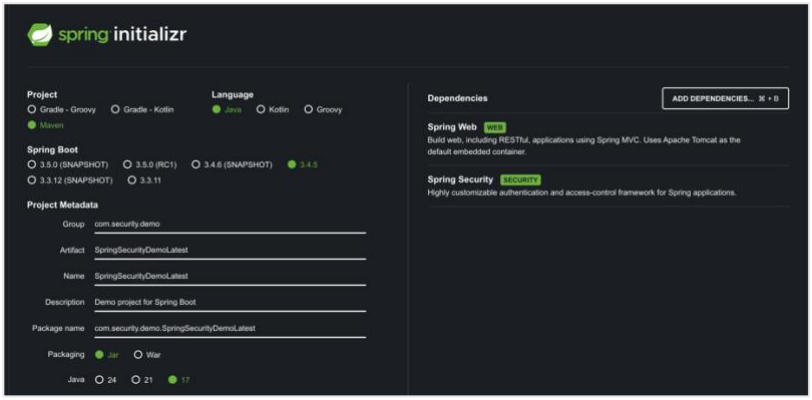
Table of Contents

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";
	}
}
```

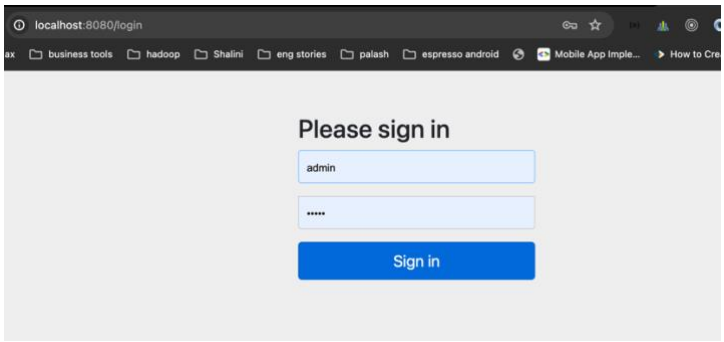
Start the application and open a browser. Pasting the below url should display the message returned by the method.
http://localhost:8080/welcome

3. Add Security

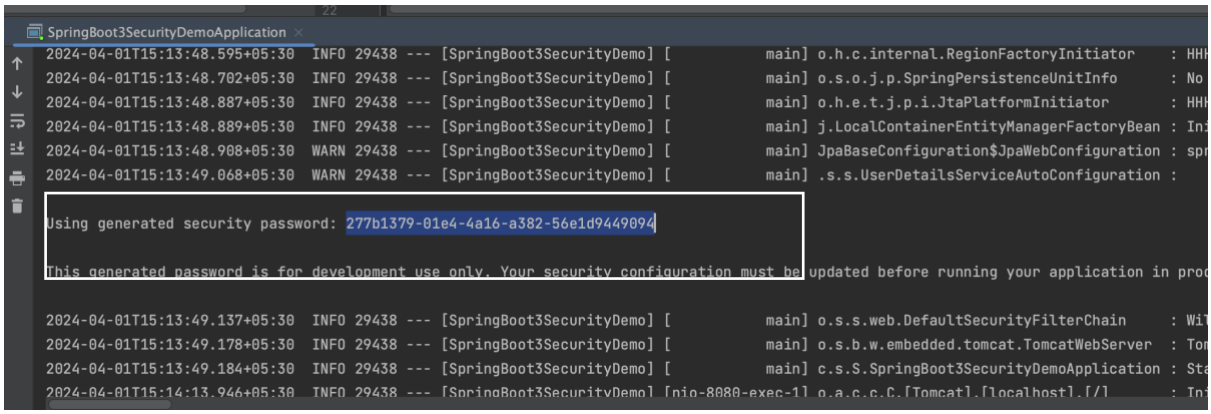
3.1. Add below dependency in pom.xml file:

```
<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:



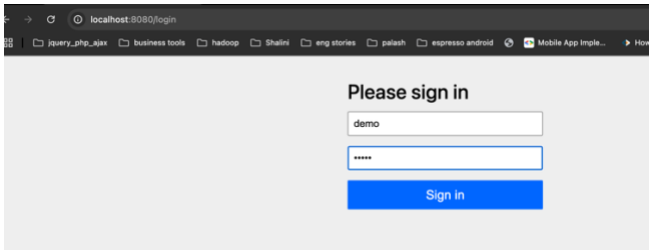
Type in user and paste the password , you should be able to login and test the urls
http://localhost:8080/welcome

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. Restart the server and enter the above credentials, should be able to login.

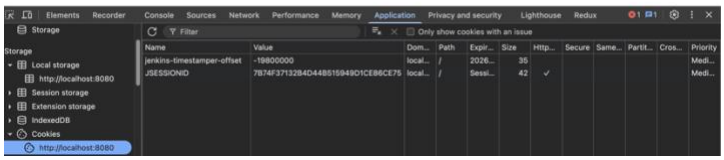


4.3. Add below property in properties file to understand the flow of spring security classes
logging.level.org.springframework.security=${SPRING_SECURITY_LOG_LEVEL:TRACE}

4.4. To logout:

http://localhost:8080/logout

4.5. 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()); //LINE1
      http.formLogin(withDefaults());                                               //LINE2
     // http.httpBasic(withDefaults());                                            //LINE3
      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", "/welcome").permitAll());
```

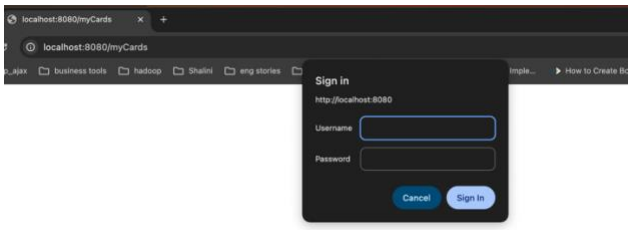
6.5. Above will allow access to routes with permit all but will have to login for other urls

7. Disable form login and http basic auth

7.1. To disable http form login add the below code by replacing **LINE2 and uncommenting LINE3**:

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

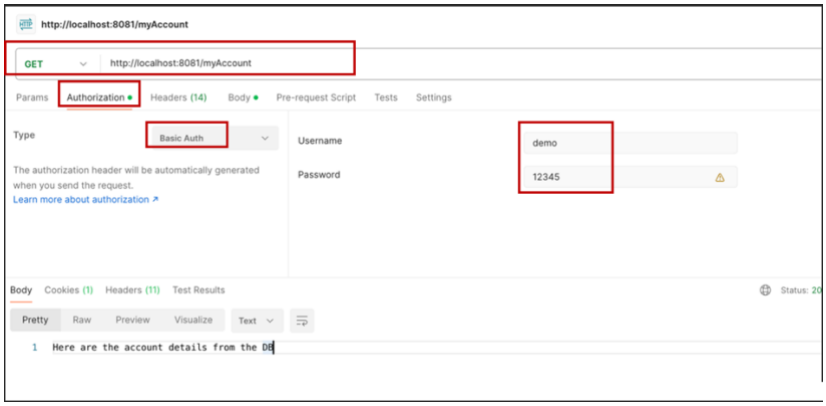
7.2. Now restart the server and you should see browser default http basic form and not the spring login page as follows: **[In case you do not get the below form try in incognito mode]**



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:

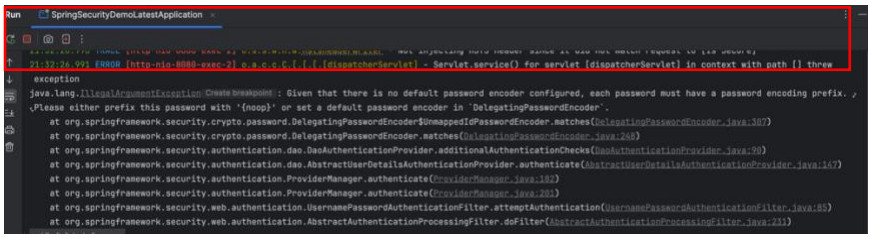


9. Using In memory details Manager

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 as follows:



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("admin")
        .password("{noop}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 as user.

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:

```
// Below is hash of 54321
UserDetails admin = User.withUsername("admin")

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

10. Create custom collection and user details manager

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

10.2. Add mongo db and Lombok dependencies.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>annotationProcessor</scope>
</dependency>
```

10.3. Add below in application.properties file:

```
spring.data.mongodb.uri=<your url>
spring.data.mongodb.database=securitydb
logging.level.org.springframework.data.mongodb.core.MongoTemplate=DEBUG

logging.level.org.mongodb.driver.protocol.command=DEBUG

spring.output.ansi.enabled=ALWAYS
```

10.4. Create Entities to map with the above table.

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

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Document
public class Customer {
   @Id
   private long id;
   private String email;
   private String pwd;
   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);

}
```

10.5. 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.entity.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);
   }
}
```

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

10.7. Add below method in the class with the main()method after the main() method to generate some dummy customers to test:

```
@Autowired
private CustomerRepository repository;
        @Autowired
        PasswordEncoder passwordEncoder;

        @PostConstruct
        public void initialize(){
                repository.save(new Customer(1, "user@a.com",passwordEncoder.encode("12345"),"user"));
                repository.save(new Customer(2,
"admin@a.com",passwordEncoder.encode("54321"),"admin"));
}
```

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

https://sanketdaru.com/blog/multiple-sources-user-details-spring-security/

https://stackoverflow.com/questions/76479689/how-to-use-multiple-userdetails-implementations-for-login-spring-security-and-s

10.9. We added roles to customer and to implement role-based authorization update the defaultSecurityFilterChain() method of ProjectSecurityConfig class as follows:

```
http.authorizeHttpRequests((requests) -> requests
 .requestMatchers("/myAccount", "/myBalance**").hasAuthority("user")**
 .requestMatchers( "/myLoans", "/myCards**").hasAuthority("admin**")
 .requestMatchers("/notices", "/contact", "/error","/welcome").permitAll()
 .anyRequest().authenticated());
  http.formLogin(withDefaults());
  return http.build();
```

11. Register customer

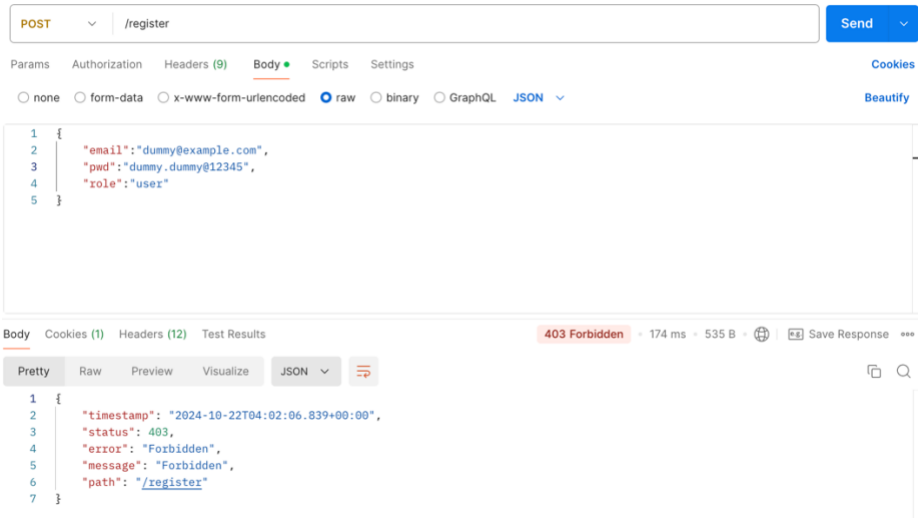
11.1. Create a rest controller with post mapping to add new customer

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

import com.security.demo.SpringSecurityDemoLatest.entity.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());
      }

   }

}
```

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

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

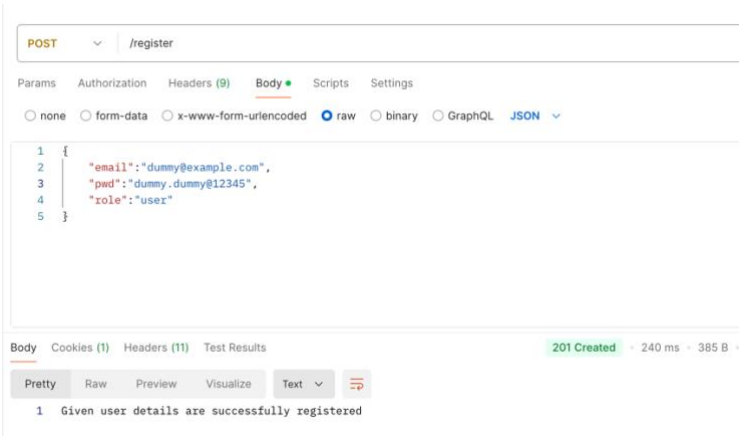
11.3. Now restart the server and send POST request as follows. You will get error



11.4. 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);
                    http.authorizeHttpRequests((requests) -> requests
            .requestMatchers("/myAccount", "/myBalance**").hasAuthority("user")**
            .requestMatchers( "/myLoans", "/myCards**").hasAuthority("admin**")
            .requestMatchers("/notices", "/contact", "/error","/welcome").permitAll()
            .anyRequest().authenticated());
             http.formLogin(withDefaults());
             return http.build();
  }
```

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



12. JWT Based Authentication

12.1. 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>
```

12.2. Add below records in entity package for login request and response:

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

public record LoginRequestDTO(String username, String password) {
}
```

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

public record LoginResponseDTO(String status, String token) {
}
```

12.3. 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 .

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

12.4. 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";
}
```

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

import com.security.demo.SpringSecurityDemoLatest.constants.ApplicationConstants;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import lombok.RequiredArgsConstructor;
import org.springframework.core.env.Environment;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Service;

import javax.crypto.SecretKey;
```

```
import java.nio.charset.StandardCharsets;
import java.util.stream.Collectors;
@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;
  }
}
```

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

import com.security.demo.SpringSecurityDemoLatest.constants.ApplicationConstants;
import com.security.demo.SpringSecurityDemoLatest.service.GenerateToken;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;
@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");
  }
```

```
}
```

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

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

import com.security.demo.SpringSecurityDemoLatest.constants.ApplicationConstants;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.core.env.Environment;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.crypto.SecretKey;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
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");
    }

}
```

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

```
@Autowired
 private JWTTokenGeneratorFilter jwtTokenGeneratorFilter;
```

12.7. Add below within defaultSecurityFilterChain() method of ProjectSecurityConfig class.

```
    http.addFilterAfter(jwtTokenGeneratorFilter, BasicAuthenticationFilter.class)
            .addFilterBefore(new JWTTokenValidatorFilter(), BasicAuthenticationFilter.class);
```

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

12.9. Lets create a REST API that gets credentials as request parameter and to authenticate the users and send the jwt token as a response.

12.10. For this we need custom AuthenticationManager. Add below code in ProjectSecurityConfig class

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

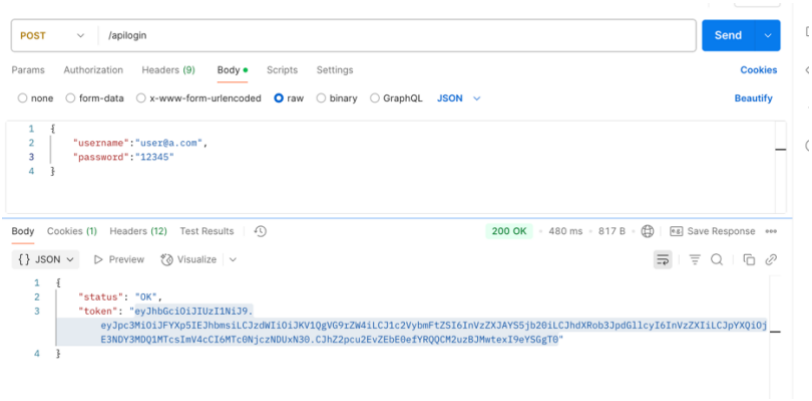
12.11. Add below POST API in UserControlller with dependencies as well:

```
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));
   }
```

12.12. **DO NOT FORGET TO ADD the above api in request matchers permit all as follows:**

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

12.13. Restart the server and open postman and send POST request to /apilogin as follows: Copy the token recieved



12.14. Now to try to access secured resource by passing jwt token as follows: