment the userDetailsService method in the Security Config class since now we are using our own authentication provider**

11.6. Restart the server and it should work with new customer credentials we created.

## 12. Register customer

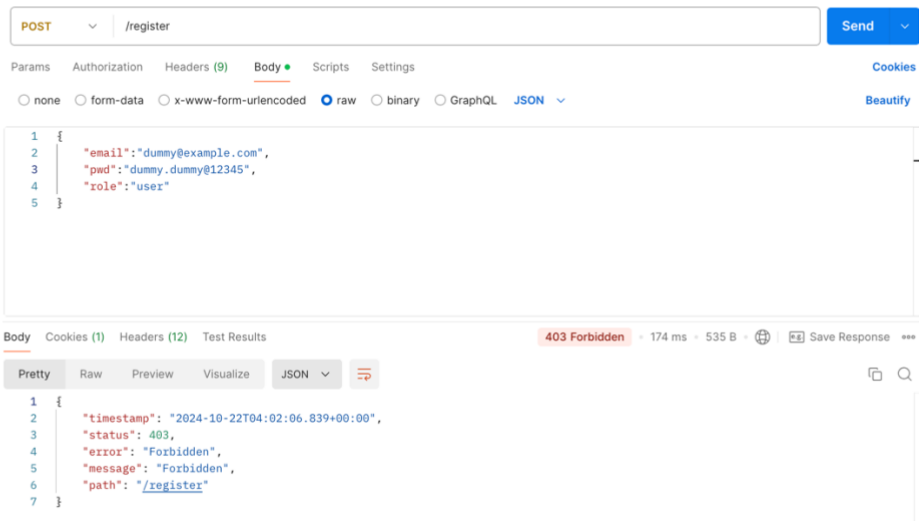
12.1. Create a rest controller with post mapping to add new customer

```
package com.security.demo.SpringSecurityDemoLatest.controller;

import com.security.demo.SpringSecurityDemoLatest.model.Customer;
import com.security.demo.SpringSecurityDemoLatest.repository.CustomerRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
public class UserController {

    private final CustomerRepository customerRepository;
    private final PasswordEncoder passwordEncoder;

    @PostMapping("/register")
    public ResponseEntity<String> registerUser(@RequestBody Customer customer) {
        try {
                String hashPwd = passwordEncoder.encode(customer.getPwd());
                customer.setPwd(hashPwd);
                Customer savedCustomer = customerRepository.save(customer);

                if(savedCustomer.getId()>0) {
                    return ResponseEntity.status(HttpStatus.CREATED).
                            body("Given user details are successfully registered");
                } else {
                    return ResponseEntity.status(HttpStatus.BAD_REQUEST).
                            body("User registration failed");
                }
        } catch (Exception ex) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).
                    body("An exception occurred: " + ex.getMessage());
        }
    }
  }
```

12.2. Add this new url to permit all in the Security Config class:

```
.requestMatchers("/notices", "/contact", "/error","/register").permitAll());
```

12.3. Now restart the server and send POST request as follows. You will get error


POST /register

Body raw JSON

```
{
    "email":"dummy@example.com",
    "pwd":"dummy.dummy@12345",
    "role":"user"
}
```

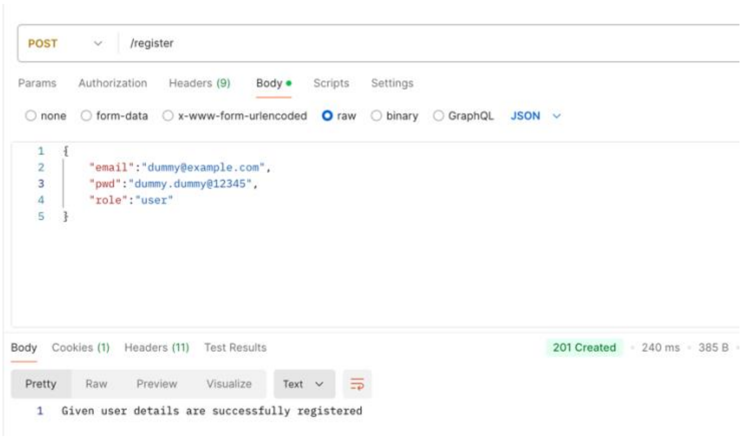
403 Forbidden 174 ms 535 B

```
{
    "timestamp": "2024-10-22T04:02:06.839+00:00",
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/register"
}
```

12.4. The reason for the error is we need to disable csrf. Update method as follows:

```
@Bean
 SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
      http.csrf(AbstractHttpConfigurer::disable)
           .authorizeHttpRequests((requests) -> requests
               .requestMatchers("/myAccount", "/myBalance", "/myLoans","/myCards").authenticated()
               .requestMatchers("/notices", "/contact", "/error","/register").permitAll());
       http.formLogin(withDefaults());
       http.httpBasic(withDefaults());
       return http.build();
   }
```

12.5. Now restart the server and post request should be successful with customer data added in the database.

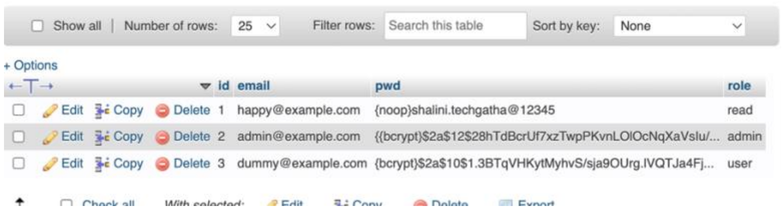

POST /register

Body raw JSON

```
{
    "email":"dummy@example.com",
    "pwd":"dummy.dummy@12345",
    "role":"user"
}
```

201 Created 240 ms 385 B

```
Given user details are successfully registered
```



| | id | email | pwd | role |
|---|---|---|---|---|
| Edit Copy Delete | 1 | happy@example.com | {noop}shalini.techgatha@12345 | read |
| Edit Copy Delete | 2 | admin@example.com | {{bcrypt}$2a$12$28hTdBcrUf7xzTwpPKvnLOlOcNqXaVslu/... | admin |
| Edit Copy Delete | 3 | dummy@example.com | {bcrypt}$2a$10$1.3BTqVHKytMyhvS/sja9OUrg.lVQTJa4Fj... | user |

## 13. Custom Authentication Provider

13.1. Create a class that implements the AuthenticationProvider as follows:

```
package com.security.demo.SpringSecurityDemoLatest.config;

@Component
@Profile("prod")
@RequiredArgsConstructor
public class CustomerUsernamePwdAuthenticationProvider implements AuthenticationProvider {

    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {

        String username = authentication.getName();
        String pwd = authentication.getCredentials().toString();
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        if (passwordEncoder.matches(pwd, userDetails.getPassword())) {
            // Fetch Age details and perform validation to check if age >18
            return new UsernamePasswordAuthenticationToken(username,pwd,userDetails.getAuthorities());
        }else {
            throw new BadCredentialsException("Invalid password!");
        }

    }
    @Override
    public boolean supports(Class<?> authentication) {
        return (UsernamePasswordAuthenticationToken.class.isAssignableFrom(authentication));
    }
}
```

13.2. Spring security framework will automatically pick the CustomerUserDetailsService implementation for UserDetailsService.

## 14. Profiles

**14.1.** Create application_prod.properties file as follows:

```
spring.config.activate.on-profile= prod
spring.application.name=${SPRING_APP_NAME:SpringSecurityDemoLatest}

logging.pattern.console = ${LOGPATTERN_CONSOLE:%green(%d{HH:mm:ss.SSS}) %blue(%-5level)
%red([%thread]) %yellow(%logger{15}) - %msg%n}

spring.security.user.name=${SECURITY_USERNAME:demo}
spring.security.user.password=${SECURITY_PASSWORD:12345}

logging.level.org.springframework.security=${SPRING_SECURITY_LOG_LEVEL:ERROR}
server.port=8081


spring.datasource.url=jdbc:mysql://${DATABASE_HOST:localhost}:${DATABASE_PORT:8889}/${DATABAS
E_NAME:bank}
spring.datasource.username=${DATABASE_USERNAME:root}
spring.datasource.password=${DATABASE_PASSWORD:root}
spring.jpa.show-sql=${JPA_SHOW_SQL:false}
spring.jpa.properties.hibernate.format_sql=${HIBERNATE_FORMAT_SQL:false}
```

14.2. Modify the application.properties file for active profile add below code

```
spring.config.import = application_prod.properties
spring.profiles.active = prod
```

14.3. Create Security Config and Authentication Providers for prod profile by adding @Profile("prod") and for default profile by adding @Profile("!prod")

14.4. Remove password authentication from the AuthenticationProvider with !prod Profile to make it easier for development or testing.

## 15. Accept HTTPS Traffic

15.1. To allow only https traffic add below in prod Security Config class:

```
http.requiresChannel(rcc-> rcc.anyRequest().requiresSecure()) // HTTPS
```

15.2. To allow insecure access for local environment add below in default security Config class

```
http.requiresChannel(rcc-> rcc.anyRequest().requiresInsecure()) // HTTP
```

## 16. Exception Handling in Spring Security

16.1. Below handlers will only work for HTTP Basic and REST API as for MVC we send HTML page and not JSON

16.2. To handle 401 i.e authentication failed exception and return a custom message, create a class that implements AuthenticationEntryPoint interface as follows:

```
package com.security.demo.SpringSecurityDemoLatest.exceptionhandling;

public class CustomBasicAuthenticationEntryPoint implements AuthenticationEntryPoint {
   @Override
   public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException
authException)
         throws IOException, ServletException {
      response.setHeader("bank-error-reason", "Authentication failed");
      response.sendError(HttpStatus.UNAUTHORIZED.value(), HttpStatus.UNAUTHORIZED.getReasonPhrase
   }
}
```

16.3. To inform spring security framework about the custom class, update the httpBasic default as follows:

```
http.httpBasic(hbc -> hbc.authenticationEntryPoint(new CustomBasicAuthenticationEntryPoint()));
```

16.4. To add custom messages update the method as below:

```
public class CustomBasicAuthenticationEntryPoint implements AuthenticationEntryPoint {
   @Override
   public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException
authException)
         throws IOException, ServletException {
      // Populate dynamic values
     LocalDateTime currentTimeStamp = LocalDateTime.now();
      String message = (authException != null && authException.getMessage() != null) ? "My Custom Message
"+authException.getMessage()
            : "Unauthorized";
      String path = request.getRequestURI();
      response.setHeader("bank-error-reason", "Authentication failed");
      response.setStatus(HttpStatus.UNAUTHORIZED.value());
      response.setContentType("application/json;charset=UTF-8");
      // Construct the JSON response
      String jsonResponse =
                String.format("{\"timestamp\": \"%s\", \"status\": %d, \"error\": \"%s\", \"message\": \"%s\", \"path\": \"%s\"}",
                        currentTimeStamp, HttpStatus.UNAUTHORIZED.value(),
                HttpStatus.UNAUTHORIZED.getReasonPhrase(), message, path);

                response.getWriter().write(jsonResponse);
    }
}
```

16.5. To handle 403 error ,add below class that implements AccessDeniedHandler

```
package com.security.demo.SpringSecurityDemoLatest.exceptionhandling;

public class CustomAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
            AccessDeniedException accessDeniedException) throws IOException, ServletException {
        // Populate dynamic values
        LocalDateTime currentTimeStamp = LocalDateTime.now();
        String message = (accessDeniedException != null && accessDeniedException.getMessage() != null) ?
                accessDeniedException.getMessage() : "Authorization failed";
        String path = request.getRequestURI();
        response.setHeader("bank-denied-reason", "Authorization failed");
        response.setStatus(HttpStatus.FORBIDDEN.value());
        response.setContentType("application/json;charset=UTF-8");
        // Construct the JSON response
        String jsonResponse =
                String.format("{\"timestamp\": \"%s\", \"status\": %d, \"error\": \"%s\", \"message\": \"%s\", \"path\": \"%s\"}",
                        currentTimeStamp, HttpStatus.FORBIDDEN.value(), HttpStatus.FORBIDDEN.getReasonPhrase(),
                        message, path);
        response.getWriter().write(jsonResponse);
    }
}
```

16.6. Update the Security config to configure for this handler

```
http.exceptionHandling(ehc -> ehc.accessDeniedHandler(new CustomAccessDeniedHandler()));
```

## 17. Session timeout and invalid session

17.1. Default time out is 30 mins. To override add below in properties file:

```
server.servlet.session.timeout=${SESSION_TIMEOUT:20m}
```

17.2. If m not provided it is seconds. But spring will not allow time out less than 2 mins

17.3. After the timeout it redirects to login page. To handle this invalid session management and send a message to user that session is invalid, add below in Security Config class:

```
http.sessionManagement(smc -> smc.invalidSessionUrl("/invalidSession"))
```

Add /invalidSession url in permitAll and create a REST API GET endpoint to return a session time out message.

After 2 mins sitting idle, browser redirects to invalidSession url or can also change the JSESSIONID, it redirects to this url

17.4. By default spring security allows multiple sessions for a single user. To test this access any secured url from more than 1 browser or postman. To restrict multiple sessions, update Security Config class as follows:

```
http.sessionManagement(smc -> smc.invalidSessionUrl("/invalidSession").maximumSessions(1))
```

Now if user tries to login again with same credentials it will logout from previous one, to verify this refresh in previous login page should get session expired message.

17.5. To prevent this behaviour and now allow login from other browser once logged in add the below:

```
http.sessionManagement(smc ->
smc.invalidSessionUrl("/invalidSession").maximumSessions(1).maxSessionsPreventsLogin(true))
```

## 18. CORS – Cross Origin Resource Sharing

18.1. Clone the SpringBootLatestWithAngular project and go through it to look for all the changes made in model, controller and repo packages

18.2. Now test this application by sending a POST request to /register and inserting a new customer record:

```
{
   "name":"John Doe",
   "email":"john@example.com",
   "pwd":"dummy.dummy@12345",
   "role":"user",
   "mobileNumber":"1212121212"
}
```

18.3. Need to install nodejs and angular-cli to run the angular application

18.4. To check for CORS we need a frontend application. Clone the angular project and run **npm install** from within the folder that contains package.json file. You should see node-modules folder created. Now run below command from within the same folder:

```
npm start
```

18.5. Once application is running it can be accessed at http://localhost:4200

18.6. Now try to click notices link and check in browser console for CORS error.

18.7. To enable CORS following are the options available to add on the respective controllers:

```
@CrossOrigin(origins = "http://localhost:4200") // Allows only a specific domain
```

OR

```
@CrossOrigin(origins = "*")
```

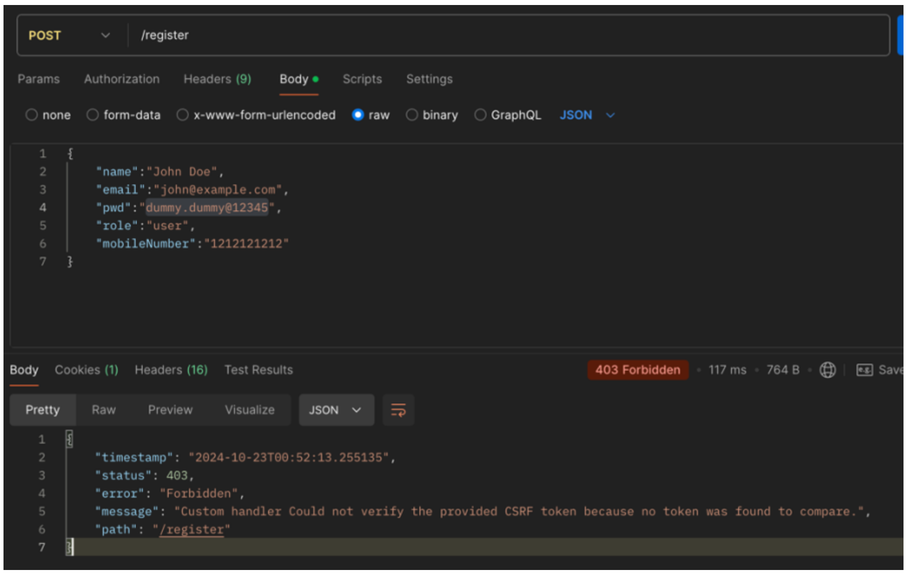
18.8. The drawback of above configuration is it needs to be added in all the controllers.

18.9. Instead can define CORS related configuration globally in Security Config as follows:

```
http.cors(corsConfig -> corsConfig.configurationSource(new CorsConfigurationSource() {
  @Override
  public CorsConfiguration getCorsConfiguration(HttpServletRequest request) {
      CorsConfiguration config = new CorsConfiguration();
      config.setAllowedOrigins(Collections.singletonList("http://localhost:4200"));
      config.setAllowedMethods(Collections.singletonList("*"));
      config.setAllowCredentials(true);
      config.setAllowedHeaders(Collections.singletonList("*"));
      config.setMaxAge(3600L);
      return config;
  }
}))
```

## 19. CSRF

19.1. Since we disable the csrf in the Security Config class we were able to make a POST request. Disabling will stop users make a POST request. We get following error:



POST /register

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "name":"John Doe",
    "email":"john@example.com",
    "pwd":"dummy.dummy@12345",
    "role":"user",
    "mobileNumber":"1212121212"
}
```

Body Cookies (1) Headers (16) Test Results 403 Forbidden 117 ms 764 B

Pretty Raw Preview Visualize JSON

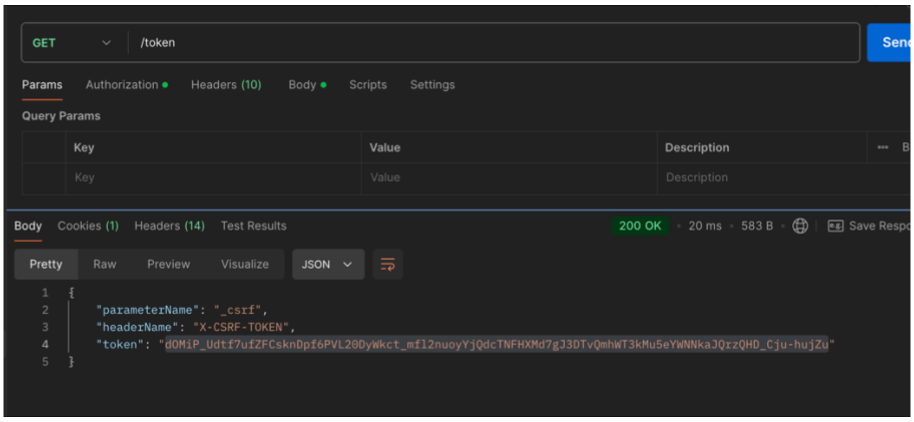
```
{
    "timestamp": "2024-10-23T00:52:13.255135",
    "status": 403,
    "error": "Forbidden",
    "message": "Custom handler Could not verify the provided CSRF token because no token was found to compare.",
    "path": "/register"
}
```

19.2. Even if we tried to add a contact from angular we will get the same error.

19.3. By default spring security provides with _csrf attribute and generates a csrf token. Add the below REST API, get the token and send it as part of header when making a POST request

```
@GetMapping("/token")
  public CsrfToken csrfToken (HttpServletRequest request) {
      CsrfToken token = (CsrfToken) request.getAttribute("_csrf");
      return token;
 }
```

19.4. Now from postman when making request you get the token as follows



GET /token

Params Authorization Headers (10) Body Scripts Settings

Query Params

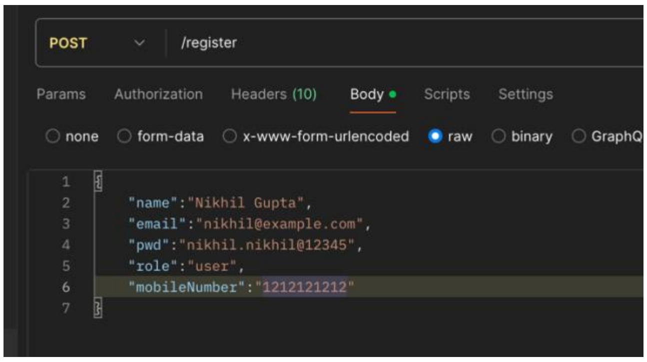
| Key | Value | Description |
|---|---|---|
| Key | Value | Description |

Body Cookies (1) Headers (14) Test Results 200 OK 20 ms 583 B

Pretty Raw Preview Visualize JSON

```
{
    "parameterName": "_csrf",
    "headerName": "X-CSRF-TOKEN",
    "token": "dOMiP_Udtf7ufZFCsknDpf6PVL20DyWkct_mfl2nuoyYjQdcTNFHXMd7gJ3DTvQmhWT3kMu5eYWNNkaJQrzQHD_Cju-hujZu"
}
```

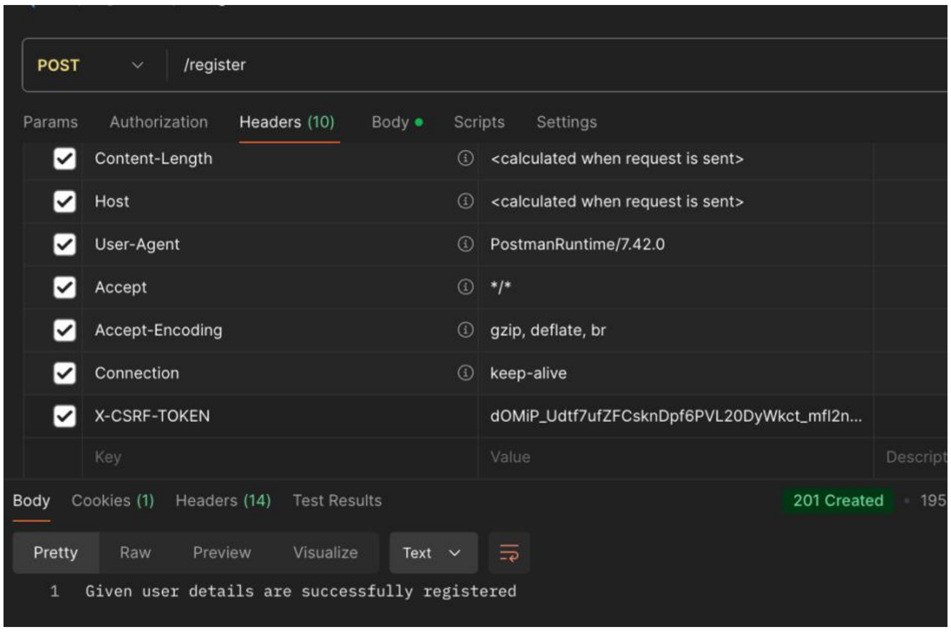
19.5. Copy the token and make POST request as follows: Do add JSON in body and header for csrf token


POST /register

Params Authorization Headers (10) Body ● Scripts Settings

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQ

```
{
    "name":"Nikhil Gupta",
    "email":"nikhil@example.com",
    "pwd":"nikhil.nikhil@12345",
    "role":"user",
    "mobileNumber":"1212121212"
}
```


POST /register

Params Authorization Headers (10) Body ● Scripts Settings

| Key | Value | Descript |
|---|---|---|
| ✓ Content-Length | <calculated when request is sent> | |
| ✓ Host | <calculated when request is sent> | |
| ✓ User-Agent | PostmanRuntime/7.42.0 | |
| ✓ Accept | */* | |
| ✓ Accept-Encoding | gzip, deflate, br | |
| ✓ Connection | keep-alive | |
| ✓ X-CSRF-TOKEN | dOMiP_Udtf7ufZFCsknDpf6PVL20DyWkct_mfl2n... | |

Body Cookies (1) Headers (14) Test Results 201 Created 195

Pretty Raw Preview Visualize Text

```
1  Given user details are successfully registered
```

19.6. Here first we need to make a request for token and then pass it. To customize for spring application to send the token as a part of request automatically need to add custom configurations

19.7. Add below for csrf token handling in Security Config class

```
http.csrf(csrfConfig -> csrfConfig.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))
```

19.8. Unless needed the spring security will not generate the csrf token. To read it manually add below filter within the filter package

```
public class CsrfCookieFilter extends OncePerRequestFilter {

  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
                                                                                                    filterChain)
        throws ServletException, IOException {
    CsrfToken csrfToken = (CsrfToken) request.getAttribute(CsrfToken.class.getName());
    // Render the token value to a cookie by causing the deferred token to be loaded
    csrfToken.getToken();
    filterChain.doFilter(request, response);
  }
}
```

19.9. Add below in Security Config class for the filter to be invoked

```
http.addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
```

19.10. The above changes will work for the token generation first time the use logs in. For subsequent validations, we need to create object of CsrfTokenRequestAttributeHandler as follows to handle the token validations:

```
CsrfTokenRequestAttributeHandler csrfTokenRequestAttributeHandler = new
CsrfTokenRequestAttributeHandler();
```

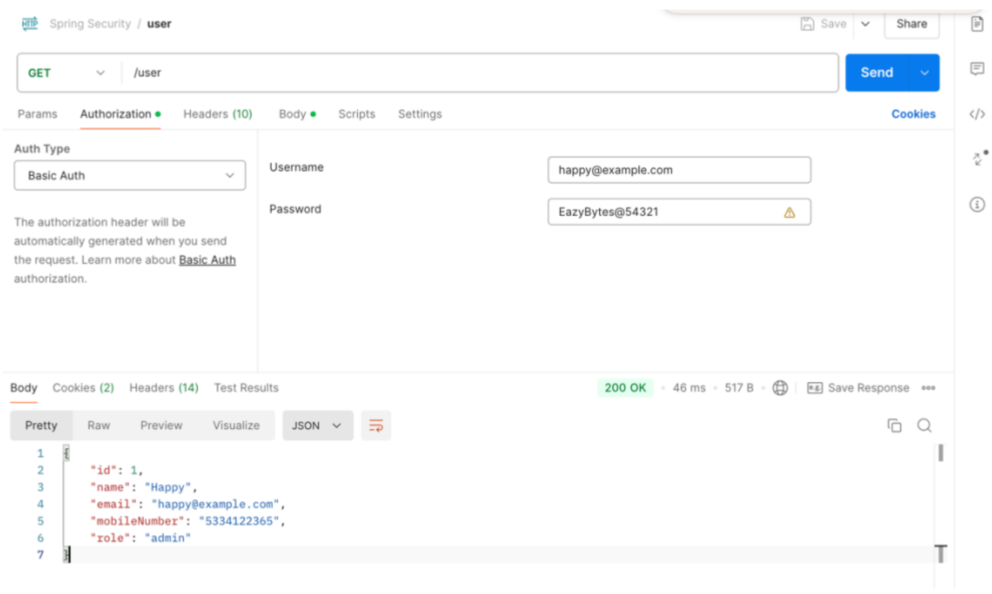
19.11. Need to inform spring security about this handler. Add below:

```
http.csrf(csrfConfig -> csrfConfig.csrfTokenRequestHandler(csrfTokenRequestAttributeHandler)
.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))
```

19.12. Since all the request will be now from front end application and it be of type http basic. So spring security needs to be informed about session management and security context. Update the code as follow:

```
http.securityContext(securityContext -> securityContext.requireExplicitSave(false))
  .sessionManagement(sessionConfig-> sessionConfig.sessionCreationPolicy(SessionCreationPolicy.ALWAYS))
```

19.13. To test the above changes, open postman and execute the GET request for /user as follows and should see xsrf-token in the response:



GET /user

Auth Type: Basic Auth

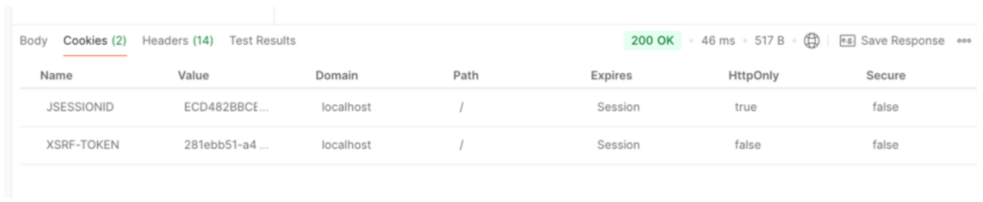
The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username: happy@example.com

Password: EazyBytes@54321

200 OK · 46 ms · 517 B

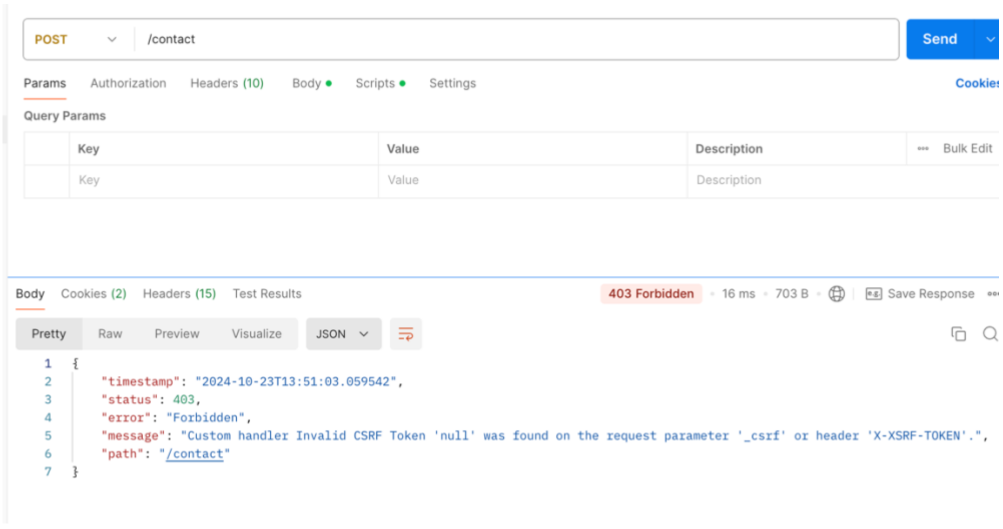
```
{
    "id": 1,
    "name": "Happy",
    "email": "happy@example.com",
    "mobileNumber": "5334122365",
    "role": "admin"
}
```



200 OK · 46 ms · 517 B

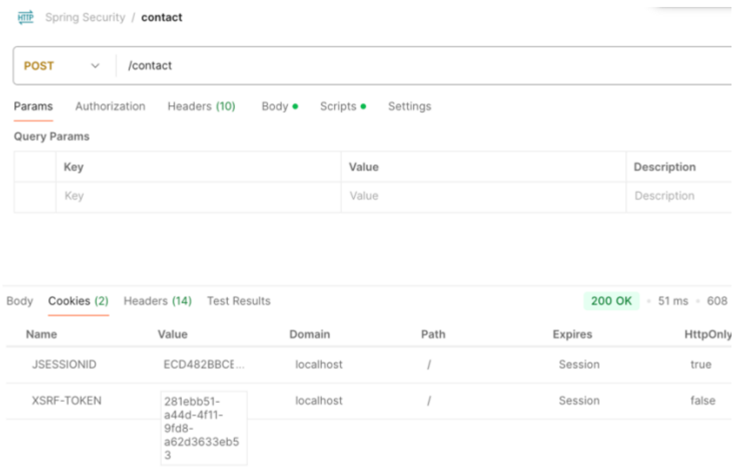
| Name | Value | Domain | Path | Expires | HttpOnly | Secure |
|---|---|---|---|---|---|---|
| JSESSIONID | ECD482BBCE... | localhost | / | Session | true | false |
| XSRF-TOKEN | 281ebb51-a4 ... | localhost | / | Session | false | false |

Now send POST request to /contact which is public url , should get an error as follows:



```
{
    "timestamp": "2024-10-23T13:51:03.059542",
    "status": 403,
    "error": "Forbidden",
    "message": "Custom handler Invalid CSRF Token 'null' was found on the request parameter '_csrf' or header 'X-XSRF-TOKEN'.",
    "path": "/contact"
}
```

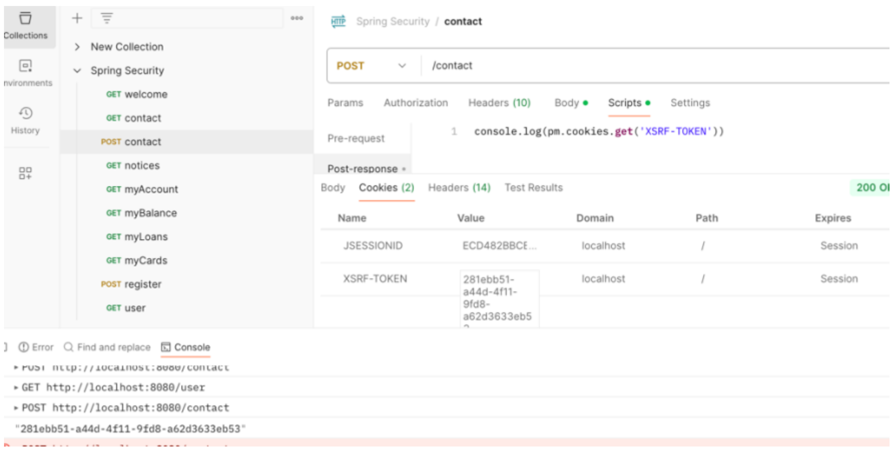
And click on Cookies tab , you should see the cookie value sent by the spring



| Name | Value | Domain | Path | Expires | HttpOnly |
|---|---|---|---|---|---|
| JSESSIONID | ECD482BBCE... | localhost | / | Session | true |
| XSRF-TOKEN | 281ebb51-a44d-4f11-9fd8-a62d3633eb53 | localhost | / | Session | false |

Resend by sending the X-XSRF-TOKEN as a part of header and the value can be copied from the previous screen and now resending the request should work successfully.

**NOTE: if unable to copy the value, use the following and resend the request. From console tab you should be able to copy**



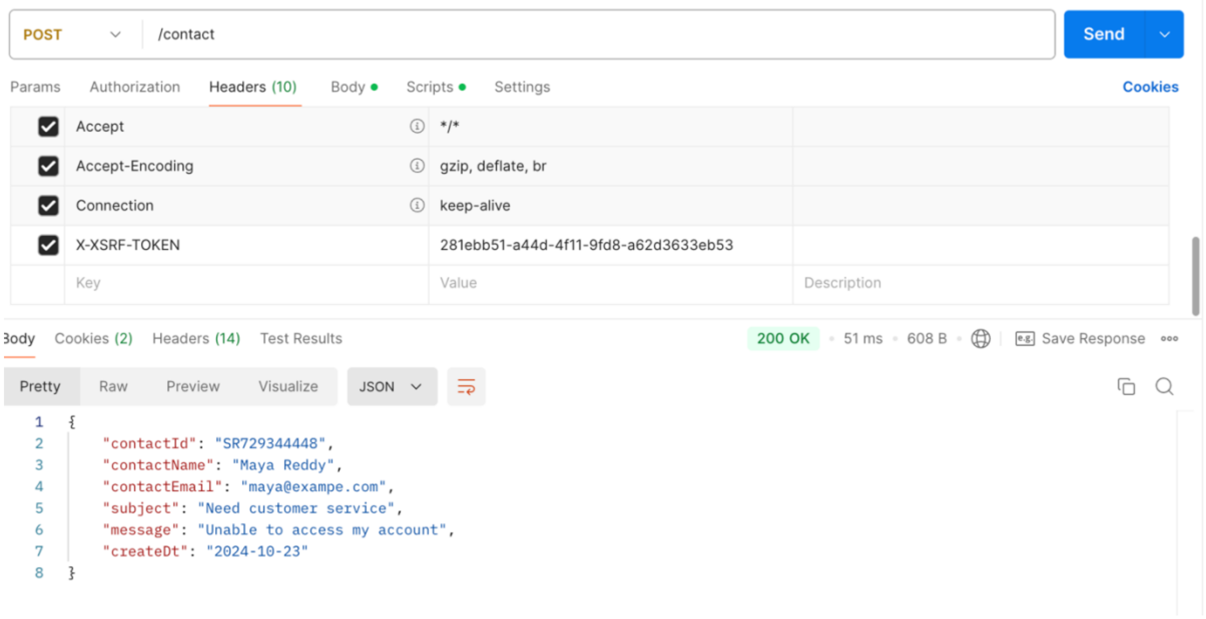
```
console.log(pm.cookies.get('XSRF-TOKEN'))
```

| Name | Value | Domain | Path | Expires |
|---|---|---|---|---|
| JSESSIONID | ECD482BBCE... | localhost | / | Session |
| XSRF-TOKEN | 281ebb51-a44d-4f11-9fd8-a62d3633eb5 | localhost | / | Session |

```
▸ POST http://localhost:8080/contact
▸ GET http://localhost:8080/user
▸ POST http://localhost:8080/contact
"281ebb51-a44d-4f11-9fd8-a62d3633eb53"
```

To ignore CSRF token for public URI add the below:

csrf(csrfConfig -> csrfConfig.csrfTokenRequestHandler(csrfTokenRequestAttributeHandler)
    **.ignoringRequestMatchers( "/contact","/register")**
    .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))



| | Key | Value | Description |
|---|---|---|---|
| ✓ | Accept | */* | |
| ✓ | Accept-Encoding | gzip, deflate, br | |
| ✓ | Connection | keep-alive | |
| ✓ | X-XSRF-TOKEN | 281ebb51-a44d-4f11-9fd8-a62d3633eb53 | |

```
{
    "contactId": "SR729344448",
    "contactName": "Maya Reddy",
    "contactEmail": "maya@exampe.com",
    "subject": "Need customer service",
    "message": "Unable to access my account",
    "createDt": "2024-10-23"
}
```

To test the same on front end application, try to send contact details, spring should throw 403 error since we did. Not logged in.

After logging in and then sending contact details will work. This is just dummy implementation of csrf token but to showcase that form submission needs token

## 20. Authorization using Authorities

20.1. Authorities are stored as GrantedAuthority and SimpleGrantedAuthority is implementation of the same. To provide authority based access to the users, create table authorities, model and update customer class. Below is the code for the same

```
CREATE TABLE `authorities` (
                    `id` int NOT NULL AUTO_INCREMENT,
                    `customer_id` int NOT NULL,
                    `name` varchar(50) NOT NULL,
                    PRIMARY KEY (`id`), KEY `customer_id` (`customer_id`),
 CONSTRAINT `authorities_ibfk_1` FOREIGN KEY (`customer_id`) REFERENCES `customer` (`customer_id`)
 );
```

```
INSERT INTO `authorities` (`customer_id`, `name`)
VALUES (1, 'VIEWACCOUNT');

INSERT INTO `authorities` (`customer_id`, `name`)
VALUES (1, 'VIEWCARDS');

INSERT INTO `authorities` (`customer_id`, `name`)
VALUES (1, 'VIEWLOANS');

INSERT INTO `authorities` (`customer_id`, `name`)
VALUES (1, 'VIEWBALANCE');

Entity
@Getter
@Setter
@Table(name="authorities")
public class Authority {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;

   private String name;

   @ManyToOne
   @JoinColumn(name="customer_id")
   private Customer customer;
}

@Entity
@Table(name = "customer")
@Getter @Setter
public class Customer {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "customer_id")
   private long id;

   private String name;

   private String email;

   @Column(name = "mobile_number")
   private String mobileNumber;

   @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
   private String pwd;

   private String role;

   @Column(name = "create_dt")
   @JsonIgnore
   private Date createDt;

   @OneToMany(mappedBy = "customer", fetch = FetchType.EAGER)
   @JsonIgnore
   private Set<Authority> authorities;

}
```

20.2. Update the UserDetails Service code as follows:

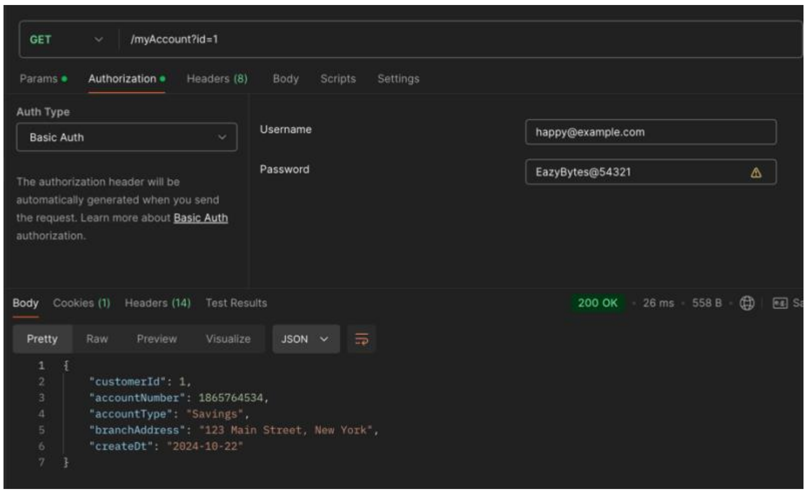
```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      Customer customer = customerRepository.findByEmail(username).orElseThrow(() -> new
            UsernameNotFoundException("User details not found for the user: " + username));
```

```
        List<GrantedAuthority> authorities = customer.getAuthorities()
                .stream()
                .map(authority -> new SimpleGrantedAuthority(authority.getName()))
                .collect(Collectors.toList());
    return new User(customer.getEmail(), customer.getPwd(), authorities);
  }
```

20.3. Update Security Config to add authority based access

```
.authorizeHttpRequests((requests) -> requests
        .requestMatchers("/myAccount").hasAuthority("VIEWACCOUNT")
        .requestMatchers("/myBalance").hasAnyAuthority("VIEWBALANCE", "VIEWACCOUNT")
        .requestMatchers("/myLoans").hasAuthority("VIEWLOANS")
        .requestMatchers("/myCards").hasAuthority("VIEWCARDS")
        .requestMatchers("/user").authenticated()
```

20.4. Now from postman if you try to access with happy@example.com and EazyBytes@54321 hitting the url http://localhost:8080/myAccount?id=1 should get the response



GET /myAccount?id=1

Params ● Authorization ● Headers (8) Body Scripts Settings

Auth Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username: happy@example.com

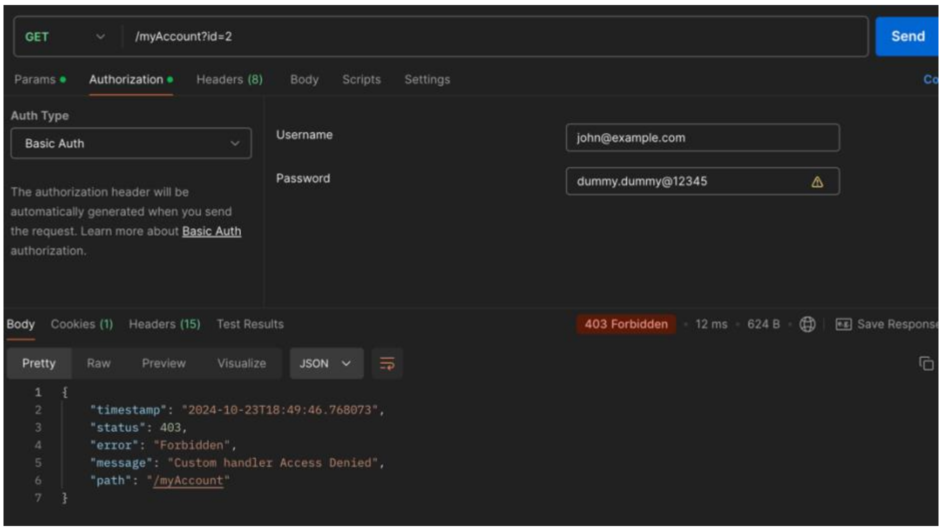
Password: EazyBytes@54321

Body Cookies (1) Headers (14) Test Results — 200 OK · 26 ms · 558 B

Pretty Raw Preview Visualize JSON

```
{
    "customerId": 1,
    "accountNumber": 1865764534,
    "accountType": "Savings",
    "branchAddress": "123 Main Street, New York",
    "createDt": "2024-10-22"
}
```

20.5. With john credentials it throws an error



GET /myAccount?id=2 Send

Params ● Authorization ● Headers (8) Body Scripts Settings

Auth Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username: john@example.com

Password: dummy.dummy@12345

Body Cookies (1) Headers (15) Test Results — 403 Forbidden · 12 ms · 624 B · Save Response

Pretty Raw Preview Visualize JSON

```
{
    "timestamp": "2024-10-23T18:49:46.768073",
    "status": 403,
    "error": "Forbidden",
    "message": "Custom handler Access Denied",
    "path": "/myAccount"
}
```

## 21. Authorization using Roles

21.1. Authority is fine grained applicable to single user and role based is coarse grained applicable to group. Spring boot provides creating roles with ROLE_ as prefix followed by role name.

21.2. Delete the entries in authorities table and add below entries:

```
DELETE FROM `authorities`;

INSERT INTO `authorities` (`customer_id`, `name`)
VALUES (1, 'ROLE_USER');

INSERT INTO `authorities` (`customer_id`, `name`)
VALUES (1, 'ROLE_ADMIN');
```

21.3. Update Security Config class as follows: DO NOT PREFIX ROLE_ for hasRole method as spring appends prefix

```
.requestMatchers("/myAccount").hasRole("USER")
        .requestMatchers("/myBalance").hasAnyRole("USER", "ADMIN")
        .requestMatchers("/myLoans").hasRole("USER")
        .requestMatchers("/myCards").hasRole("USER")
        .requestMatchers("/user").authenticated()
```

21.4. With these changes restart the server and try to access endpoint with customer having a role and with a customer not having a role.

## 22. Custom Filters

22.1. To see the list of filters invoked add below annotation on the class with the main method as follows:

```
@SpringBootApplication
@EnableWebSecurity(debug = true)
```

22.2. Restart the server and you should see the list of filter chain and all the sensitive information on the console.
DO NOT USE THIS FEATURE IN PRODUCTION

22.3. We can also create custom filters as follows:

```
 package com.security.demo.SpringSecurityDemoLatest.filter;

@Slf4j
public class AuthoritiesLoggingAfterFilter implements Filter {
  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
ServletException {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if(null != authentication) {
      log.info("User " + authentication.getName() + " is successfully authenticated and "
          + "has the authorities " + authentication.getAuthorities().toString());
    }
    chain.doFilter(request,response);
  }
}

package com.security.demo.SpringSecurityDemoLatest.filter;

@Slf4j
public class AuthoritiesLoggingAtFilter implements Filter {
  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    log.info("Authentication Validation is in progress");
    chain.doFilter(request,response);
  }
}
```

```
package com.security.demo.SpringSecurityDemoLatest.filter;

public class RequestValidationBeforeFilter implements Filter {


    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        String header = req.getHeader(HttpHeaders.AUTHORIZATION);
        if(null != header) {
            header = header.trim();
            if(StringUtils.startsWithIgnoreCase(header, "Basic ")) {
                byte[] base64Token = header.substring(6).getBytes(StandardCharsets.UTF_8);
                byte[] decoded;
                try {
                    decoded = Base64.getDecoder().decode(base64Token);
                    String token = new String(decoded, StandardCharsets.UTF_8); // un:pwd
                    int delim = token.indexOf(":");
                    if(delim== -1) {
                        throw new BadCredentialsException("Invalid basic authentication token");
                    }
                    String email = token.substring(0,delim);
                    if(email.toLowerCase().contains("test")) {
                        res.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                        return;
                    }
                } catch (IllegalArgumentException exception) {
                    throw new BadCredentialsException("Failed to decode basic authentication token");
                }
            }
        }
        chain.doFilter(request, response);
    }
}
```

22.4. Inform spring security about these custom filters and when do you want them to be invoked. Update the Security config class as follows:

```
http..addFilterBefore(new RequestValidationBeforeFilter(), BasicAuthenticationFilter.class)
        .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
        .addFilterAt(new AuthoritiesLoggingAtFilter(), BasicAuthenticationFilter.class)
```

## 23. JWT Based Authentication

23.1. Add below dependency:

```
 <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-api</artifactId>
            <version>0.12.5</version>
</dependency>
<dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-impl</artifactId>
            <version>0.12.5</version>
            <scope>runtime</scope>
</dependency>
<dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-jackson</artifactId>
            <version>0.12.5</version>
            <scope>runtime</scope>
</dependency>
```

**23.2.** To tell spring to not generate JSESSIONID and to go with JWT token format change the ALWAYS session creation policy to STATELESS so that no token is stored either on client or backend side. Backendwill validate token by calculating hash value .

```
.sessionManagement(sessionConfig->
sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
```

**23.3.** Remove security context since we are not generating JSESSIONID and add below in cors configuration to be able to expose header information  that will be sent in the response

```
config.setExposedHeaders(List.of("Authorization"));
```

23.4. JWT token should be generated after the initial login is completed. So create clases as follows:

```
package com.security.demo.SpringSecurityDemoLatest.constants;

public final class ApplicationConstants {

   public static final String JWT_SECRET_KEY = "JWT_SECRET";
   public static final String JWT_SECRET_DEFAULT_VALUE =
"jxgEQeXHuPq8VdbyYFNkANdudQ53YUn4";
   public static final String JWT_HEADER = "Authorization";
}

 @Service
@RequiredArgsConstructor
public class GenerateToken {

   private final Environment env;

   public  String createToken(Authentication authentication) {
      String secret = env.getProperty(ApplicationConstants.JWT_SECRET_KEY,
            ApplicationConstants.JWT_SECRET_DEFAULT_VALUE);
      SecretKey secretKey = Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8));
      String jwt = Jwts.builder().issuer("Eazy Bank").subject("JWT Token")
            .claim("username", authentication.getName())
            .claim("authorities", authentication.getAuthorities().stream().map(
                  GrantedAuthority::getAuthority).collect(Collectors.joining(",")))
            .issuedAt(new java.util.Date())
            .expiration(new java.util.Date((new java.util.Date()).getTime() + 30000000))
            .signWith(secretKey).compact();
      return jwt;
   }
}


@Component
@Slf4j
public class JWTTokenGeneratorFilter extends OncePerRequestFilter {

   @Autowired
   private  GenerateToken generateToken;

   @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
        FilterChain filterChain) throws ServletException, IOException {
     Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
     if (null != authentication) {
        Environment env = getEnvironment();
        if (null != env) {
                   String jwt = generateToken.createToken(authentication);
                   response.setHeader(ApplicationConstants.JWT_HEADER, jwt);
        }
     }
     filterChain.doFilter(request, response);
  }
```

```
    @Override
    protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {
        return !request.getServletPath().equals("/user");
    }

}
```

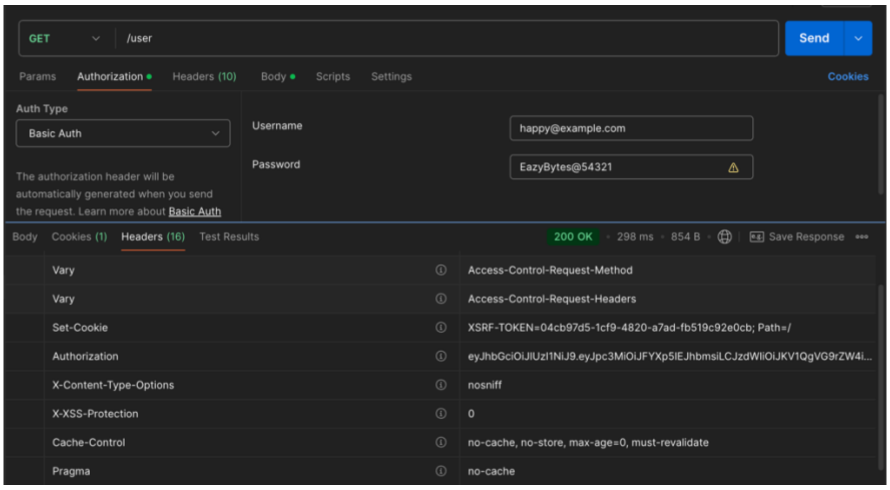
23.5. Validate JWT token for subsequent requests hence add below classes

```
public class JWTTokenValidatorFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain)
            throws ServletException, IOException {
        String jwt = request.getHeader(ApplicationConstants.JWT_HEADER);
        if(null != jwt) {
            try {
                Environment env = getEnvironment();
                if (null != env) {
                    String secret = env.getProperty(ApplicationConstants.JWT_SECRET_KEY,
                            ApplicationConstants.JWT_SECRET_DEFAULT_VALUE);
                    SecretKey secretKey = Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8));
                    if(null !=secretKey) {
                        Claims claims = Jwts.parser().verifyWith(secretKey)
                                .build().parseSignedClaims(jwt).getPayload();
                        String username = String.valueOf(claims.get("username"));
                        String authorities = String.valueOf(claims.get("authorities"));
                        Authentication authentication = new UsernamePasswordAuthenticationToken(username, null,
                                AuthorityUtils.commaSeparatedStringToAuthorityList(authorities));
                        SecurityContextHolder.getContext().setAuthentication(authentication);
                    }
                }

            } catch (Exception exception) {
                throw new BadCredentialsException("Invalid Token received!");
            }
        }
        filterChain.doFilter(request,response);
    }

    /**
     * thie filter should not be invoked while logging in
     * @param request current HTTP request
     * @return
     * @throws ServletException
     */
    @Override
    protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {
        return request.getServletPath().equals("/user");
    }

}
```

23.6. Update the security config to add these filters as follows:

```
@Autowired
 private JWTTokenGeneratorFilter jwtTokenGeneratorFilter;

            .addFilterAfter(jwtTokenGeneratorFilter, BasicAuthenticationFilter.class)

            .addFilterBefore(new JWTTokenValidatorFilter(), BasicAuthenticationFilter.class)
```

23.7. Now restart the application and test if jwt token is generated and validated: Send a GET request to /user as follows and should see the token as part of the header



GET /user

Params | Authorization | Headers (10) | Body | Scripts | Settings

Auth Type: Basic Auth

Username: happy@example.com

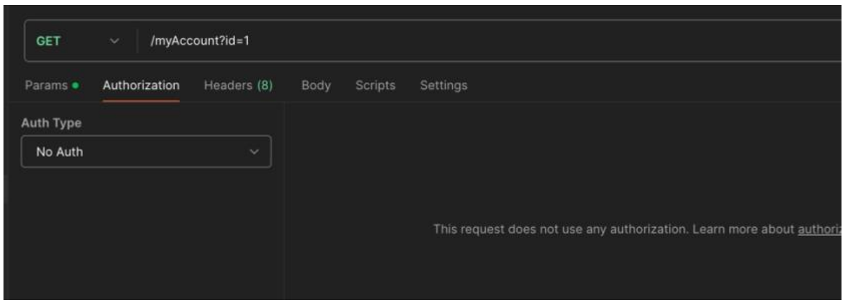
Password: EazyBytes@54321

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth

Body | Cookies (1) | Headers (16) | Test Results — 200 OK · 298 ms · 854 B

| | |
|---|---|
| Vary | Access-Control-Request-Method |
| Vary | Access-Control-Request-Headers |
| Set-Cookie | XSRF-TOKEN=04cb97d5-1cf9-4820-a7ad-fb519c92e0cb; Path=/ |
| Authorization | eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJFYXp5IEJhbmsiLCJzdWIiOiJKV1QgVG9rZW4i... |
| X-Content-Type-Options | nosniff |
| X-XSS-Protection | 0 |
| Cache-Control | no-cache, no-store, max-age=0, must-revalidate |
| Pragma | no-cache |

23.8. Once the token is generated copy the token from response header and try to access any authenticated with No Auth and passing the token as a part of the header as follows:

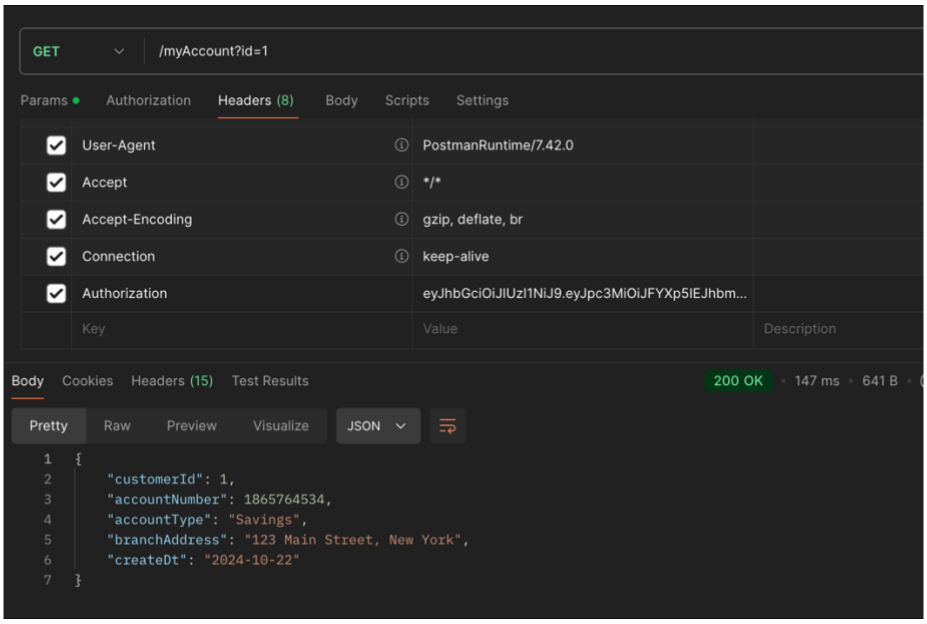


GET /myAccount?id=1

Params | Authorization | Headers (8) | Body | Scripts | Settings

Auth Type: No Auth

This request does not use any authorization. Learn more about authori



GET /myAccount?id=1

Params | Authorization | Headers (8) | Body | Scripts | Settings

| Key | Value | Description |
|---|---|---|
| ✓ User-Agent | PostmanRuntime/7.42.0 | |
| ✓ Accept | */* | |
| ✓ Accept-Encoding | gzip, deflate, br | |
| ✓ Connection | keep-alive | |
| ✓ Authorization | eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJFYXp5IEJhbm... | |

Body | Cookies | Headers (15) | Test Results — 200 OK · 147 ms · 641 B

Pretty | Raw | Preview | Visualize | JSON

```json
{
    "customerId": 1,
    "accountNumber": 1865764534,
    "accountType": "Savings",
    "branchAddress": "123 Main Street, New York",
    "createDt": "2024-10-22"
}
```

23.9. Lets see when token expiration happens then what happens: Modify the expiration time to 3 seconds

.expiration(new Date((new Date()).getTime() + 3000))

23.10. Now send request to /user get the token and after 3 seconds send a request to other secured url and you should see error

23.11. Till now we saw that we are sending credentials from http form or http basic. But in real time scenarios we may need a REST API that gets credentials as request parameter and not part of Basic Auth and to authenticate the users and send the jwt token as a response.

23.12. For this we need custom AuthenticationManager. Add below code in Security Config

```
@Bean
  public AuthenticationManager authenticationManager(UserDetailsService userDetailsService,
                                                      PasswordEncoder passwordEncoder) {
      CustomerUsernamePwdAuthenticationProvider authenticationProvider =
              new CustomerUsernamePwdAuthenticationProvider(userDetailsService, passwordEncoder);
      ProviderManager providerManager = new ProviderManager(authenticationProvider);
      providerManager.setEraseCredentialsAfterAuthentication(false);
      return  providerManager;
  }
```

OR

```
  @Bean
  public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws
Exception {
      return configuration.getAuthenticationManager();
  }
```

23.13. Add below POST API in UserControlller with dependencies as well:

```
private final AuthenticationManager authenticationManager;
 private final Environment env;
 private final GenerateToken generateToken;

@PostMapping("/apiLogin")
    public ResponseEntity<LoginResponseDTO> apiLogin (@RequestBody LoginRequestDTO loginRequest) {
        String jwt = "";
    Authentication authenticationResponse =
                authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(loginRequest.username(),loginRequest.password()));
        if(null != authenticationResponse && authenticationResponse.isAuthenticated()) {
            if (null != env) {
                jwt = generateToken.createToken(authenticationResponse);
            }
        }
        return ResponseEntity.status(HttpStatus.OK).header(ApplicationConstants.JWT_HEADER,jwt)
                .body(new LoginResponseDTO(HttpStatus.OK.getReasonPhrase(), jwt));
    }
```

**23.14. DO NOT FORGET TO ADD the above api in request matchers permit all as well as csrf ingnore matchers**

## 24. Method Level Security

24.1. To enable method level security that can be used even in a non-web application, add below annotation on the class with main method :

```
@EnableMethodSecurity
```

24.2. To apply invocation authorization to check if user can invoke a method or can return a response, spring security provides with pre and post authorize annotations. Update the LoanRepository method as follows:

```
public interface LoanRepository extends CrudRepository<Loans, Long> {

        @PreAuthorize("hasRole('USER')")
        List<Loans> findByCustomerIdOrderByStartDtDesc(long customerId);

}
```

Now run the application get jwt token by making a request to /user with valid credentials and then make a request to /myLoans should give the response. Now change the rolw in above annotation to the one that does not exist say MGR. Now repeating the steps \myLoans fails

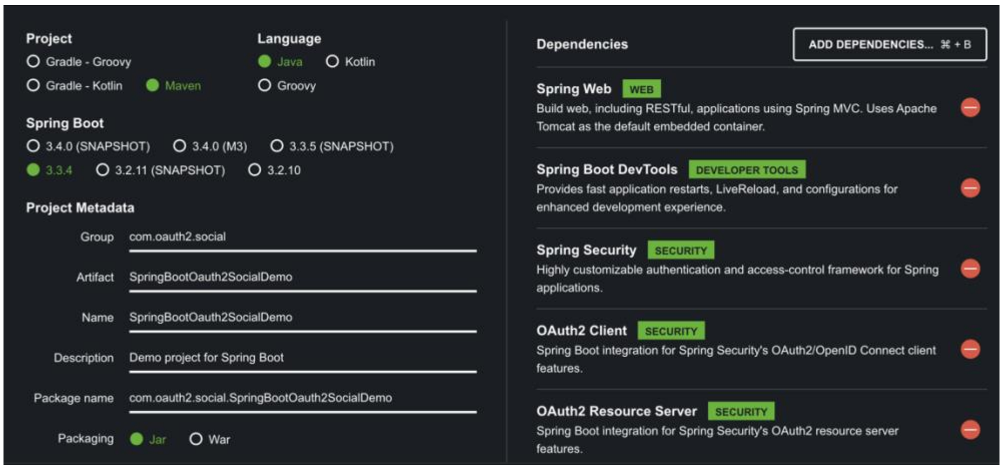
24.3. Likewise to test for post update the LoanController method as follows

```
@GetMapping("/myLoans")
  @PostAuthorize("hasRole('USER')")
  public List<Loans> getLoanDetails(@RequestParam long id) {
      List<Loans> loans = loanRepository.findByCustomerIdOrderByStartDtDesc(id);
      if (loans != null) {
          return loans;
      } else {
          return null;
      }
  }
```

With proper role it works if the role changed to the one logged in user does not has the response fails

## 25. OAUTH2 Social Media

25.1. Create a spring boot application as follows:



Project: Gradle - Groovy, Gradle - Kotlin, [✓ Maven]

Language: [✓ Java], Kotlin, Groovy

Spring Boot: 3.4.0 (SNAPSHOT), 3.4.0 (M3), 3.3.5 (SNAPSHOT), [✓ 3.3.4], 3.2.11 (SNAPSHOT), 3.2.10

Project Metadata

| | |
|---|---|
| Group | com.oauth2.social |
| Artifact | SpringBootOauth2SocialDemo |
| Name | SpringBootOauth2SocialDemo |
| Description | Demo project for Spring Boot |
| Package name | com.oauth2.social.SpringBootOauth2SocialDemo |
| Packaging | [✓ Jar] War |

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Security SECURITY
Highly customizable authentication and access-control framework for Spring applications.

OAuth2 Client SECURITY
Spring Boot integration for Spring Security's OAuth2/OpenID Connect client features.

OAuth2 Resource Server SECURITY
Spring Boot integration for Spring Security's OAuth2 resource server features.

25.2. Create a controller as follows:

```
@Controller
public class SecureController {

    @GetMapping("/secure")
    public String securePage(Authentication authentication) {
        if(authentication instanceof UsernamePasswordAuthenticationToken
              usernamePasswordAuthenticationToken){
            System.out.println(usernamePasswordAuthenticationToken);
        } else if (authentication instanceof OAuth2AuthenticationToken oAuth2AuthenticationToken) {
            System.out.println(oAuth2AuthenticationToken);
        }
        return "secure.html";
    }
}
```

25.3. Create a folder static within resources folder and create  a secure.html file as follows:

```
<!DOCTYPE html>
```

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Sample OAuth2 Client App</title>
</head>
<body>
<h1>Hey, I am secured page !!!!</h1>
</body>
</html>
```

25.4. Add below in properties file:

```
spring.application.name=springsecOAUTH2
spring.security.user.name=${SECURITY_USERNAME:eazybytes}
spring.security.user.password=${SECURITY_PASSWORD:12345}
logging.level.org.springframework.security=${SPRING_SECURITY_LOG_LEVEL:TRACE}
logging.pattern.console = ${LOGPATTERN_CONSOLE:%green(%d{HH:mm:ss.SSS}) %blue(%-5level)
%red([%thread]) %yellow(%logger{15}) - %msg%n}
```

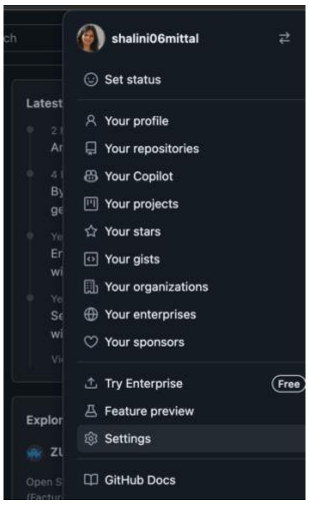
25.5. Run the application to test the html page is displayed after entering the credentials on the spring security login page.

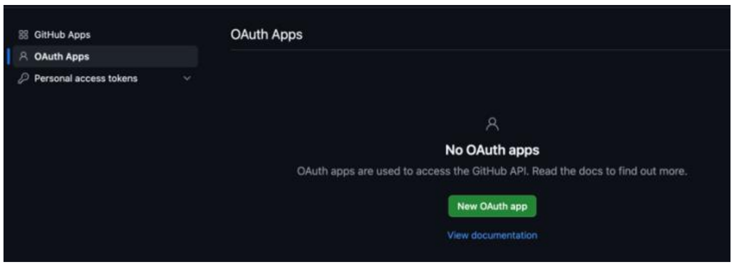
25.6. Now create a ProjectSecurityConfig class as follows to allow users to login via github or facebook:

```java
@Configuration
public class ProjectSecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeHttpRequests((requests) -> requests.requestMatchers("/secure").authenticated()
                .anyRequest().permitAll())
                .formLogin(Customizer.withDefaults())
                .oauth2Login(Customizer.withDefaults());
        return httpSecurity.build();
    }

    @Bean
    ClientRegistrationRepository clientRegistrationRepository() {
        ClientRegistration github = githubClientRegistration();
        ClientRegistration facebook = facebookClientRegistration();
        return new InMemoryClientRegistrationRepository(github, facebook);
    }

    private ClientRegistration githubClientRegistration() {
        return CommonOAuth2Provider.GITHUB.getBuilder("github").clientId("")
                .clientSecret("").build();
    }

    private ClientRegistration facebookClientRegistration() {
        return CommonOAuth2Provider.FACEBOOK.getBuilder("facebook").clientId("")
                .clientSecret("").build();
    }

}
```

25.7. Now we need to register our application as a client with github and facebook to allow the end user login with these social media links and get the client id and secret:

25.8. Go to github.com and login. After login on right side click on your profile menu -> Settings and click on Developer Setting on the left side menu. Click on Outh Apps and no apps are configured yet.

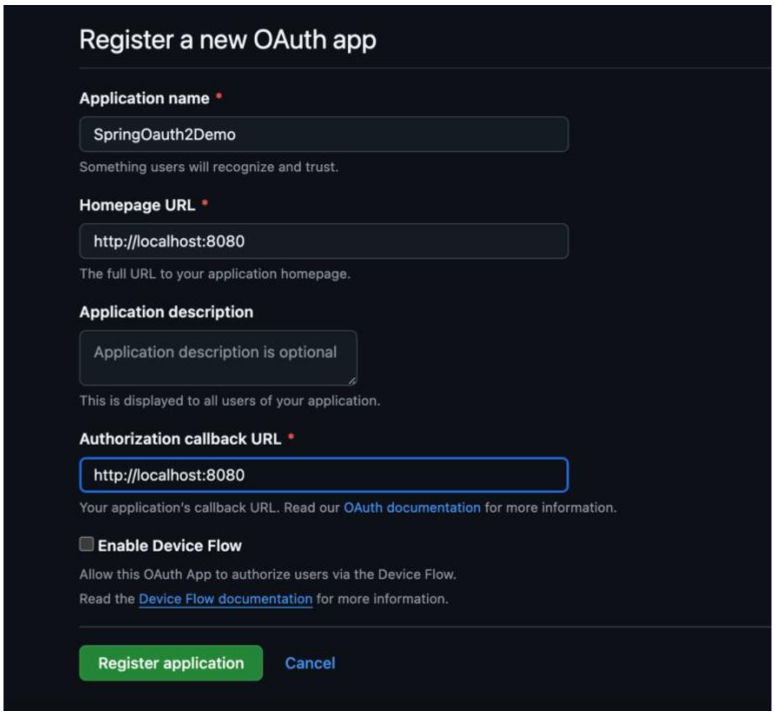




No OAuth apps

OAuth apps are used to access the GitHub API. Read the docs to find out more.

25.9. Click on Oauth App and enter the following:



Register a new OAuth app

Application name *

SpringOauth2Demo

Something users will recognize and trust.

Homepage URL *

http://localhost:8080

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

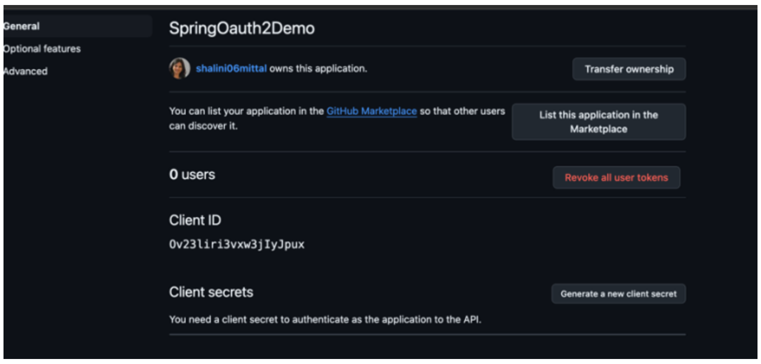
http://localhost:8080

Your application's callback URL. Read our OAuth documentation for more information.

☐ Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow.
Read the Device Flow documentation for more information.

25.10. Next clicking on registration gives the client id and for secret click on generate a new client secret follow the steps, copy it and paste in the secret.



SpringOauth2Demo

shalini06mittal owns this application.

You can list your application in the GitHub Marketplace so that other users can discover it.

0 users

Client ID

0v23liri3vxw3jIyJpux

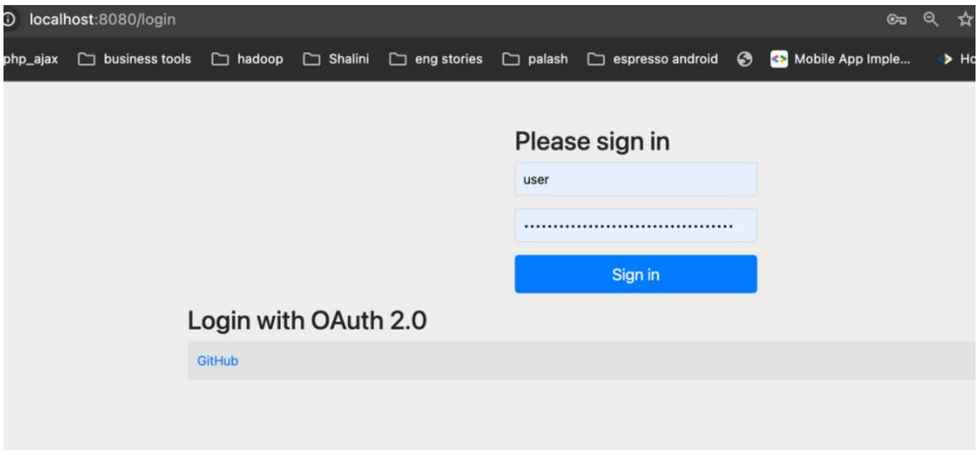
Client secrets

You need a client secret to authenticate as the application to the API.

25.11. Make sure to copy secret since once lost you cannot get it back and paste in your code

25.12. Likewise can do for facebook and other social media links.

25.13. Restart the server and should see normal spring login page as well as an option to login with git :



## 26. OAUTH2 Keycloak Server

26.1. Download keycloak or use docker installation from below:
[https://www.keycloak.org/guides](https://www.keycloak.org/guides)

26.2. Download and extract keycloak-26.0.2.zip from the Keycloak website. After extracting this file, you should have a directory that is named keycloak-26.0.2.

26.3. Start Keycloak

From a terminal, open the keycloak-26.0.2 directory.
Enter the following command:

On Linux, run:
bin/kc.sh start-dev
**bin/kc.sh start-dev --http-port=9090 [ to change default port 8080 ]**

On Windows, run:
bin\kc.bat start-dev

26.4. Using the start-dev option, you are starting Keycloak in development mode. In this mode, you can try out Keycloak for the first time to get it up and running quickly. This mode offers convenient defaults for developers, such as for developing a new Keycloak theme.

26.5. Create an admin user

26.5.1. Keycloak has no default admin user. You need to create an admin user before you can start Keycloak.
26.5.2. Open http://localhost:9090/.
26.5.3. Fill in the form with your preferred username and password.

26.6. Log in to the Admin Console

26.6.1. Go to the Keycloak Admin Console.
26.6.2. Log in with the username and password you created earlier.

26.7. Create a realm

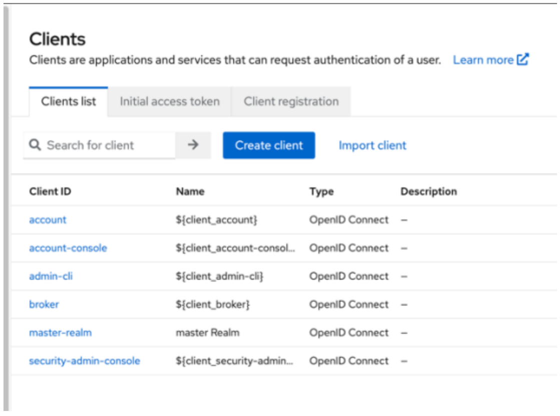
26.7.1. A realm in Keycloak is equivalent to a tenant. Each realm allows an administrator to create isolated groups of applications and users. Initially, Keycloak includes a single realm, called master. Use this realm only for managing Keycloak and not for managing any applications.
26.7.2. Use these steps to create the first realm.
26.7.3. Open the Keycloak Admin Console.
26.7.4. Click Keycloak next to master realm, then click Create Realm.
26.7.5. Enter oauthdemoapp in the Realm name field.
26.7.6. Click Create.

## 27. OAUTH2 Keycloak Server – Client credentials grant type

Secure the first application

27.1. To secure the first application, you start by registering the application with your Keycloak instance:
27.2. Open the Keycloak Admin page.
27.3. Click Clients.

27.4. Click Create client



Clients

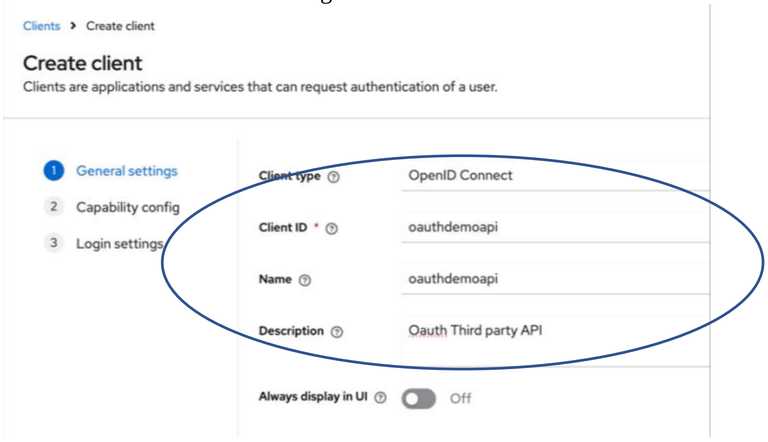
Clients are applications and services that can request authentication of a user. Learn more

Clients list | Initial access token | Client registration

Search for client | Create client | Import client

| Client ID | Name | Type | Description |
|---|---|---|---|
| account | ${client_account} | OpenID Connect | – |
| account-console | ${client_account-consol... | OpenID Connect | – |
| admin-cli | ${client_admin-cli} | OpenID Connect | – |
| broker | ${client_broker} | OpenID Connect | – |
| master-realm | master Realm | OpenID Connect | – |
| security-admin-console | ${client_security-admin... | OpenID Connect | – |

27.5. Fill in the form with the following values:



Clients > Create client

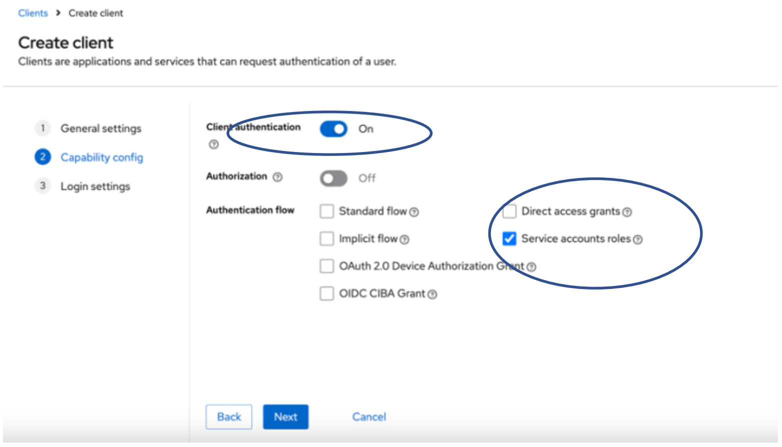
Create client

Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

| | |
|---|---|
| Client type | OpenID Connect |
| Client ID * | oauthdemoapi |
| Name | oauthdemoapi |
| Description | Oauth Third party API |
| Always display in UI | Off |

27.6. Click Next – select the following



Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

Client authentication: On

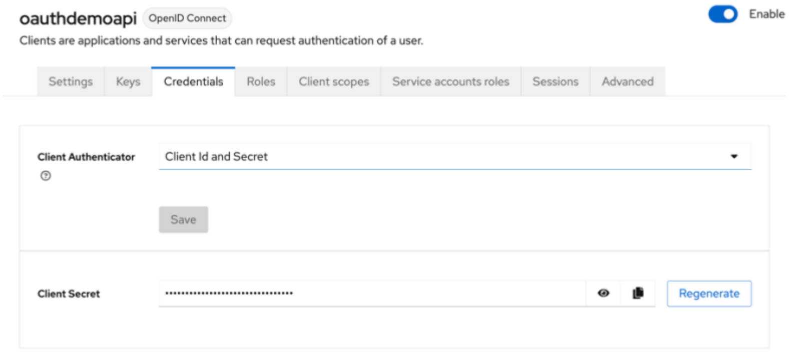
Authorization: Off

Authentication flow:
- [ ] Standard flow
- [ ] Implicit flow
- [ ] OAuth 2.0 Device Authorization Grant
- [ ] OIDC CIBA Grant
- [ ] Direct access grants
- [x] Service accounts roles

Back | Next | Cancel

27.7. Click Next and Save

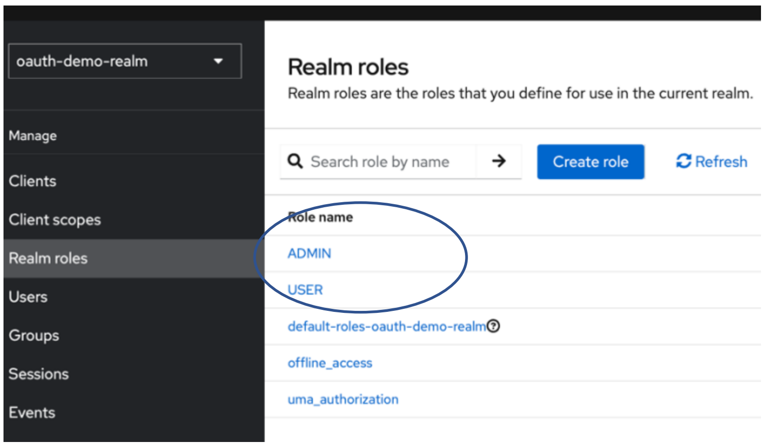
27.8. After save dashboard looks as follows: Click on Credentials tab for client secret


oauthdemoapi OpenID Connect
Clients are applications and services that can request authentication of a user.

Client Authenticator: Client Id and Secret

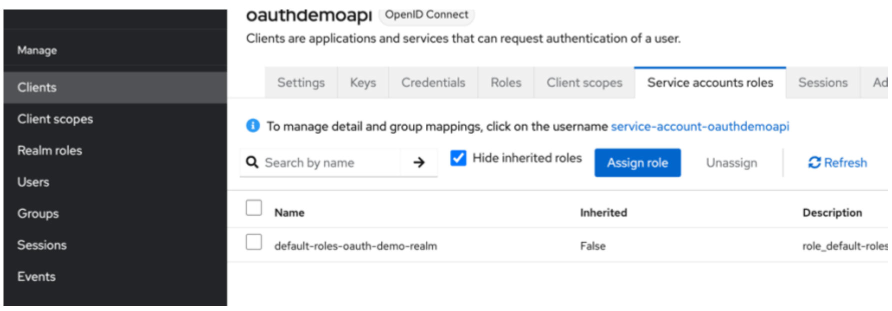
Client Secret: ••••••••••••••••••••••••••••••••

27.9. For creating roles, click on Realm Roles in the left pane, then click Create Role and add 2 roles : USER and ADMIN


Realm roles
Realm roles are the roles that you define for use in the current realm.

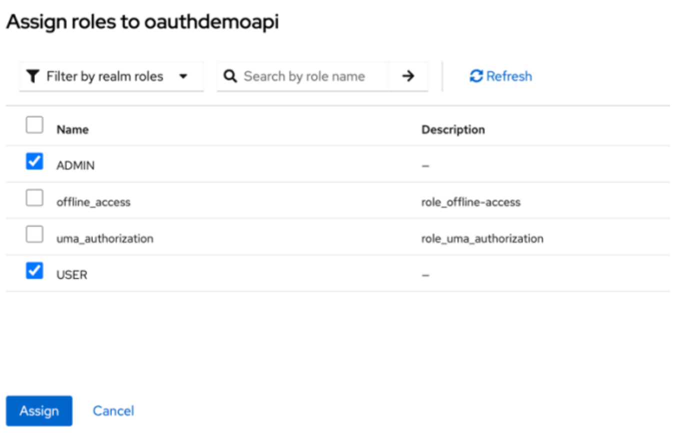
| Role name |
|---|
| ADMIN |
| USER |
| default-roles-oauth-demo-realm |
| offline_access |
| uma_authorization |

27.10. Next click on Clients -> Service Account Roles -> Assign Role -> Filter by realms and assign the above 2 roles created to this client


oauthdemoapi OpenID Connect
Clients are applications and services that can request authentication of a user.

To manage detail and group mappings, click on the username service-account-oauthdemoapi

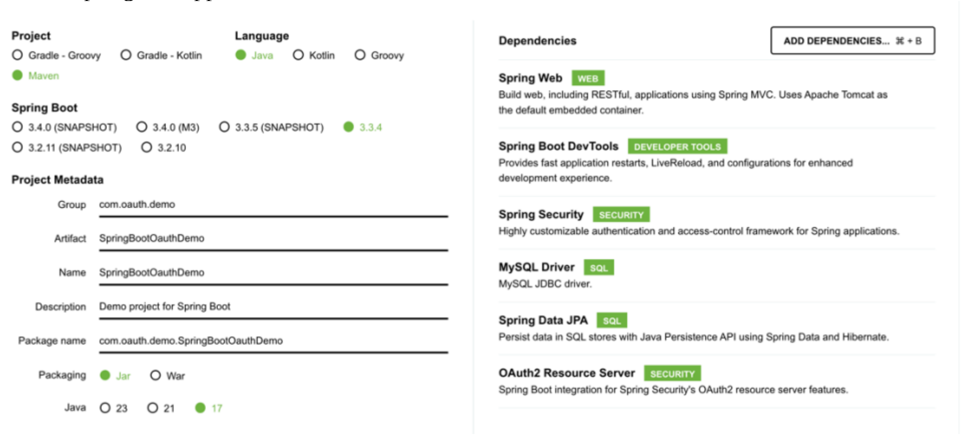
| Name | Inherited | Description |
|---|---|---|
| default-roles-oauth-demo-realm | False | role_default-roles |


Assign roles to oauthdemoapi

| | Name | Description |
|---|---|---|
| ✓ | ADMIN | – |
| | offline_access | role_offline-access |
| | uma_authorization | role_uma_authorization |
| ✓ | USER | – |

## 28. Spring Boot applicaiton

28.1. Create a springboot application as follows:



**Project**

○ Gradle - Groovy ○ Gradle - Kotlin [✓ Maven]

**Language**

[✓ Java] ○ Kotlin ○ Groovy

**Spring Boot**

○ 3.4.0 (SNAPSHOT) ○ 3.4.0 (M3) ○ 3.3.5 (SNAPSHOT) [✓ 3.3.4]
○ 3.2.11 (SNAPSHOT) ○ 3.2.10

**Project Metadata**

| | |
|---|---|
| Group | com.oauth.demo |
| Artifact | SpringBootOauthDemo |
| Name | SpringBootOauthDemo |
| Description | Demo project for Spring Boot |
| Package name | com.oauth.demo.SpringBootOauthDemo |
| Packaging | [✓ Jar] ○ War |
| Java | ○ 23 ○ 21 [✓ 17] |

**Dependencies**

ADD DEPENDENCIES... ⌘ + B

**Spring Web** WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot DevTools** DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Spring Security** SECURITY
Highly customizable authentication and access-control framework for Spring applications.

**MySQL Driver** SQL
MySQL JDBC driver.

**Spring Data JPA** SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**OAuth2 Resource Server** SECURITY
Spring Boot integration for Spring Security's OAuth2 resource server features.

28.2. Copy all the folders from previous spring boot project and exclude the following

28.2.1. BankUserDetailsService
28.2.2. CustomerProdUsernamePwdAuthenticationProvider
28.2.3. CustomerUsernamePwdAuthenticationProvider
28.2.4. ApplicationConstants
28.2.5. AuthoritiesLoggingAfterFilter
28.2.6. AuthoritiesLoggingAtFilter
28.2.7. JwtTokenGenerateFilter
28.2.8. JwtTokenValidationFilter
28.2.9. RequestValidationBeforeFilter
28.2.10. GenerateToken
28.2.11. Security Config class looks as below:

```java
@Configuration
public class ProjectSecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {

        CsrfTokenRequestAttributeHandler csrfTokenRequestAttributeHandler = new CsrfTokenRequestAttributeHandler();

            http.sessionManagement(sessionConfig-> sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

                http.cors(corsConfig -> corsConfig.configurationSource(new CorsConfigurationSource() {
            @Override
            public CorsConfiguration getCorsConfiguration(HttpServletRequest request) {
                CorsConfiguration config = new CorsConfiguration();
                config.setAllowedOrigins(Collections.singletonList("http://localhost:4200"));
                config.setAllowedMethods(Collections.singletonList("*"));
                config.setAllowCredentials(true);
                config.setAllowedHeaders(Collections.singletonList("*"));
                config.setExposedHeaders(List.of("Authorization"));
                config.setMaxAge(3600L);
                return config;
            }}))
               .requiresChannel(rcc-> rcc.anyRequest().requiresInsecure()) // HTTP
               .csrf(csrfConfig -> csrfConfig.csrfTokenRequestHandler(csrfTokenRequestAttributeHandler)
                       .ignoringRequestMatchers( "/contact","/register")
                       .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))
                        .addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
                        .authorizeHttpRequests((requests) -> requests
                    .requestMatchers("/myAccount").hasRole("USER")
                    .requestMatchers("/myBalance").hasAnyRole("USER", "ADMIN")
                    .requestMatchers("/myLoans").hasRole("USER")
                    .requestMatchers("/myCards").hasRole("USER")
                    .requestMatchers("/user").authenticated()
                .requestMatchers("/notices","/token", "/contact","/error","/register").permitAll());

                http.exceptionHandling(ehc -> ehc.accessDeniedHandler(new CustomAccessDeniedHandler()));
        return http.build();
    }
}
```

28.3. Create a class to convert the jwt token and get the information:

```
package com.oauth.demo.SpringBootOauthDemo.config;

import org.springframework.core.convert.converter.Converter;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.oauth2.jwt.Jwt;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class KeycloakRoleConverter implements Converter<Jwt, Collection<GrantedAuthority>> {
    /**
     * @param source the source object to convert, which must be an instance of {@code S} (never {@code null})
     * @return
     */
    @Override
    public Collection<GrantedAuthority> convert(Jwt source) {
        Map<String, Object> realmAccess = (Map<String, Object>) source.getClaims().get("realm_access");
        if (realmAccess == null || realmAccess.isEmpty()) {
            return new ArrayList<>();
        }
        Collection<GrantedAuthority> returnValue = ((List<String>) realmAccess.get("roles"))
                .stream().map(roleName -> "ROLE_" + roleName)
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toList());
        return returnValue;
    }
}
```

28.3.1. For an application using JWT as its main security mechanism, the authorization aspect consists of:

28.3.1.1. Extracting claim values from the JWT payload, usually the scope or scp claim

28.3.1.2. Mapping those claims into a set of GrantedAuthority objects

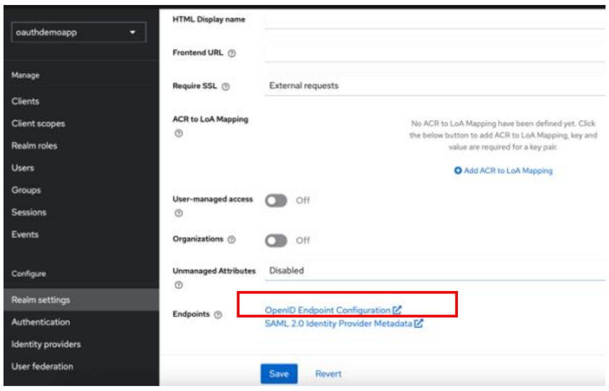
28.3.2. Once the security engine has set up those authorities, it can then evaluate whether any access restrictions apply to the current request and decide whether it can proceed.

JwtAuthenticationConverter: Converts a raw JWT into an AbstractAuthenticationToken
JwtGrantedAuthoritiesConverter: Extracts a collection of GrantedAuthority instances from the raw JWT.
Internally, JwtAuthenticationConverter uses JwtGrantedAuthoritiesConverter to populate a JwtAuthenticationToken with GrantedAuthority objects along with other attributes.

28.3.3. Add below in Security Config class defaultSecurityFilterChain() method:

```
JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();
jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(new KeycloakRoleConverter());

http.oauth2ResourceServer(rsc -> rsc.jwt(jwtConfigurer ->
            jwtConfigurer.jwtAuthenticationConverter(jwtAuthenticationConverter)));
```

28.4. Update properties files as follows: To get the jwt-uri, go to keycloack admin page -> click on your realm ->



HTML Display name

Frontend URL

Require SSL: External requests

ACR to LoA Mapping: No ACR to LoA Mapping have been defined yet. Click the below button to add ACR to LoA Mapping, key and value are required for a key pair.

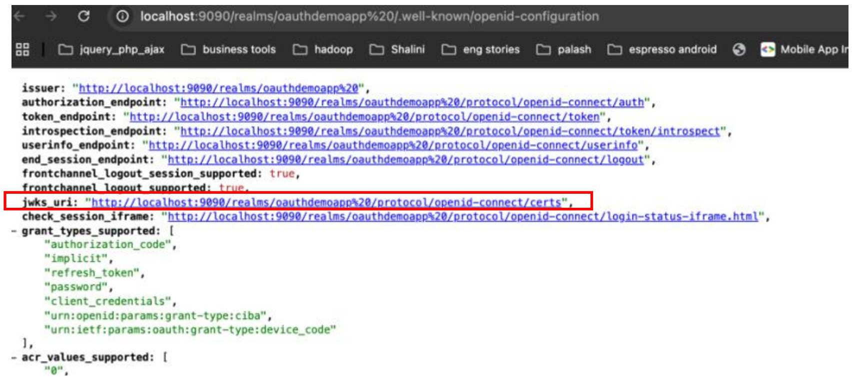
Add ACR to LoA Mapping

User-managed access: Off

Organizations: Off

Unmanaged Attributes: Disabled

Endpoints: OpenID Endpoint Configuration
SAML 2.0 Identity Provider Metadata

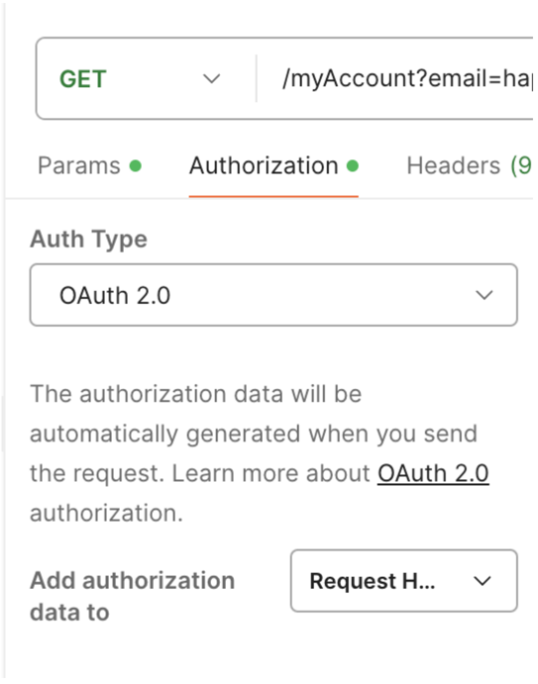


localhost:9090/realms/oauthdemoapp%20/.well-known/openid-configuration

```
  issuer: "http://localhost:9090/realms/oauthdemoapp%20",
  authorization_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/auth",
  token_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/token",
  introspection_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/token/introspect",
  userinfo_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/userinfo",
  end_session_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/logout",
  frontchannel_logout_session_supported: true,
  frontchannel_logout_supported: true,
  jwks_uri: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/certs",
  check_session_iframe: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/login-status-iframe.html",
- grant_types_supported: [
     "authorization_code",
     "implicit",
     "refresh_token",
     "password",
     "client_credentials",
     "urn:openid:params:grant-type:ciba",
     "urn:ietf:params:oauth:grant-type:device_code"
  ],
- acr_values_supported: [
     "0",
```

spring.security.oauth2.resourceserver.jwt.jwk-set-uri=${JWK_SET_URI:http://localhost:9090/realms/oauth-demo-realm/protocol/openid-connect/certs}

28.5. Update the controllers to take customer email as parameter and not customer id.
28.6. Postman acts as client. Spring boot application as resource server and keycloak as authorization server.
28.7. For postman to get the access token to access a resource from spring boot application needs to configure as follows: Make a GET request to http://localhost:8080/myAccount?email=happy@example.com
Select Oauth 2.0 for Authorization:

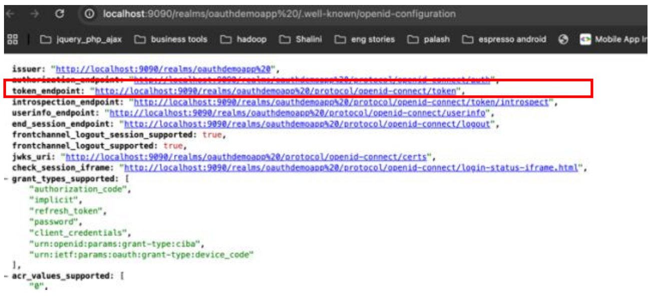


Auth Type: OAuth 2.0

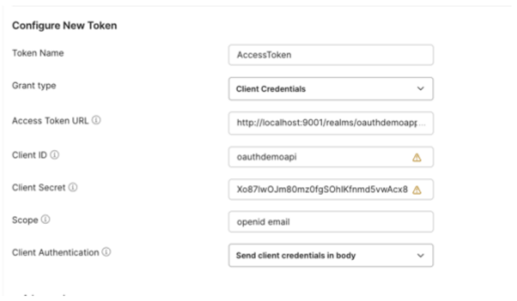
The authorization data will be automatically generated when you send the request. Learn more about OAuth 2.0 authorization.

Add authorization data to: Request H...

28.8. Scroll down and enter details as follows. Get access token url from keycloak server :



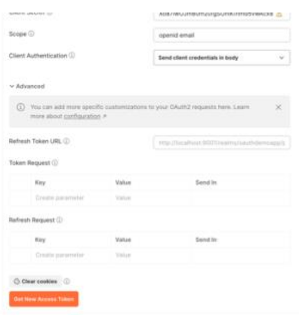
```
issuer: "http://localhost:9090/realms/oauthdemoapp%20",
authorization_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/auth",
token_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/token",
introspection_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/token/introspect",
userinfo_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/userinfo",
end_session_endpoint: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/logout",
frontchannel_logout_session_supported: true,
frontchannel_logout_supported: true,
jwks_uri: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/certs",
check_session_iframe: "http://localhost:9090/realms/oauthdemoapp%20/protocol/openid-connect/login-status-iframe.html",
- grant_types_supported: [
    "authorization_code",
    "implicit",
    "refresh_token",
    "password",
    "client_credentials",
    "urn:openid:params:grant-type:ciba",
    "urn:ietf:params:oauth:grant-type:device_code"
  ],
- acr_values_supported: [
    "0",
```



Configure New Token

| | |
|---|---|
| Token Name | AccessToken |
| Grant type | Client Credentials |
| Access Token URL | http://localhost:9001/realms/oauthdemoapp |
| Client ID | oauthdemoapi |
| Client Secret | Xo87lwOJm80mz0fgSOhlKfnmd5vwAcx8 |
| Scope | openid email |
| Client Authentication | Send client credentials in body |

Click on Get New Access Token



| | |
|---|---|
| Scope | openid email |
| Client Authentication | Send client credentials in body |

Advanced

You can add more specific customizations to your OAuth2 requests here. Learn more about configuration

Refresh Token URL

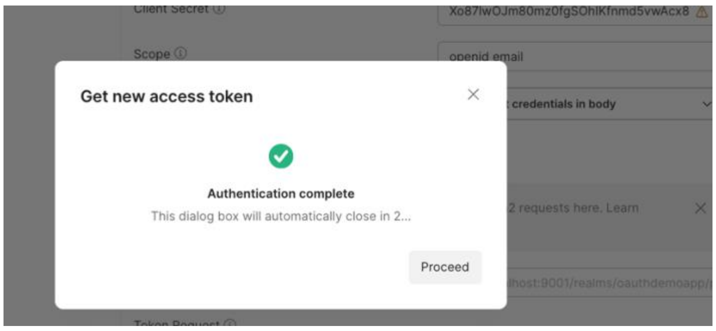
Token Request

| Key | Value | Send In |
|---|---|---|
| Create parameter | Value | |

Refresh Request

| Key | Value | Send In |
|---|---|---|
| Create parameter | Value | |

If successful click on Approved

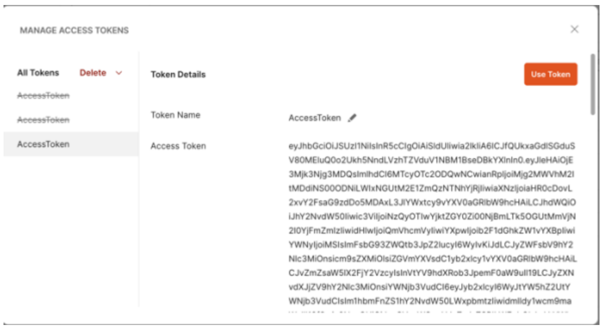


Get new access token

Authentication complete

This dialog box will automatically close in 2...

Then click on Use Token :

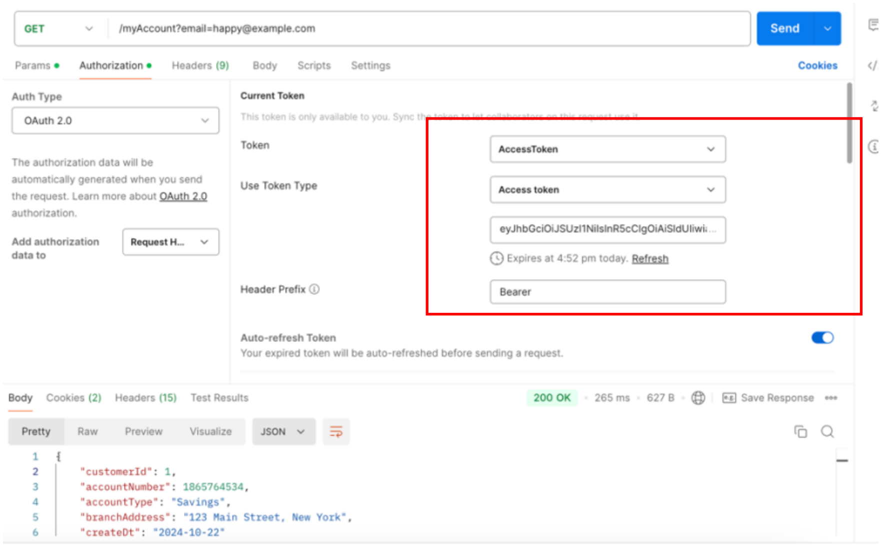


MANAGE ACCESS TOKENS

Token Details

| | |
|---|---|
| Token Name | AccessToken |
| Access Token | eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJfQUkxaGdlSGduS V80MEluQ0o2Ukh5NndLVzhTZVduV1NBM1BseDBkYXInIn0.eyJleHAiOjE 3Mjk3Njg3MDQsImlhdCI6MTcyOTc2ODQwNCwianRpIjoiMjg2MWVhM2I tMDdiNS00ODNiLWIxNGUtM2E1ZmQzNTNhYjRjIiwiaXNzIjoiaHR0cDovL 2xvY2FsaG9zdDo5MDAxL3JlYWxtcy9vYXV0aGRlbW9hcHAiLCJhdWQiO iJhY2NvdW50Iiwic3ViIjoiNzQyOTIwYjktZGY0Zi00NjBmLTk5OGUtMmVjN 2I0YjFmZmIzIiwidHlwIjoiQmVhcmVyIiwiYXpwIjoib2F1dGhkZW1vYXBpIiwi YWNyIjoiMSIsImFsbG93ZWQtb3JpZ2lucyI6WyIvKiJdLCJyZWFsbV9hY2 Nlc3MiOnsicm9sZXMiOlsiZGVmYXVsdC1yb2xlcy1vYXV0aGRlbW9hcHAiL CJvZmZsaW5lX2FjY2VzcyIsInVtYV9hdXRob3JpemF0aW9uIl19LCJyZXN vdXJjZV9hY2Nlc3MiOnsiYWNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtY WNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbmtzIiwidmlldy1wcm9ma |

Once clicked on Use Token it appears under current token as follows: Make sure for all the fields are same as below. Now the response should be sccessful



GET /myAccount?email=happy@example.com

Params • Authorization • Headers (9) Body Scripts Settings Cookies

Auth Type: OAuth 2.0

The authorization data will be automatically generated when you send the request. Learn more about OAuth 2.0 authorization.

Add authorization data to: Request H...

Current Token

This token is only available to you. Sync the token to let collaborators on this request use it.

Token: AccessToken

Use Token Type: Access token

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwi...

Expires at 4:52 pm today. Refresh

Header Prefix: Bearer

Auto-refresh Token
Your expired token will be auto-refreshed before sending a request.

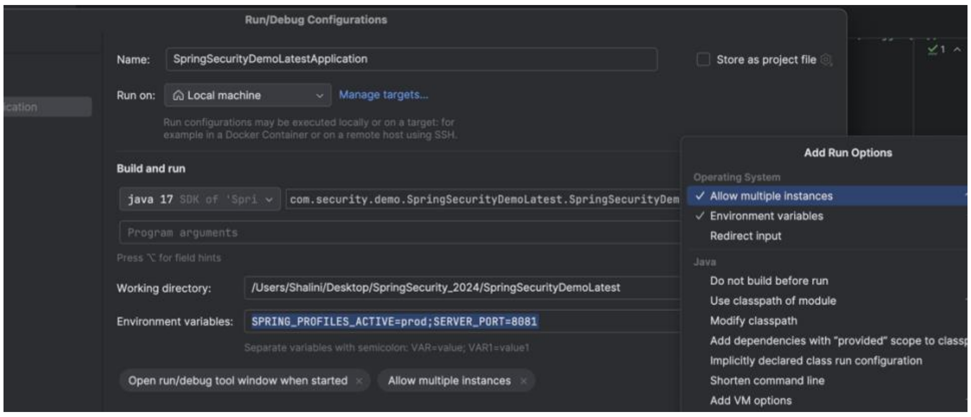
Body Cookies (2) Headers (15) Test Results — 200 OK · 265 ms · 627 B · Save Response

Pretty Raw Preview Visualize JSON

```
{
    "customerId": 1,
    "accountNumber": 1865764534,
    "accountType": "Savings",
    "branchAddress": "123 Main Street, New York",
    "createDt": "2024-10-22"
```

## 29. EXTRAS

29.1. https://github.com/shalini06mittal/SpringSecurity_2024 - All springboot security related projects
29.2. Install codeium for inteelij as AI plugin
29.3. To start multiple instance of a springboot application and provide environment variables



Run/Debug Configurations

Name: SpringSecurityDemoLatestApplication

Store as project file

Run on: Local machine — Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run

java 17 SDK of 'Spri — com.security.demo.SpringSecurityDemoLatest.SpringSecurityDem

Program arguments

Press ⌥ for field hints

Working directory: /Users/Shalini/Desktop/SpringSecurity_2024/SpringSecurityDemoLatest

Environment variables: SPRING_PROFILES_ACTIVE=prod;SERVER_PORT=8081

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started × | Allow multiple instances ×

Add Run Options

Operating System
- ✓ Allow multiple instances
- ✓ Environment variables
- Redirect input

Java
- Do not build before run
- Use classpath of module
- Modify classpath
- Add dependencies with "provided" scope to classp
- Implicitly declared class run configuration
- Shorten command line
- Add VM options

29.4. https://youtu.be/996OiexHze0 -> Understand OAuth
29.5. https://youtu.be/t9O99l4gjAc - spring security basics – oauth
29.6. https://youtu.be/mPPhcU7oWDU - spring boot microservice crash course
29.7. https://youtube.com/playlist?list=PLqq-6Pq4lTTYTEooakHchTGglSvkZAjnE – complete security playlist [ Kaushik – microservices playlist ]
29.8. https://www.oauth.com/playground/