JWT Authentication authorization and Role based Authentication authorization.

**Spring Security Basics:** Spring security is a powerful framework that helps protect our web applications by managing authentication and authorization. It’s like a gatekeeper of our application ensuring only the right users get access to the right resources.

Authentication and Authorization:

Authentication is a process of verifying the identity of user, while authorization controls what a user is allowed to do once they’re authenticated.

Imagine trying to enter a top-secret facility: first, we should show our Id(authentication), and then, depending on our clearance level, we would be granted access to certain rooms or anything (authorization).

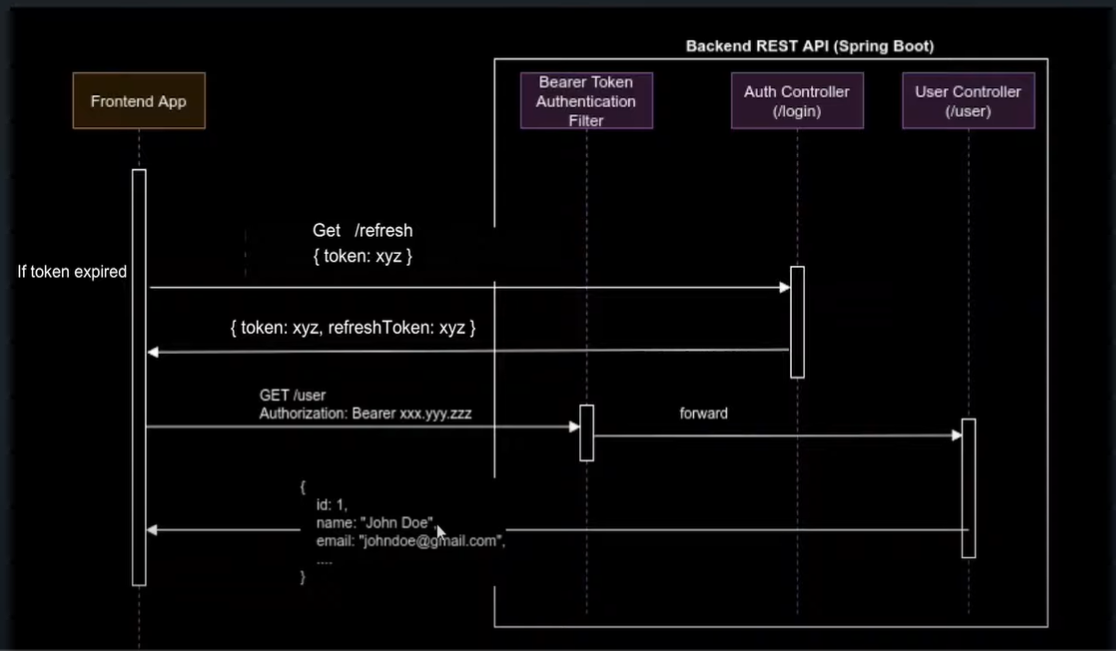
Role-Based Auth:

Moving on to the role-based authentication and authorization. Roles define a user’s group, like ‘user’ or ‘admin’  
with this approach, we can finally tune who can access what in our application.

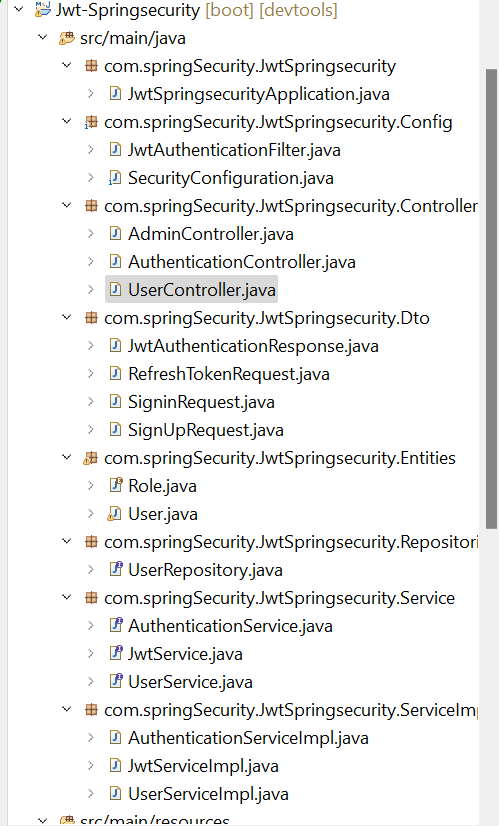
Refresh Tokens:

But what if the JWT expires? This is where refresh token comes into play. A refresh token is a longer-lived token that can be used to get a fresh JWT without requiring the user to login again. The minimizes the exposure time of the access token and enhance security.

Flow Diagram of working JWT Authentication and authorization:



The project folder looks like this



Step 1 – Creating a User Entity

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.Table;

**import** lombok.Data;

@Data

@Entity

@Table(name="user")

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String password;

**private** Role role;

}

Step 2 – Creating a Enum for Role:

**public** **enum** Role {

***USER***,

***ADMIN***

}

Step 3 – Configuring a application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/Jwt\_Springsecurity

spring.datasource.username=root

spring.datasource.password=Varun@123

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=false

spring.jpa.open-in-view=false

spring.jpa.properties.hibernate.format\_sql=true

Step 4 – Creating a UserRepository:

**import** java.util.Optional;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

**import** com.springSecurity.JwtSpringsecurity.Entities.Role;

**import** com.springSecurity.JwtSpringsecurity.Entities.User;

@Repository

**public** **interface** UserRepository **extends** JpaRepository<User, Long> {

Optional<User> findByEmail(String email);

User findByRole(Role role);

}

Step 5 – Adding a Spring security dependency:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

Step 6 – Adding JWT dependencies:

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

</dependency>

Step 7 – Implements a UserDetails for User Entity Class with UserDetails

**public** **class** User **implements** UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String password;

**private** Role role;

@Override

**public** Collection<? **extends** GrantedAuthority> getAuthorities() {

**return** List.*of*(**new** SimpleGrantedAuthority(role.name()));

}

@Override

**public** String getUsername() {

**return** email;

}

@Override

**public** **boolean** isAccountNonExpired() {

**return** **true**;

}

@Override

**public** **boolean** isAccountNonLocked() {

**return** **true**;

}

@Override

**public** **boolean** isCredentialsNonExpired() {

**return** **true**;

}

@Override

**public** **boolean** isEnabled() {

**return** **true**;

}

}

Step 8 – Creating a JwtService interface and JwtServiceImpl

JwtService:

**import** java.util.Map;

**import** org.springframework.security.core.userdetails.UserDetails;

**public** **interface** JwtService {

String extractUserName(String token);

String generateToken(UserDetails userDetails);

**boolean** isTokenValid(String token, UserDetails userDetails);

Object generateRefreshToken(Map<String, Object> extraClaims, UserDetails userDetails);

}

JwtServiceImpl: In this class we are returning all the methods that can generate the JWT or extract any information in our token

**import** java.security.Key;

**import** java.util.Date;

**import** java.util.Map;

**import** java.util.function.Function;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.stereotype.Service;

**import** com.springSecurity.JwtSpringsecurity.Service.JwtService;

**import** io.jsonwebtoken.Claims;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.SignatureAlgorithm;

**import** io.jsonwebtoken.io.Decoders;

**import** io.jsonwebtoken.security.Keys;

**import** lombok.RequiredArgsConstructor;

@Service

@RequiredArgsConstructor

**public** **class** JwtServiceImpl **implements** JwtService{

**public** String generateToken(UserDetails userDetails) {

**return** Jwts.*builder*().setSubject(userDetails.getUsername())

.setIssuedAt(**new** Date(System.*currentTimeMillis*()))

.setExpiration(**new** Date(System.*currentTimeMillis*() + 1000 \* 60 \* 24))

.signWith(getSigninKey(),SignatureAlgorithm.***HS256***)

.compact();

}

**public** String extractUserName(String token) {

**return** extractClaim(token, Claims::getSubject);

}

**private** <T> T extractClaim(String token, Function<Claims, T> claimsResolvers) {

**final** Claims claims = extractAllClaims(token);

**return** claimsResolvers.apply(claims);

}

**private** Key getSigninKey() {

**byte**[] key = Decoders.***BASE64***.decode("764398rt4gitrufhbjkvabfaeyiu4387orGETL8UI4GRT7");

**return** Keys.*hmacShaKeyFor*(key);

}

**private** Claims extractAllClaims(String token) {

**return** Jwts.*parserBuilder*().setSigningKey(getSigninKey()).build().parseClaimsJws(token).getBody();

}

**public** **boolean** isTokenValid(String token, UserDetails userDetails) {

**final** String username = extractUserName(token);

**return** (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

**private** **boolean** isTokenExpired(String token) {

**return** extractClaim(token, Claims::getExpiration).before(**new** Date());

}

@Override

**public** Object generateRefreshToken(Map<String, Object> extraClaims, UserDetails userDetails) {

**return** Jwts.*builder*().setClaims(extraClaims).setSubject(userDetails.getUsername())

.setIssuedAt(**new** Date(System.*currentTimeMillis*()))

.setExpiration(**new** Date(System.*currentTimeMillis*() + 604800000))

.signWith(getSigninKey(),SignatureAlgorithm.***HS256***)

.compact();

}

}

Step 9 – Creating JwtAuthenticationFilter and UserService in our project

JwtAuthenticationFilter – is the class where we will validate the Jwt for the every Apis

This dependency is used for lang3.StringUtils.*startsWith*(authHeader, "Bearer ") taking for this method

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

**import** java.io.IOException;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.context.SecurityContext;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.filter.OncePerRequestFilter;

**import** com.springSecurity.JwtSpringsecurity.Service.JwtService;

**import** com.springSecurity.JwtSpringsecurity.Service.UserService;

**import** io.micrometer.common.util.StringUtils;

**import** jakarta.servlet.FilterChain;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** lombok.RequiredArgsConstructor;

@Component

@RequiredArgsConstructor

**public** **class** JwtAuthenticationFilter **extends** OncePerRequestFilter {

**private** **final** JwtService jwtService;

**private** **final** UserService userService;

@Override

**protected** **void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

**throws** ServletException, IOException {

**final** String authHeader = request.getHeader("Authorization");

**final** String jwt;

**final** String userEmail;

**if**(StringUtils.*isEmpty*(authHeader) || !org.apache.commons.lang3.StringUtils.*startsWith*(authHeader, "Bearer ")) {

filterChain.doFilter(request, response);

**return**;

}

jwt = authHeader.substring(7);

userEmail = jwtService.extractUserName(jwt);

**if**(StringUtils.*isNotEmpty*(userEmail) && SecurityContextHolder.*getContext*().getAuthentication() == **null**) {

UserDetails userDetails = userService.userDetailsService().loadUserByUsername(userEmail);

**if**(jwtService.isTokenValid(jwt, userDetails)) {

SecurityContext securityContext = SecurityContextHolder.*createEmptyContext*();

UsernamePasswordAuthenticationToken token = **new** UsernamePasswordAuthenticationToken(

userDetails, **null**, userDetails.getAuthorities()

);

token.setDetails(**new** WebAuthenticationDetailsSource().buildDetails(request));

securityContext.setAuthentication(token);

SecurityContextHolder.*setContext*(securityContext);

}

}

filterChain.doFilter(request, response);

}

}

Step 10: Creating a UserService interface and UserServiceIImpl class

UserService:

**import** org.springframework.security.core.userdetails.UserDetailsService;

**public** **interface** UserService {

**public** UserDetailsService userDetailsService();

}

UserServiceIImpl:

**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** com.springSecurity.JwtSpringsecurity.Repositories.UserRepository;

**import** com.springSecurity.JwtSpringsecurity.Service.UserService;

**import** lombok.RequiredArgsConstructor;

@Service

@RequiredArgsConstructor

**public** **class** UserServiceImpl **implements** UserService{

**private** **final** UserRepository userRepository;

@Override

**public** UserDetailsService userDetailsService() {

**return** **new** UserDetailsService() {

@Override

**public** UserDetails loadUserByUsername(String username) {

**return** userRepository.findByEmail(username)

.orElseThrow(() -> **new** UsernameNotFoundException("User Not found"));

}

};

}

}

Step 11 – Creating a SecurityConfiguration class on config package

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.AuthenticationProvider;

**import** org.springframework.security.authentication.dao.DaoAuthenticationProvider;

**import** org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;

**import** org.springframework.security.config.http.SessionCreationPolicy;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** org.springframework.security.web.SecurityFilterChain;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

**import** com.springSecurity.JwtSpringsecurity.Entities.Role;

**import** com.springSecurity.JwtSpringsecurity.Service.UserService;

**import** lombok.RequiredArgsConstructor;

@Configuration

@EnableWebSecurity

@RequiredArgsConstructor

**public** **class** SecurityConfiguration {

**private** **final** JwtAuthenticationFilter jwtAuthenticationFilter;

**private** **final** UserService userService;

@Bean

**public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {

http.csrf(AbstractHttpConfigurer::disable)

.authorizeHttpRequests(request -> request.requestMatchers("/api/v1/auth/\*\*")

.permitAll()

.requestMatchers("/api/v1/admin").hasAuthority(Role.***ADMIN***.name())

.requestMatchers("/api/v1/user").hasAuthority(Role.***USER***.name())

.anyRequest().authenticated())

.sessionManagement(manager -> manager.sessionCreationPolicy(SessionCreationPolicy.***STATELESS***))

.authenticationProvider(authenticationProvider()).addFilterBefore(

jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.**class**

);

**return** http.build();

}

@Bean

**public** AuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authenticationProvider = **new** DaoAuthenticationProvider();

authenticationProvider.setUserDetailsService(userService.userDetailsService());

authenticationProvider.setPasswordEncoder(passwordEncoder());

**return** authenticationProvider;

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

@Bean

**public** AuthenticationManager authenticationManager(AuthenticationConfiguration config) **throws** Exception {

**return** config.getAuthenticationManager();

}

}

Step 12 – Creating our Api end points to controller layer

UserController, AdminController, AuthenticationController

AuthenticationController:

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

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

**import** com.springSecurity.JwtSpringsecurity.Dto.JwtAuthenticationResponse;

**import** com.springSecurity.JwtSpringsecurity.Dto.RefreshTokenRequest;

**import** com.springSecurity.JwtSpringsecurity.Dto.SignUpRequest;

**import** com.springSecurity.JwtSpringsecurity.Entities.User;

**import** com.springSecurity.JwtSpringsecurity.Service.AuthenticationService;

**import** lombok.RequiredArgsConstructor;

@RestController

@RequestMapping("/api/v1/auth")

@RequiredArgsConstructor

**public** **class** AuthenticationController {

**private** **final** AuthenticationService authenticationService;

@PostMapping("/signup")

**public** ResponseEntity<User> signUp(@RequestBody SignUpRequest signUpRequest) {

**return** ResponseEntity.*ok*(authenticationService.signUp(signUpRequest));

}

@PostMapping("/signin")

**public** ResponseEntity<JwtAuthenticationResponse> signin(@RequestBody SignUpRequest signinRequest) {

**return** ResponseEntity.*ok*(authenticationService.signin(signinRequest));

}

@PostMapping("/refresh")

**public** ResponseEntity<JwtAuthenticationResponse> refresh(@RequestBody RefreshTokenRequest refreshTokenRequest) {

**return** ResponseEntity.*ok*(authenticationService.refreshToken(refreshTokenRequest));

}

}

UserController:

**import** org.springframework.http.ResponseEntity;

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

**import** org.springframework.web.bind.annotation.RequestMapping;

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

**import** lombok.RequiredArgsConstructor;

@RestController

@RequestMapping("/api/v1/user")

@RequiredArgsConstructor

**public** **class** UserController {

@GetMapping

**public** ResponseEntity<String> sayHello(){

**return** ResponseEntity.*ok*("Hi User this is your page");

}

}

AdminController:

**import** org.springframework.http.ResponseEntity;

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

**import** org.springframework.web.bind.annotation.RequestMapping;

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

**import** lombok.RequiredArgsConstructor;

@RestController

@RequestMapping("/api/v1/admin")

@RequiredArgsConstructor

**public** **class** AdminController {

@GetMapping

**public** ResponseEntity<String> sayHello(){

**return** ResponseEntity.*ok*("Hi Admin this is your page");

}

}

Step 13 – We are creating a registered Apis

For the creating a registered Apis, we need to create a dto package and add it

SignUpRequest:

**import** lombok.Data;

@Data

**public** **class** SignUpRequest {

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String password;

}

SigninRequest:

**import** lombok.Data;

@Data

**public** **class** SigninRequest {

**private** String email;

**private** String password;

}

RefreshTokenRequest:

**import** lombok.Data;

@Data

**public** **class** RefreshTokenRequest {

**private** String token;

}

JwtAuthenticationResponse:

**import** lombok.Data;

@Data

**public** **class** JwtAuthenticationResponse {

**private** String token;

**private** Object refreshToken;

}

Step 14 – For this we are creating a AuthenticationService interface and AuthenticationServiceImpl class on Service package

AuthenticationService :

**import** com.springSecurity.JwtSpringsecurity.Dto.JwtAuthenticationResponse;

**import** com.springSecurity.JwtSpringsecurity.Dto.RefreshTokenRequest;

**import** com.springSecurity.JwtSpringsecurity.Dto.SignUpRequest;

**import** com.springSecurity.JwtSpringsecurity.Entities.User;

**public** **interface** AuthenticationService {

User signUp(SignUpRequest signUpRequest);

JwtAuthenticationResponse signin(SignUpRequest signinRequest);

JwtAuthenticationResponse refreshToken(RefreshTokenRequest refreshTokenRequest);

}

AuthenticationServiceImpl:

**import** java.util.HashMap;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** org.springframework.stereotype.Service;

**import** com.springSecurity.JwtSpringsecurity.Dto.JwtAuthenticationResponse;

**import** com.springSecurity.JwtSpringsecurity.Dto.RefreshTokenRequest;

**import** com.springSecurity.JwtSpringsecurity.Dto.SignUpRequest;

**import** com.springSecurity.JwtSpringsecurity.Entities.Role;

**import** com.springSecurity.JwtSpringsecurity.Entities.User;

**import** com.springSecurity.JwtSpringsecurity.Repositories.UserRepository;

**import** com.springSecurity.JwtSpringsecurity.Service.AuthenticationService;

**import** com.springSecurity.JwtSpringsecurity.Service.JwtService;

**import** lombok.RequiredArgsConstructor;

@Service

@RequiredArgsConstructor

**public** **class** AuthenticationServiceImpl **implements** AuthenticationService {

**private** **final** UserRepository userRepository;

**private** **final** PasswordEncoder passwordEncoder;

**private** **final** AuthenticationManager authenticationManager;

**private** **final** JwtService jwtService;

**public** User signUp(SignUpRequest signUpRequest) {

User user = **new** User();

user.setEmail(signUpRequest.getEmail());

user.setFirstName(signUpRequest.getFirstName());

user.setLastName(signUpRequest.getLastName());

user.setRole(Role.***USER***);

user.setPassword(passwordEncoder.encode(signUpRequest.getPassword()));

**return** userRepository.save(user);

}

**public** JwtAuthenticationResponse signin(SignUpRequest signinRequest) {

authenticationManager.authenticate(**new** UsernamePasswordAuthenticationToken(

signinRequest.getEmail(),

signinRequest.getPassword())

);

**var** user = userRepository.findByEmail(signinRequest.getEmail()).orElseThrow(() ->

**new** IllegalArgumentException("Invalid email or password."));

**var** jwt = jwtService.generateToken(user);

**var** refreshToken = jwtService.generateRefreshToken(**new** HashMap<>(), user);

JwtAuthenticationResponse jwtAuthenticationResponse = **new** JwtAuthenticationResponse();

jwtAuthenticationResponse.setToken(jwt);

jwtAuthenticationResponse.setRefreshToken(refreshToken);

**return** jwtAuthenticationResponse;

}

**public** JwtAuthenticationResponse refreshToken(RefreshTokenRequest refreshTokenRequest) {

String userEmail = jwtService.extractUserName(refreshTokenRequest.getToken());

User user = userRepository.findByEmail(userEmail).orElseThrow();

**if**(jwtService.isTokenValid(refreshTokenRequest.getToken(), user)) {

**var** jwt = jwtService.generateToken(user);

JwtAuthenticationResponse jwtAuthenticationResponse = **new** JwtAuthenticationResponse();

jwtAuthenticationResponse.setToken(jwt);

jwtAuthenticationResponse.setRefreshToken(refreshTokenRequest.getToken());

**return** jwtAuthenticationResponse;

}

**return** **null**;

}

}

Step 15 – **implements** CommandLineRunner in our main application, because we need save our default user or admin

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** com.springSecurity.JwtSpringsecurity.Entities.Role;

**import** com.springSecurity.JwtSpringsecurity.Entities.User;

**import** com.springSecurity.JwtSpringsecurity.Repositories.UserRepository;

@SpringBootApplication

**public** **class** JwtSpringsecurityApplication **implements** CommandLineRunner{

@Autowired

**private** UserRepository userRepository;

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(JwtSpringsecurityApplication.**class**, args);

}

@Override

**public** **void** run(String... args) {

User adminAccount = userRepository.findByRole(Role.***ADMIN***);

**if**(**null** == adminAccount) {

User user = **new** User();

user.setEmail("admin@gmail.com");

user.setFirstName("admin");

user.setLastName("admin");

user.setRole(Role.***ADMIN***);

user.setPassword(**new** BCryptPasswordEncoder().encode("admin"));

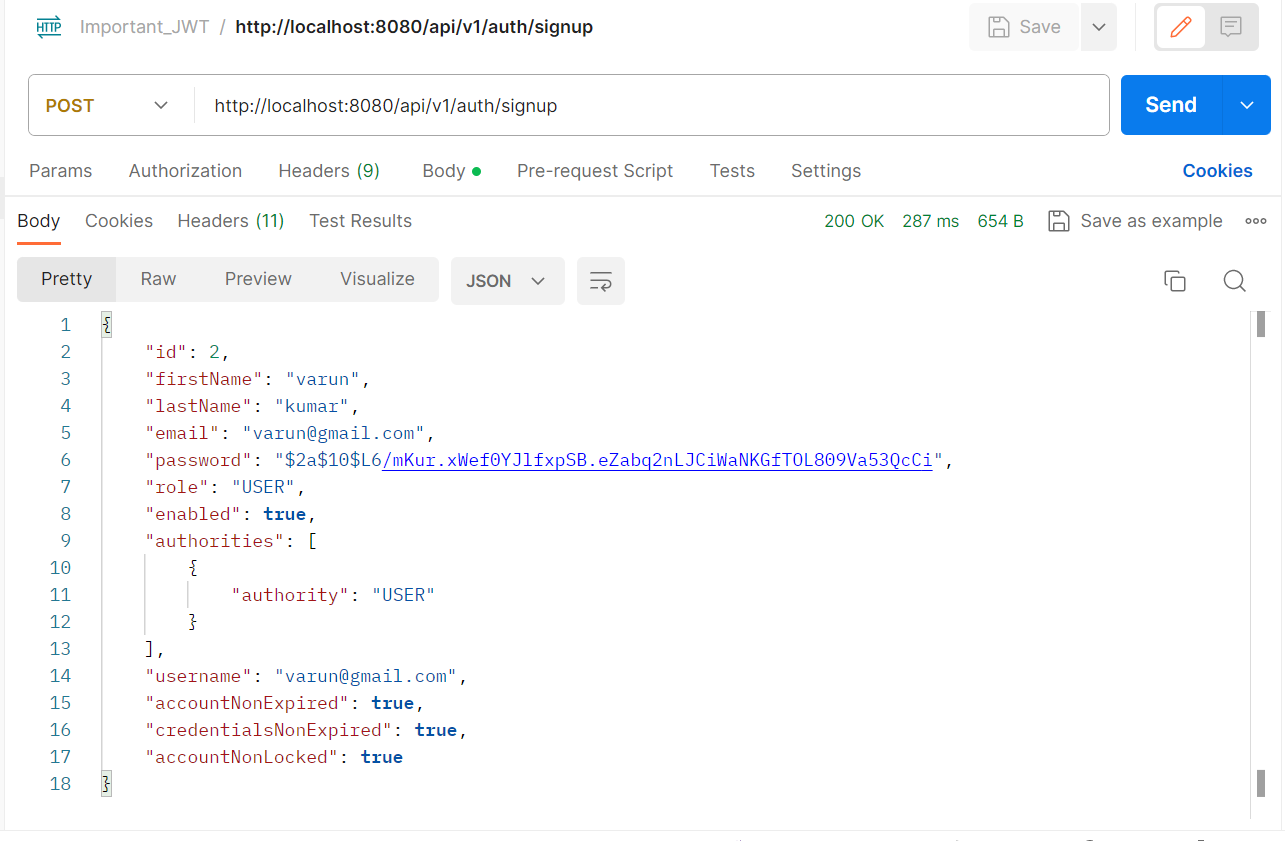
userRepository.save(user);

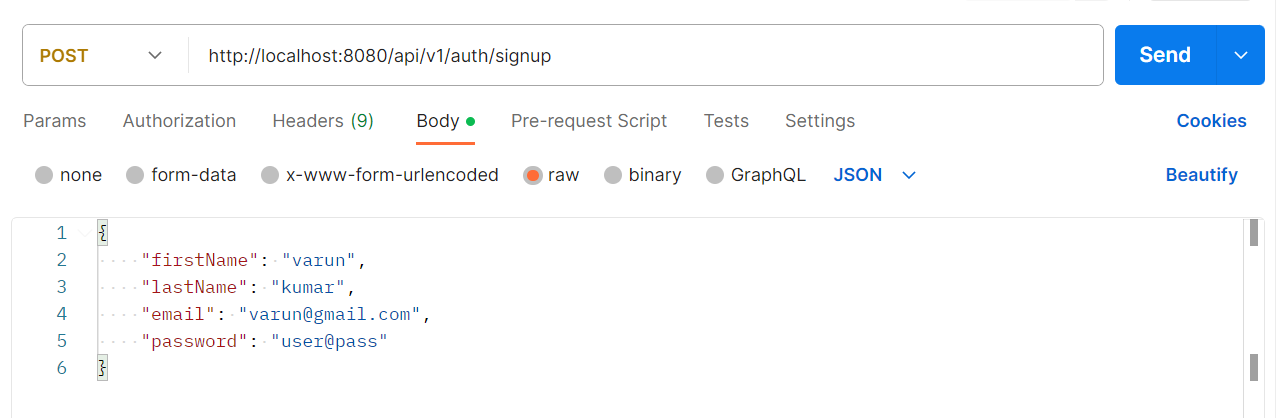
}

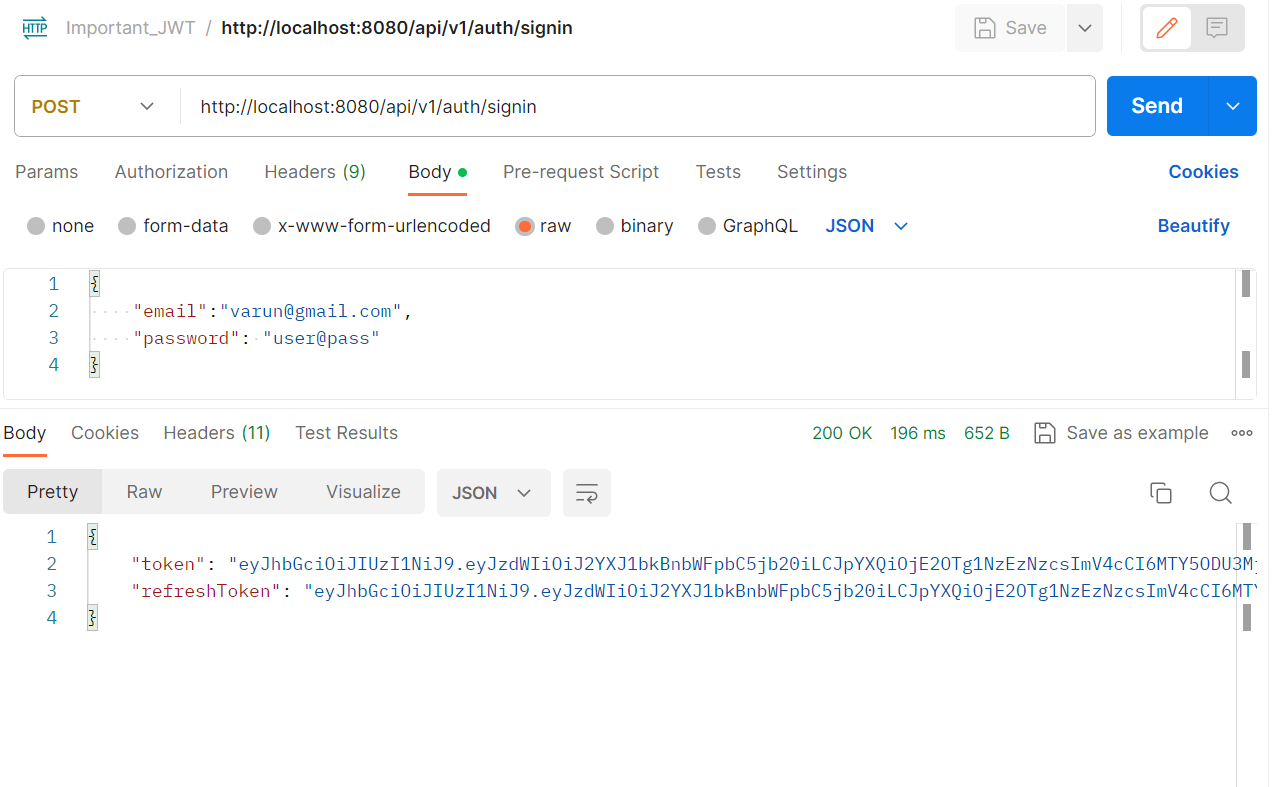
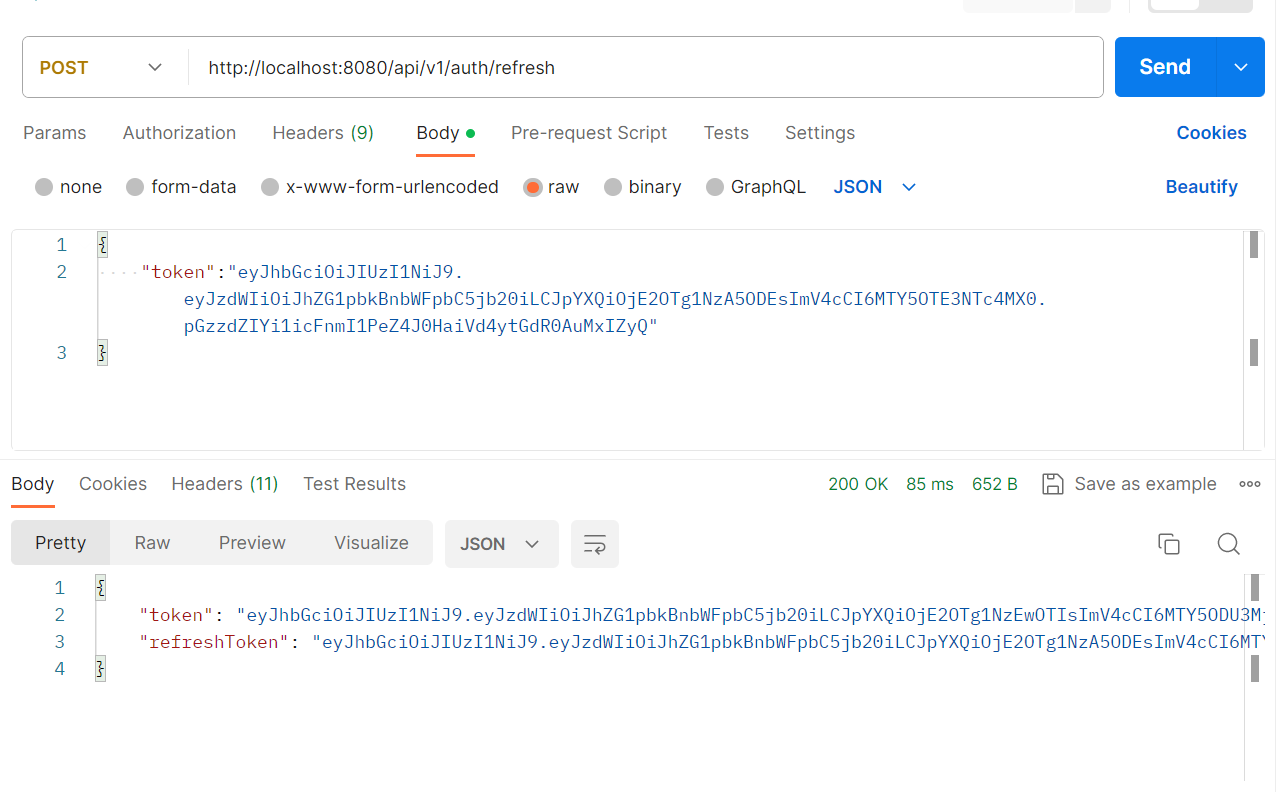
}

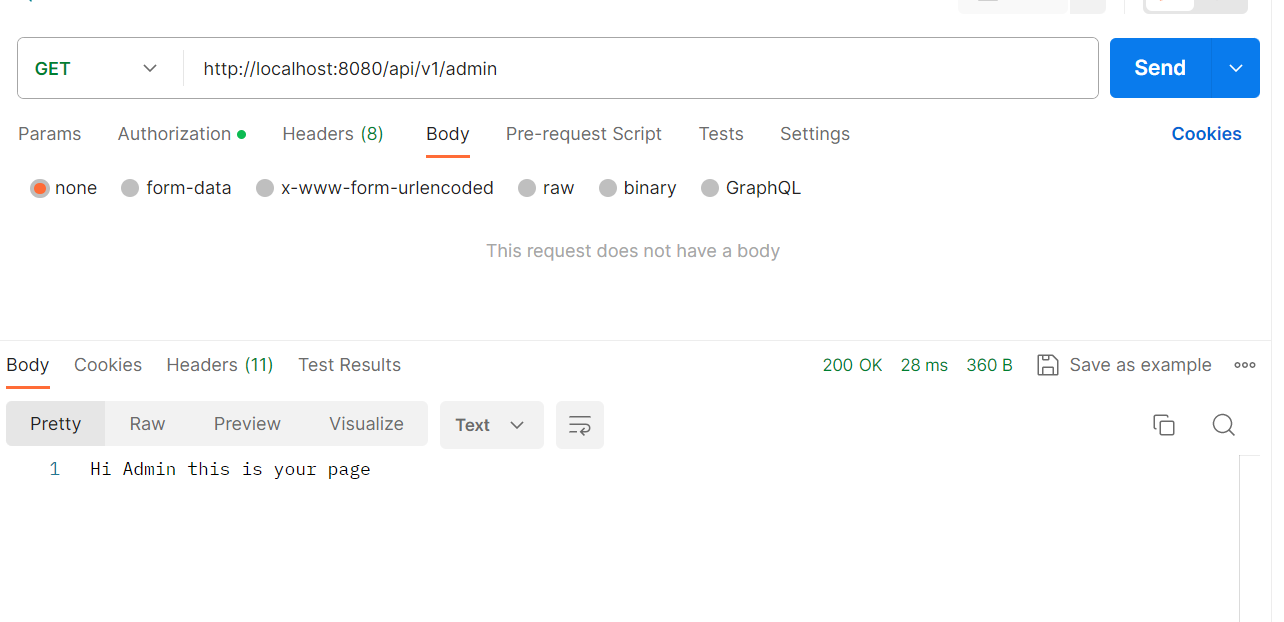
}

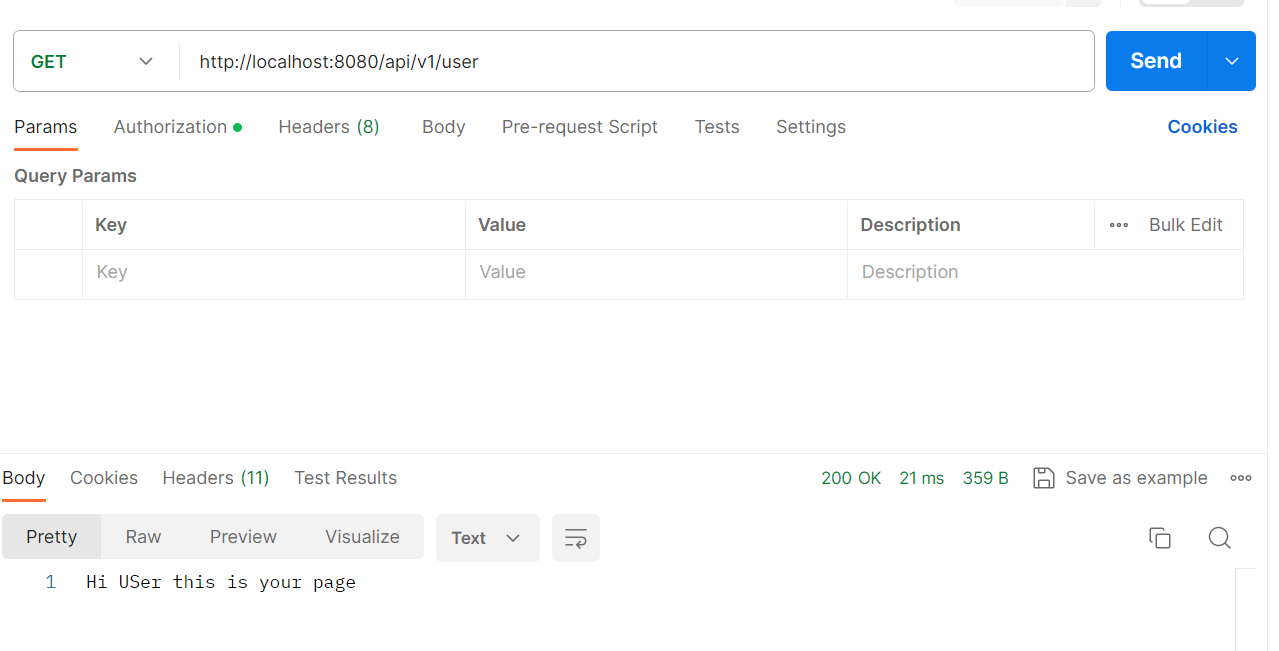
Output of all the end points are:  
  
<http://localhost:8080/api/v1/auth/signup>

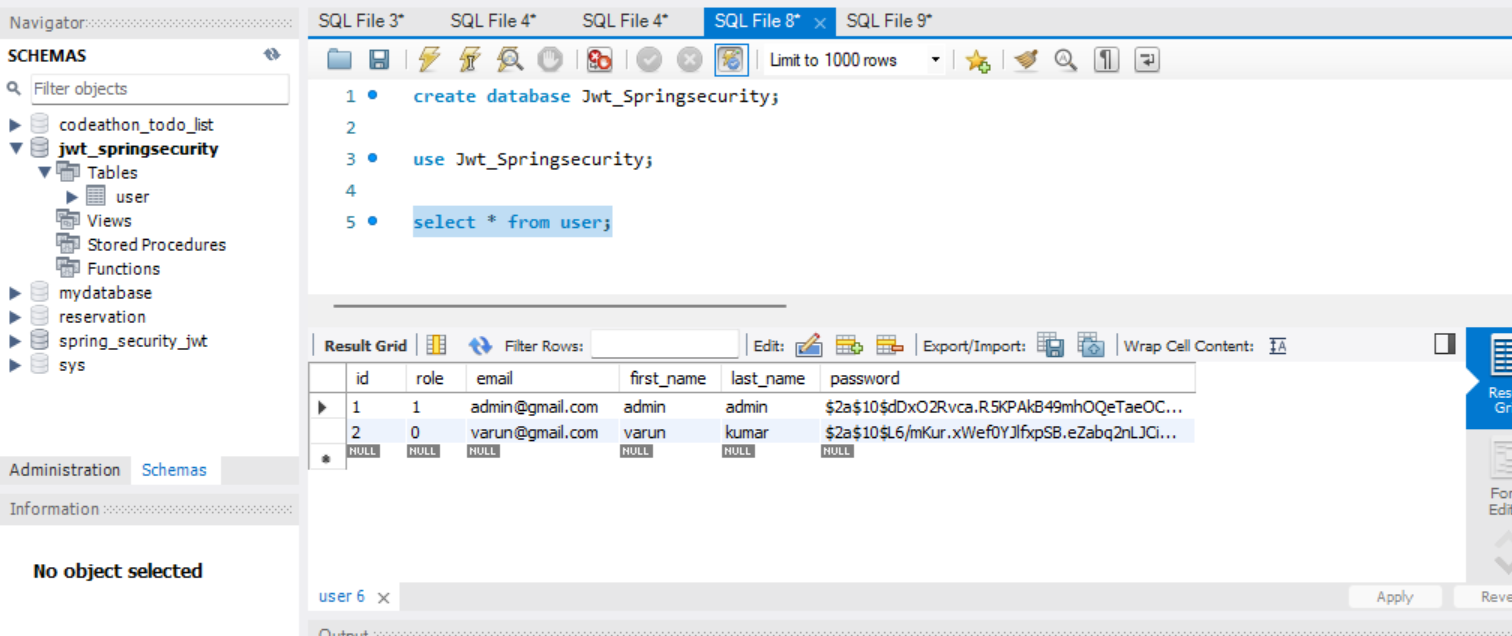




<http://localhost:8080/api/v1/auth/signin>  
  
  
<http://localhost:8080/api/v1/auth/refresh>  
  


<http://localhost:8080/api/v1/admin>  
  


<http://localhost:8080/api/v1/user>  
  
  
  
Data base looks like this:



Steps to create a Spring security in Spring Boot Application?

1) Creating a user entity

2) Creating a Enum for role

3) Configure an application.properties

4) Creating a user repository

5) Adding a Spring security and JWT dependencies

6) Implements a UserDetails for User Entity Class with UserDetails

7) Creating a JwtService interface and JwtServiceImpl

8) Creating JwtAuthenticationFilter and UserService in our project

9) Creating a UserService interface and UserServiceIImpl class

10) Creating a SecurityConfiguration class on config package

11) Creating our Api end points to controller layer

12) We are creating a registered Apis. For the creating a registered Apis, we need to create a dto package and add it

13) For this we are creating a AuthenticationService interface and AuthenticationServiceImpl class on Service package

14) implements CommandLineRunner in our main application, because we need save our default user or admin