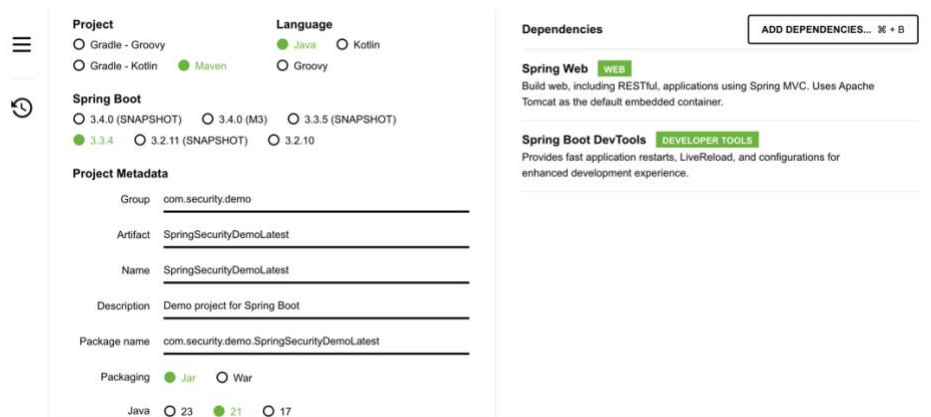


Table of Contents

1.	Spring Boot Maven Project.....	2
2.	Create a Rest Controller as follows:.....	2
3.	Add Security.....	2
4.	Add Custom credentials.....	3
5.	Prepare Controllers	3
6.	Security Config	5
7.	Disable form login and http basic auth	5
8.	POSTMAN	5
9.	Using In memory details Manager	5
10.	Using database to manage users.....	7
11.	Create custom table and user details manager	7
12.	Register customer	9
13.	Custom Authentication Provider	11
14.	Profiles.....	11
15.	Accept HTTPS Traffic	12
16.	Exception Handling in Spring Security	12
17.	Session timeout and invalid session	13
18.	CORS	14
19.	CSRF	15
20.	EXTRAS.....	16

1. Spring Boot Maven Project

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



- 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:

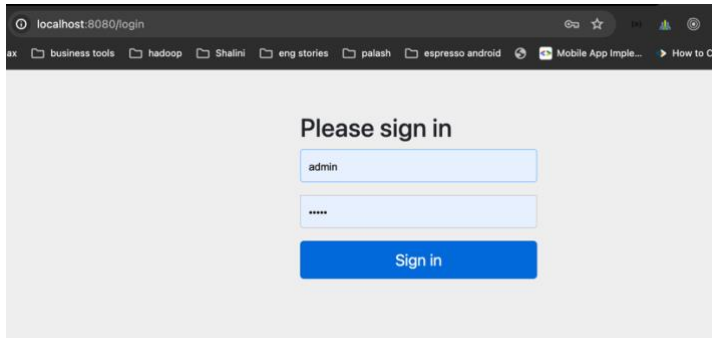
```
@RestController
public class WelcomeController {
    @GetMapping("/welcome")
    public String sayWelcome(){
        return "Welcome to spring application for security;
    }
}
```

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



- 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 : HH
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 : HH
2024-04-01T15:13:48.889+05:30 INFO 29438 --- [SpringBoot3SecurityDemo] [main] j.LocalContainerEntityManagerFactoryBean : In
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 prod
2024-04-01T15:13:49.137+05:30 INFO 29438 --- [SpringBoot3SecurityDemo] [main] o.s.s.web.DefaultSecurityFilterChain : Wi
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].[/] : Int
```

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";
    }
}
```

```

}

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());
        http.formLogin(withDefaults());
        http.httpBasic(withDefaults());
        return http.build();
    }
}

```

Line 1
Line 2
Line 3

- 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**:

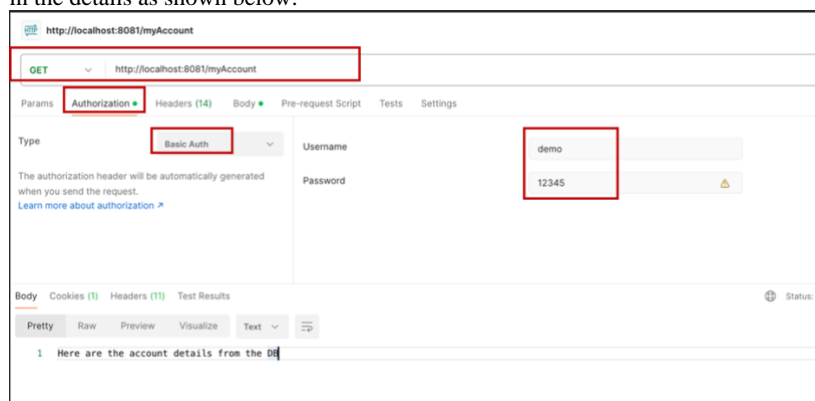
```
http.formLogin(AbstractHttpConfigurer::disable);
```

- 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:



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/>

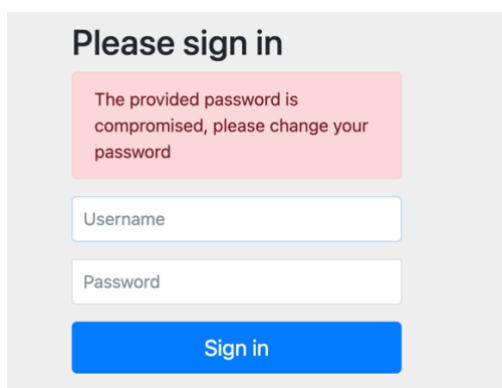
- 9.9. Update as below to use encrypted code instead of hard-coded passwords:

```
UserDetails admin = User.withUsername("admin")
    .password("{bcrypt}$2a$12$28hTdBcrUf7xzTwpPKvnLOIOcNqXaVsIu/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:



- 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$28hTdBcrUf7xzTwpPKvnLOIOcNqXaVsIu/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 BankUserDetailsService 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());
        }
    }
}

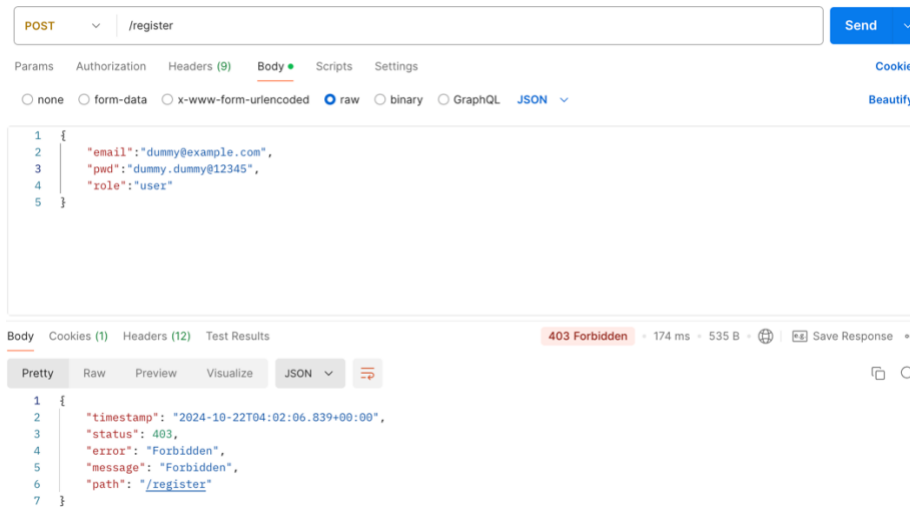
```

```
}
}
```

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

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

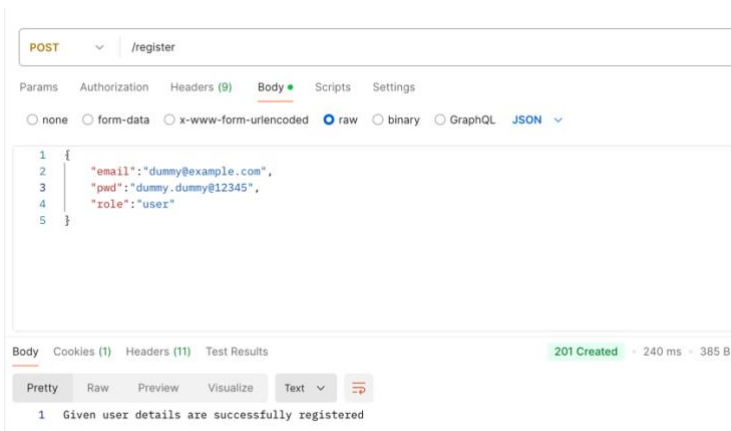
12.3. Now restart the server and send POST request as follows. You will get error



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.



☐ Show all

Number of rows: 25

Filter rows: Search this table

Sort by key: None

+ Options

↕

↕

↕

↕

		id	email	pwd	role
<input type="checkbox"/>	Edit Copy Delete	1	happy@example.com	{noop}shalini.techgatha@12345	read
<input type="checkbox"/>	Edit Copy Delete	2	admin@example.com	{{bcrypt}\$2a\$12\$28hTdBcrUf7xzTwpPKvnlOIocNqXaVslu/...	admin
<input type="checkbox"/>	Edit Copy Delete	3	dummy@example.com	{bcrypt}\$2a\$10\$1.3BTqVHKytMyhvs/sja9OUrg.IVQTJa4Fj...	user

↕

↕

↕

↕

Check all

With selected:

Edit Copy Delete Event

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}/${DATABASE_NAME:bank}
spring.datasource.username=${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

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 = "*")
```

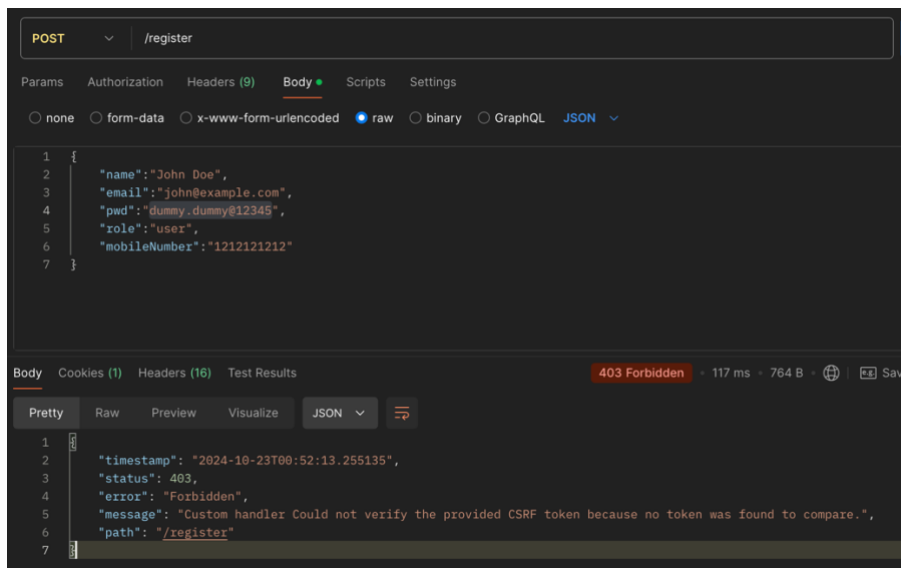
18.8. The drawback of above configuration is it needs to be added in all the controllers.

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:



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

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

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

19.4. 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);
    }
}
```

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

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

19.6. 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();
```

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

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

19.8. 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))
```

19.9. To test the above changes, open postman and execute the GET request for /user

20. EXTRAS

20.1. Install codeium for intellij