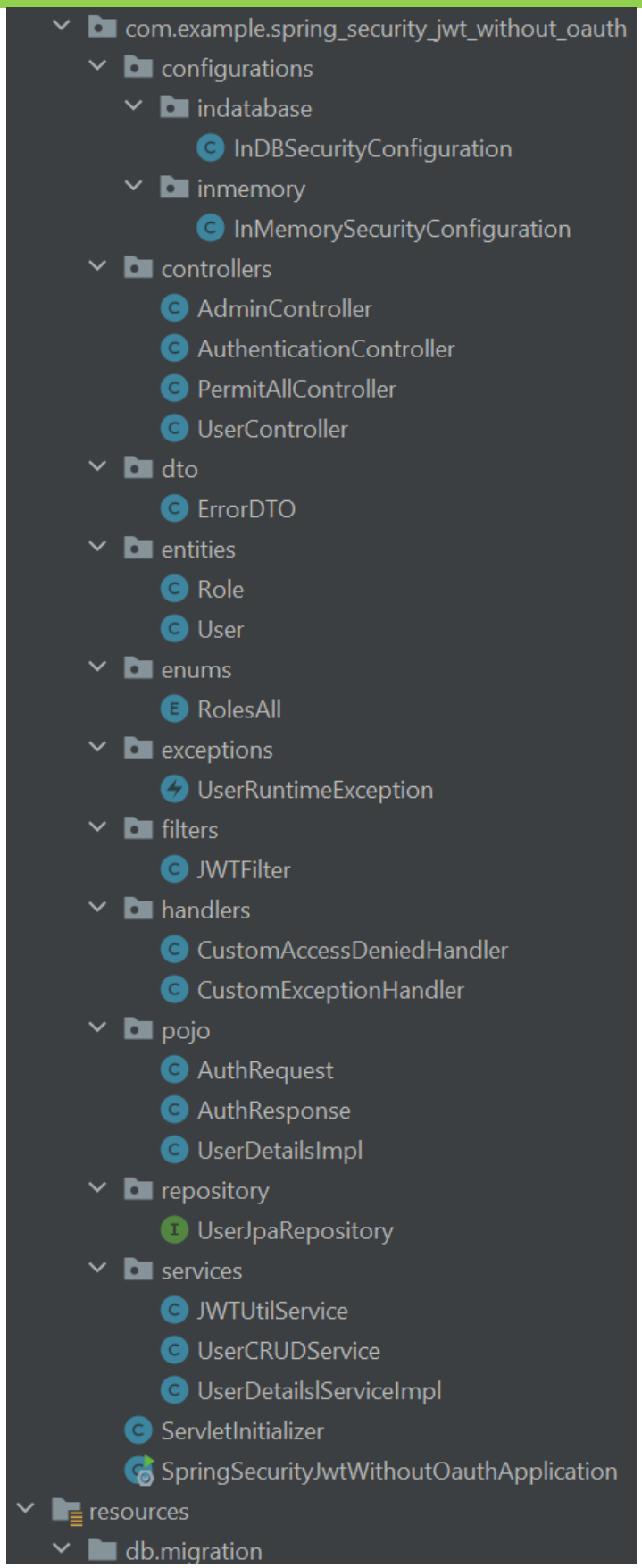
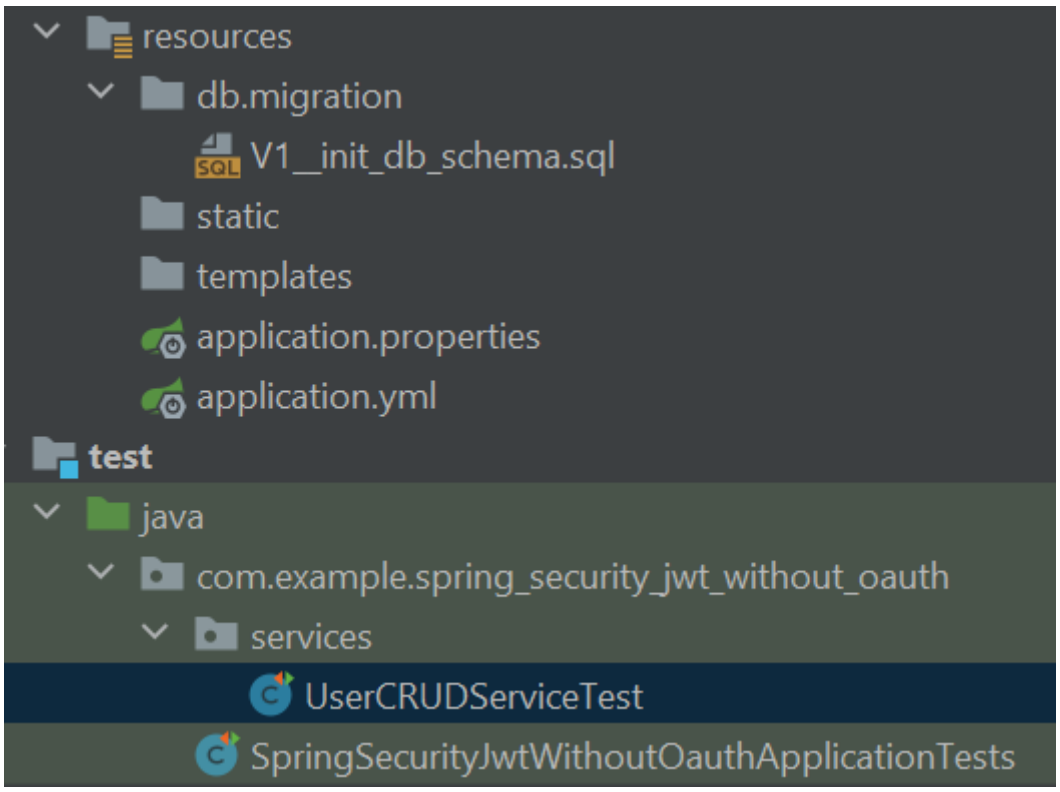
1. Project structure





2. Gradle build settings

```
plugins {
    id 'java'
    id 'war'
    id 'org.springframework.boot' version '3.0.1'
    id 'io.spring.dependency-management' version
'1.1.0'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    //
```

```
https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt
    implementation 'io.jsonwebtoken:jjwt:0.9.1'
    //
https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api
    implementation 'javax.xml.bind:jaxb-api:2.3.1'


    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.flywaydb:flyway-core'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
}

tasks.named('test') {
    useJUnitPlatform()
}
```

3. Application properties files

application.yml

```
spring:
  .websecurity:
    debug: true
  jpa:
    hibernate:
      ddl-auto: create
      show-sql: true
  sql:
    init:
      mode: always
jwt:
```

```
  sessionTime: 120000000
  secret: secret_key
```

application.properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=postgres
spring.datasource.password=sA#259979148307
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Po
stgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
#globally_quoted_identifiers=true -> this means we can
usereserved(quoted)names in our code
spring.jpa.properties.hibernate.globally_quoted_identifiers=true
#With FlyDB you can not create spring.jpa.defer-
datasourceinitialization=true property
#spring.jpa.defer-datasource-initialization=true
spring.flyway.enabled= true
spring.flyway.locations[0] = classpath:/db/migration
spring.flyway.baseline-on-migrate=true
```

4. SQL for FlyDB migration

```
CREATE SEQUENCE IF NOT EXISTS roles_id_seq
start 1
increment 1;

CREATE SEQUENCE IF NOT EXISTS users_id_seq
start 1
increment 1;


create table if not exists users
(
    id bigint not null primary key,
    login varchar(255) unique not null,
    password varchar(255) not null
);

create table if not exists roles
(
    id bigint not null primary key,
    role varchar(255) not null,
    user_id  bigint  references users
);


insert into users(id, login, password) values (-1, 'user' ,
'$2a$12$6BlCjcTXmS/3IfcSiNI2COpDKk8LEqKr7PXcv9GJuLWgJfIXhGv
```

```
RC' ) ,
                                                    (-2, 'doctor'
,'$2a$12$Hrm75q6NsauybkERr3Kz9eKX1JzIm2FaGMpRmg9t7HizfnPoqV
G/2' ) ,
                                                   (-3, 'admin' ,
'$2a$12$6aZWlxR/7WmzEobBqezqh.uYXjXdV0VCpO2FZVRXB/Pl9gD5D.q
iC' ) ;

insert into roles(id, role) values (-1, 'USER' ) ,
                                    (-2, 'DOCTOR' ) ,
                                    (-3, 'ADMIN' ) ;
```

5. Spring security configuration from DB with JWT token

```
package
com.example.spring_security_jwt_without_oauth.confi
gurations.indatabase;

import
com.example.spring_security_jwt_without_oauth.filte
rs.JWTFilter;
import
com.example.spring_security_jwt_without_oauth.handl
ers.CustomAccessDeniedHandler;
import
com.example.spring_security_jwt_without_oauth.servi
ces.UserDetailslServiceImpl;
import
org.springframework.beans.factory.annotation.Autowi
red;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuratio
n;
import org.springframework.http.HttpMethod;
import
org.springframework.security.authentication.Authent
icationManager;
import
org.springframework.security.authentication.Authent
icationProvider;
import
org.springframework.security.authentication.dao.Dao
AuthenticationProvider;
```

```
import
org.springframework.security.config.annotation.auth
entication.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.meth
od.configuration.EnableMethodSecurity;
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.
configuration.WebSecurityCustomizer;
import
org.springframework.security.config.http.SessionCre
ationPolicy;
import
org.springframework.security.core.userdetails.User;
import
org.springframework.security.core.userdetails.UserD
etailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPa
sswordEncoder;
import
org.springframework.security.crypto.password.NoOpPa
sswordEncoder;
import
org.springframework.security.crypto.password.Passwo
rdEncoder;
import
org.springframework.security.provisioning.InMemoryU
serDetailsManager;
import
org.springframework.security.web.SecurityFilterChai
n;
import
org.springframework.security.web.access.AccessDenie
dHandler;
import
org.springframework.security.web.authentication.Use
rnamePasswordAuthenticationFilter;
```

```java
/**
 * Spring 3.0 Security configuration with DB with
JwtFilter (JWT token)
 */

@Configuration
@EnableWebSecurity
@EnableMethodSecurity

public class InDBSecurityConfiguration {
    @Value("${spring.websecurity.debug}")
    boolean webSecurityDebug;

    @Autowired
    private JWTFilter jwtFilter;

    @Autowired
    private UserDetailslServiceImpl
userDetailslServiceImpl;

    @Autowired
    private CustomAccessDeniedHandler
customAccessDeniedHandler;

    @Bean
    public BCryptPasswordEncoder
bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UserDetailsService userDetailsService()
{
        return userDetailslServiceImpl;
    }

    //Create AuthenticationManager
    //@Bean
    /*
    public AuthenticationManager
authenticationManager(HttpSecurity http,
BCryptPasswordEncoder bCryptPasswordEncoder,
UserDetailslServiceImpl userDetailService)
        throws Exception {
        return
```

```
http.getSharedObject(AuthenticationManagerBuilder.class)

.userDetailsService(userDetailService)

.passwordEncoder(bCryptPasswordEncoder)
                .and()
                .build();
    }
    */

    //Create AuthenticationProvider which compare login and passwords
    @Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider authenticationProvider=new DaoAuthenticationProvider();

authenticationProvider.setUserDetailsService(userDetailsService());

authenticationProvider.setPasswordEncoder(bCryptPasswordEncoder());
        return authenticationProvider;
    }

    //Chain of configuration, HTTP settings
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
                .csrf()
                    .disable()
                .authorizeHttpRequests()
                    .requestMatchers(   "/",

"/permitall/**",

"/authenticate/"
                    )
                    .permitAll()
                .and()
```

```
                .sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATEL
ESS)
                .and()
                .authorizeHttpRequests()
                    .requestMatchers(
"/api/v1/users/",

"/api/v1/admins/"
                    )
                    .authenticated()
                .anyRequest().denyAll()
                .and()
                .exceptionHandling()

.accessDeniedHandler(customAccessDeniedHandler)
                .and()
                .addFilterBefore(jwtFilter,

UsernamePasswordAuthenticationFilter.class);
                //.formLogin();
        return  httpSecurity.build();
    }
//  Security debugging is enabled.
    @Bean
    public WebSecurityCustomizer
webSecurityCustomizer() {
        return (web) ->
web.debug(webSecurityDebug);
    }
}
```

from memory security configuration

```
package
com.example.spring_security_jwt_without_oauth.configura
tions.inmemory;

import
com.example.spring_security_jwt_without_oauth.handlers.
CustomAccessDeniedHandler;
import
org.springframework.beans.factory.annotation.Autowired;
import
```

```
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.authentication.Authenticat
ionManager;
import
org.springframework.security.config.annotation.authenti
cation.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.c
onfiguration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.buil
ders.HttpSecurity;
import
org.springframework.security.config.annotation.web.conf
iguration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.conf
iguration.WebSecurityCustomizer;
import
org.springframework.security.core.userdetails.User;
import
org.springframework.security.core.userdetails.UserDetai
lsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswo
rdEncoder;
import
org.springframework.security.provisioning.InMemoryUserD
etailsManager;
import
org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.access.AccessDeniedHan
dler;
import
org.springframework.security.web.access.AccessDeniedHan
dlerImpl;

/**
 * Spring 3.0 Security configuration without DB, in
```

```
memory
 */
/*
    @Configuration
    @EnableWebSecurity
    @EnableMethodSecurity
*/
public class InMemorySecurityConfiguration {

    @Autowired
    private CustomAccessDeniedHandler
customAccessDeniedHandler;

    @Bean
    public BCryptPasswordEncoder
bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
    //Create InMemoryUserDetailService which return
users from memory
    @Bean
    public UserDetailsService
userDetailsService(BCryptPasswordEncoder
bCryptPasswordEncoder) {
        InMemoryUserDetailsManager
inMemoryUserDetailsManager =
                new InMemoryUserDetailsManager();
        //Create user
        inMemoryUserDetailsManager
                .createUser(
                        User.withUsername("user")

.password(bCryptPasswordEncoder.encode("user"))
                                .roles("USER")
                                .build()
                );
        //Create admin
        inMemoryUserDetailsManager
                .createUser(
                        User.withUsername("admin")

.password(bCryptPasswordEncoder.encode("admin"))
                                .roles("USER", "ADMIN")
                                .build()
                );
```

```
        return inMemoryUserDetailsManager;

    }

    //Create AuthenticationManager which compare login
and passwords
    @Bean
    public AuthenticationManager
authenticationManager(HttpSecurity httpSecurity,

BCryptPasswordEncoder bCryptPasswordEncoder,

UserDetailsService userDetailsService)
        throws Exception {
        AuthenticationManager authenticationManager =
httpSecurity

.getSharedObject(AuthenticationManagerBuilder.class)
                .userDetailsService(userDetailsService)
                .passwordEncoder(bCryptPasswordEncoder)
                .and()
                .build();
        return authenticationManager;
    }

    //Chain of configuration, HTTP settings
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity
httpSecurity) throws Exception {
        httpSecurity
                .csrf().disable()
                .authorizeHttpRequests(requests -> {
                    try {
                        requests

.requestMatchers(HttpMethod.GET, "/",
"/permitall/**").permitAll()

.requestMatchers(HttpMethod.POST, "/",
"/permitall/**").permitAll()

.requestMatchers(HttpMethod.GET,
"/api/v1/users/").hasAnyRole("USER", "ADMIN")

.requestMatchers(HttpMethod.POST,
```

```
"/api/v1/users/").hasAnyRole("USER", "ADMIN")

.requestMatchers(HttpMethod.GET,
"/api/v1/admins/").hasRole("ADMIN")

.requestMatchers(HttpMethod.POST,
"/api/v1/admins/").hasRole("ADMIN")
                                    .anyRequest().denyAll()
                                    .and();
/*                                    .exceptionHandling()

.accessDeniedHandler(customAccessDeniedHandler);*/
                        } catch (Exception e) {
                            throw new
RuntimeException(e.getMessage());
                        }
                    })
                    //.httpBasic();
                    .formLogin();
            return  httpSecurity.build();
    }
    @Value("${spring.websecurity.debug:false}")
    boolean webSecurityDebug;

    //Add path to ignore authentication
    @Bean
    public WebSecurityCustomizer
webSecurityCustomizer() {
            return (web) -> web.debug(webSecurityDebug)
                    .ignoring()
                    .requestMatchers("/ignoring/**",
"/favicon.ico");
    }

}
```

6. Controllers

Admin url controller

```
package
com.example.spring_security_jwt_without_oauth.controlle
rs;

import org.springframework.http.ResponseEntity;
import
```

```
org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;

/*
"/api/v1/users/**"
*/
@RestController
@RequestMapping(path = "/api/v1/admins/")
public class AdminController {
    @GetMapping()
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<String> getAdmin() {
        return ResponseEntity.ok("Get admin");
    }

    @PostMapping()
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<String> postAdmin() {
        return ResponseEntity.ok("Post admin");
    }
}
```

Authentication controller that return JWT token to client

```
package
com.example.spring_security_jwt_without_oauth.controllers;

import
com.example.spring_security_jwt_without_oauth.pojo.AuthRequest;
import
com.example.spring_security_jwt_without_oauth.pojo.AuthResponse;
import
com.example.spring_security_jwt_without_oauth.pojo.UserDetailsImpl;
```

```java
import
com.example.spring_security_jwt_without_oauth.services.
JWTUtilService;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import
org.springframework.security.authentication.Authenticat
ionManager;
import
org.springframework.security.authentication.Authenticat
ionProvider;
import
org.springframework.security.authentication.BadCredenti
alsException;
import
org.springframework.security.authentication.UsernamePas
swordAuthenticationToken;
import
org.springframework.security.core.Authentication;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.ResponseStatus;
import
org.springframework.web.bind.annotation.RestController;
import
org.springframework.web.server.ResponseStatusException;

/**
 * This is where the JWT token is actually issued.
 * The user makes a POST request with a username and
password to /authenticate,
 * and receives a generated token in response. The
token is generated by the generateToken() method
 * from JWTUtilService.java.
 */
@RestController
public class AuthenticationController {
    @Autowired
    private AuthenticationProvider
authenticationProvider;
    //Or we can can use AuthenticationManager
```

```
    //private AuthenticationManager
authenticationManager;
    @Autowired
    private JWTUtilService jwtTokenUtil;

    @PostMapping("/authenticate/")
    @ResponseStatus(HttpStatus.OK)
    public AuthResponse
createAuthenticationToken(@RequestBody AuthRequest
authRequest) {
        Authentication authentication;
        try {
            authentication = authenticationProvider
                    .authenticate(
                            new
UsernamePasswordAuthenticationToken(

authRequest.getName(),

authRequest.getPassword()
                                    )
                    );
            System.out.println(authentication);
        } catch (BadCredentialsException e) {
            throw new
ResponseStatusException(HttpStatus.UNAUTHORIZED,
                    "UNAUTHORIZED : Name and password
are wrong", e);
        }
        // when creating a token, the username is put
in it as a Subject claim(claim subject)
        // and the list of authorities as a custom
claim(claim authorities)
        String jwt =
jwtTokenUtil.generateToken((UserDetailsImpl)
authentication.getPrincipal());
        return new AuthResponse(jwt);
    }
}
```

PermitALL (доступ разрешен всем) controllers

```
package
com.example.spring_security_jwt_without_oauth.controlle
rs;
```

```java
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;


@RestController
@RequestMapping(path = "/")
public class PermitAllController {
    @GetMapping()
    public ResponseEntity<String> getIndex() {
        return ResponseEntity.ok("Get index");
    }
    @PostMapping()
    public ResponseEntity<String> postIndex() {
        return ResponseEntity.ok("Post index");
    }

    @GetMapping(path = "permitall/**")
    public ResponseEntity<String> getPermitAll() {
        return ResponseEntity.ok("Get PermitAll");
    }

    @PostMapping(path = "permitall/**")
    public ResponseEntity<String> postPermitAll() {
        return ResponseEntity.ok("Post PermitAll");
    }

    @GetMapping(path = "ignoring/**")
    public ResponseEntity<String> getIgnoringAll() {
        return ResponseEntity.ok("Get Ignorin");
    }

    @PostMapping(path = "ignoring/**")
    public ResponseEntity<String> postIgnoringAll() {
        return ResponseEntity.ok("Post Ignorin");
    }

}
```

UserController

```
package
com.example.spring_security_jwt_without_oauth.controlle
rs;

import org.springframework.http.ResponseEntity;
import
org.springframework.security.access.prepost.PreAuthoriz
e;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;
/*
"/api/v1/users/**"
 */
@RestController
@RequestMapping(path = "/api/v1/users/")
public class UserController {

    @GetMapping()
    @PreAuthorize("hasAnyAuthority('ADMIN','USER')")
    public ResponseEntity<String> getUser() {

        return ResponseEntity.ok("Get user");
    }

    @PostMapping()
    @PreAuthorize("hasAnyAuthority('ADMIN','USER')")
    public ResponseEntity<String> postUser() {
        return ResponseEntity.ok("Post user");
    }
}
```

7. DTO

ErrorDTO

```
package
com.example.spring_security_jwt_without_oauth.dto;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * Just object, which RestAdviceController will
response to
 client
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ErrorDTO {
    private int statusCode;
    private String messages;
}
```

8. Entities

Role

```
package
com.example.spring_security_jwt_without_oauth.entities;

import
com.example.spring_security_jwt_without_oauth.enums.Rol
esAll;
import jakarta.persistence.*;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.Cascade;

@NoArgsConstructor
@EqualsAndHashCode
@Getter

@Entity
@Table(name = "roles")
@SequenceGenerator(sequenceName = "roles_id_seq",
        name = "roles_id_seq", allocationSize = 1)
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.
SEQUENCE, generator =
```

```
            "roles_id_seq")
    @Column(nullable = false)
    private Long id;
    @Column(nullable = false)
    private String role;

    public Role(RolesAll role) {
        this.role = role.toString();
    }
}
```

User

```
package
com.example.spring_security_jwt_without_oauth.entities;

import
com.example.spring_security_jwt_without_oauth.enums.Rol
esAll;
import jakarta.persistence.*;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.HashSet;
import java.util.Set;


@NoArgsConstructor
@EqualsAndHashCode
@Getter

@Entity
@Table(name = "users")
@SequenceGenerator(sequenceName = "users_id_seq",
        name = "users_id_seq", allocationSize = 1)
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.
SEQUENCE, generator =
            "users_id_seq")
    @Column(nullable = false)
    private Long id;
```

```
    @Column(unique = true, nullable = false)
    private String login;
    @Column(nullable = false)
    private String password;


    @OneToMany(
            orphanRemoval = true,
            cascade ={ CascadeType.ALL},
            fetch = FetchType.EAGER
    )
    @JoinColumn(name = "user_id")
    @Column(nullable = false)
    private Set<Role> roles = new HashSet<>() ;

    public User(String login, String password, RolesAll
role) {
        this.login = login;
        this.password = password;
        this.roles.add(new Role(role));
    }
}
```

9. Enums

RolesALL

```
package
com.example.spring_security_jwt_without_oauth.enums
;

public enum RolesAll {
    USER,
    ADMIN
}
```

10. Exceptions

```
package
com.example.spring_security_jwt_without_oauth.excep
tions;

import lombok.Getter;
import lombok.NoArgsConstructor;
import
org.springframework.web.bind.annotation.GetMapping;
```

```
@NoArgsConstructor
@Getter
public class UserRuntimeException extends
RuntimeException{
    private int statusCode;
    public UserRuntimeException(int statusCode,
String message) {
        super(message) ;
        this. statusCode = statusCode;
    }
}
```

11. Filters

```
package
com.example.spring_security_jwt_without_oauth.filte
rs;

import
com.example.spring_security_jwt_without_oauth.servi
ces.JWTUtilService;
import
com.example.spring_security_jwt_without_oauth.servi
ces.UserDetailslServiceImpl;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import
org.springframework.beans.factory.annotation.Autowi
red;
import
org.springframework.security.authentication.Usernam
ePasswordAuthenticationToken;
import
org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.Authori
tyUtils;
import
org.springframework.security.core.context.SecurityC
ontextHolder;
import org.springframework.stereotype.Component;
import
```

```java
org.springframework.web.filter.OncePerRequestFilter
;

import java.io.IOException;
import java.util.List;

/**
 * Filter for check exist or not JWT token
 * With each request, we take a JWT token from the
Authorization header (it starts with the "Bearer"
prefix).
 * We extract the username (claim subject) and the
list of authorities (claim authorities) from it.
 * We wrote both brands into the token when it was
generated in the controller.
 * At the same time, when retrieving claims, the
validity of the token is checked.
 * To do this, you do not need to make any requests
to the database: the token itself and jwt.secret
(written in application.yml)
 * are enough. Based on this secret, the token was
generated, and based on it, it is then checked each
time using a hash function
 * (this is done by the jjwt library).
 * If everything is OK, then having the username
and the list of authorities (extracted in step 2),
 * we create an Authentication object (more
precisely, its subclass
UsernamePasswordAuthenticationToken).
 * And set the Authentication object to the
SecurityContext. So it is necessary for Spring
Security.
 * If not everything is OK with the token, then an
exception was thrown in paragraphs 2-3, and the
filter will not let the request to the controller
to the protected / url pass.
 */
@Component
public class JWTFilter extends OncePerRequestFilter
{
    @Autowired
    private JWTUtilService jwtUtil;

    @Override
    protected void
```

```
doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {
        //Get header Authorization from request
        final String authorizationHeader =
request.getHeader("Authorization");

        String username = null;
        String jwt = null;
        //Cut prefix bearer if it not null and
start with Bearer
        if (authorizationHeader != null &&
authorizationHeader.startsWith("Bearer ")) {
            jwt = authorizationHeader.substring(7);
            //if the signature does not match the
calculated one, then SignatureException
            //if the signature is incorrect (not
parsed), then MalformedJwtException
            //if the signature has expired, then
ExpiredJwtException
            username =
jwtUtil.extractUsername(jwt);
        }
            /*
            * If everything is OK, then having the
username and the list of authorities (extracted in
step 2),
            * we create an Authentication object
(more precisely, its subclass
UsernamePasswordAuthenticationToken).
            * And set the Authentication object to
the SecurityContext. So it is necessary for Spring
Security.
            */
        if (username != null &&
SecurityContextHolder.getContext().getAuthenticatio
n() == null) {

            String commaSeparatedListOfAuthorities
= jwtUtil.extractAuthorities(jwt);
            List<GrantedAuthority> authorities =
AuthorityUtils.commaSeparatedStringToAuthorityList(
commaSeparatedListOfAuthorities);
            UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken =
```

```
                    new
UsernamePasswordAuthenticationToken(
                            username, null,
authorities);


SecurityContextHolder.getContext().setAuthenticatio
n(usernamePasswordAuthenticationToken);

        }
        chain.doFilter(request, response);
    }
}
```

12.Handlers

Access denied handler in Spring Security

```
package
com.example.spring_security_jwt_without_oauth.handl
ers;

import
com.example.spring_security_jwt_without_oauth.dto.E
rrorDTO;
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import
org.springframework.security.access.AccessDeniedExc
eption;
import
org.springframework.security.web.access.AccessDenie
dHandler;
import org.springframework.stereotype.Component;

import java.io.IOException;

@Component
public class CustomAccessDeniedHandler implements
AccessDeniedHandler {
    ObjectMapper objectMapper = new ObjectMapper();
    @Override
    public void handle(HttpServletRequest request,
HttpServletResponse response, AccessDeniedException
```

```
accessDeniedException) throws IOException,
ServletException {
        System.out.println("-------------ACCESS
DENIED HANDLE---------------");
        String requestURI =
request.getRequestURI();
        ErrorDTO errorDTO = new ErrorDTO(500,
"CustomAccessDeniedHandler : requestURI : " +
requestURI);
        String string =
objectMapper.writeValueAsString(errorDTO);
        response.setStatus(500);

response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write(string);
    }
}
```

CustomExceptionHandler.java

```
package
com.example.spring_security_jwt_without_oauth.handlers;

import
com.example.spring_security_jwt_without_oauth.dto.Error
DTO;
import
com.example.spring_security_jwt_without_oauth.exception
s.UserRuntimeException;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.ExceptionHandle
r;
import
org.springframework.web.bind.annotation.RestControllerA
dvice;

@RestControllerAdvice
public class CustomExceptionHandler {
    @ExceptionHandler(value
            = {RuntimeException. class})
    public ResponseEntity<ErrorDTO>
    exceptRuntimeException(RuntimeException exception)
{
```

```
        System.out.println("CustomExceptionHandler
exceptRuntimeException");
        String message = exception.getMessage() ;
        return ResponseEntity
                . status(500)
                .body(new ErrorDTO(500,
"CustomExceptionHandler exceptRuntimeException" +
message)) ;
    }
    @ExceptionHandler(value
            = {UserRuntimeException. class})
    public ResponseEntity<ErrorDTO>
    exceptUserRuntimeException(UserRuntimeException
exception) {
        System.out.println("CustomExceptionHandler
exceptUserRuntimeException");
        int statusCode = exception.getStatusCode() ;
        String message = exception.getMessage() ;
        return ResponseEntity
                . status(500)
                .body(new ErrorDTO(statusCode,
"CustomExceptionHandler exceptUserRuntimeException " +
message)) ;
    }
}
```

13.Pojo objects

Request for JWT

```
package
com.example.spring_security_jwt_without_oauth.pojo;

import lombok.*;

/**
 * JWT authenticate request for JWT token
 */
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode
@Getter
@Setter
public class AuthRequest {
    private String name;
```

```
    private String password;
}
```

Response for JWT

```
package
com.example.spring_security_jwt_without_oauth.pojo;

import lombok.*;

/**
 * JWT authenticate response for JWT token
 */
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode
@Getter
@Setter
public class AuthResponse {
    private String jwtToken;
}
```

UserDetails implementation

```
package
com.example.spring_security_jwt_without_oauth.pojo;

import
com.example.spring_security_jwt_without_oauth.entities.
Role;
import
com.example.spring_security_jwt_without_oauth.entities.
User;
import
com.example.spring_security_jwt_without_oauth.enums.Rol
esAll;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import
org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrant
edAuthority;
import
```

```
org.springframework.security.core.userdetails.UserDetai
ls;

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

/**
 * Custom implementation of UserDetails
 */
@EqualsAndHashCode
public class UserDetailsImpl implements UserDetails {
    private final String name;
    private final String password;
    private final List<GrantedAuthority> authorities;

    public UserDetailsImpl (User user) {
        this.name = user.getLogin();
        this.password = user.getPassword();
        authorities = user.getRoles().stream()
                .map(role -> new

SimpleGrantedAuthority(role.getRole()))
                .collect(Collectors.toList()) ;
    }
    @Override
    public Collection<? extends GrantedAuthority>
getAuthorities() {
        return authorities;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return name;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
```

```
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

14.Jpa repositories

UserDetails implementation

```
package
com.example.spring_security_jwt_without_oauth.repositor
y;

import
com.example.spring_security_jwt_without_oauth.entities.
User;
import
org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface UserJpaRepository extends
JpaRepository<User,Long> {
    List<User> findAll();

    User findUserByLogin(String login);

    User save(User user);
}
```

15. Services

JWTUtilService

```
package
com.example.spring_security_jwt_without_oauth.services;

import
com.example.spring_security_jwt_without_oauth.pojo.User
DetailsImpl;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.SignatureAlgorithm;
import jakarta.annotation.PostConstruct;
import
org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

import io.jsonwebtoken.Jwts;
/**
    Service for creating JWT token
 */
@Service
public class JWTUtilService {

    @Value("${jwt.secret}")
    private String SECRET_KEY;

    @Value("${jwt.sessionTime}")
    private long sessionTime;

    //Create jwt token (put into name and authorities)
    public String generateToken(UserDetailsImpl
userDetails) {
        Map<String, Object> claims = new HashMap<>();
        String commaSeparatedListOfAuthorities =
                userDetails
                        .getAuthorities()

.stream().map(GrantedAuthority::getAuthority)

.collect(Collectors.joining(","));
```

```
        //Add claim authorities to ou jwt token
        claims.put("authorities",
commaSeparatedListOfAuthorities);
        String token = createToken(claims,
userDetails.getUsername());
        return token;
    }

    //extracting username from token (inside token
validation)
    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    //fetching authorities (inside token validation)
    public String extractAuthorities(String token) {
        Function<Claims, String> claimsListFunction =
claims -> {
            return (String)claims.get("authorities");
        };
        return extractClaim(token, claimsListFunction);
    }

    private  <T> T extractClaim(String token,
Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return
Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(
token).getBody();
    }

    private String createToken(Map<String, Object>
claims, String subject) {
        String token = Jwts
                .builder()
                .setClaims(claims)
                .setSubject(subject)
                .setExpiration(expireTimeFromNow())
                .signWith(SignatureAlgorithm.HS256,
                        SECRET_KEY).compact();
        return token;
```

```
    }

    private Date expireTimeFromNow() {
        Date date = new Date(System.currentTimeMillis()
+ sessionTime);
        return date;
    }

}
```

UserCRUDService

```
package
com.example.spring_security_jwt_without_oauth.services;

import
com.example.spring_security_jwt_without_oauth.entities.
User;
import
com.example.spring_security_jwt_without_oauth.exception
s.UserRuntimeException;
import
com.example.spring_security_jwt_without_oauth.repositor
y.UserJpaRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class UserCRUDService {
    @Autowired
    private UserJpaRepository userJpaRepository;

    public List<User> findAllUsers() {
        List<User> allUsers =
userJpaRepository.findAll();
        if(allUsers.isEmpty() || allUsers == null) {
            throw new UserRuntimeException(
                    500,
                    "users table is empty"
            );
        }
        return allUsers;
    }
```

```
    public User findUserByLogin(String login) {
        User userByLogin =
userJpaRepository.findUserByLogin(login);
        if (userByLogin == null) {
            throw new UserRuntimeException(
                    500,
                    ("Cant find user with login : " +
login)
            );
        }
        return userByLogin;
    }

    public User saveUser(User user) {
        System.out.println(user);
        User savedUser = userJpaRepository.save(user);
        if (savedUser == null) {
            throw new UserRuntimeException(
                    500,
                    "Saved user is null"
            );
        }
        return savedUser;
    }
}
```

UserDetailslServiceImpl.java

```
package
com.example.spring_security_jwt_without_oauth.services;

import
com.example.spring_security_jwt_without_oauth.entities.
User;
import
com.example.spring_security_jwt_without_oauth.exception
s.UserRuntimeException;
import
com.example.spring_security_jwt_without_oauth.pojo.User
DetailsImpl;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.userdetails.UserDetai
ls;
```

```
import
org.springframework.security.core.userdetails.UserDetai
lsService;
import
org.springframework.security.core.userdetails.UsernameN
otFoundException;
import org.springframework.stereotype.Component;

@Component
public class UserDetailslServiceImpl implements
UserDetailsService {

    @Autowired
    private UserCRUDService userCRUDService;

    @Override
    public UserDetails loadUserByUsername(String
username) throws UsernameNotFoundException {
        User userByLogin =
userCRUDService.findUserByLogin(username);
        UserDetailsImpl userDetails = new
UserDetailsImpl(userByLogin);
        if (userDetails == null) {
            throw new UserRuntimeException(
                    500,
                    ("Cant create userDetails in" +
this.getClass().getSimpleName())
            );
        }
        System.out.println("----------------------User
details---------------------------");
        System.out.println(userDetails);
        System.out.println(userDetails.getUsername());
        System.out.println(userDetails.getPassword());
        return userDetails;
    }
}
```

16. Integration tests

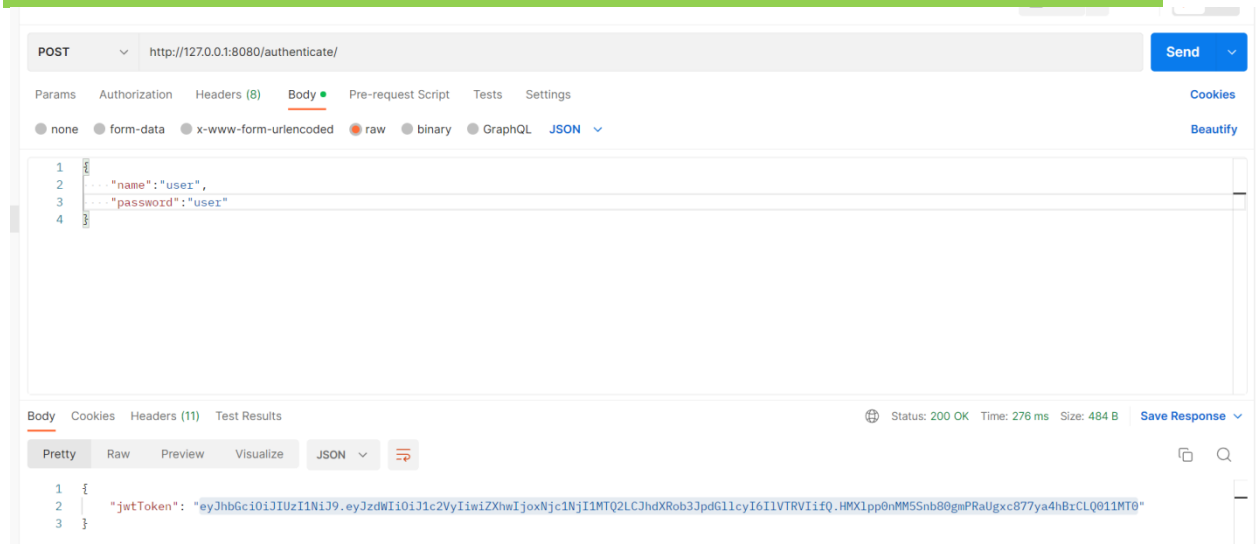
```
package
com.example.spring_security_jwt_without_oauth.servi
ces;

import
```

```
com.example.spring_security_jwt_without_oauth.entit
ies.User;
import
com.example.spring_security_jwt_without_oauth.enums
.RolesAll;
import org.assertj.core.api.Assertions;
import org.junit.jupiter.api.Test;
import
org.springframework.beans.factory.annotation.Autowi
red;
import
org.springframework.boot.test.context.SpringBootTes
t;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class UserCRUDServiceTest {
    @Autowired
    private UserCRUDService userCRUDService;

    @Test
    void findAllUsers() {
        List<User> allUsers =
userCRUDService.findAllUsers();

Assertions.assertThat(allUsers.size()).isEqualTo(1)
;
        System.out.println(allUsers);
    }

    @Test
    void saveUser() {
        User newUser = new User(
                "user",

"$2a$12$1Afgpcbxwgvc5Qj71/4gJeuC9bcoHl21vdLktqEThuz
hL9TwmkDKG",
                RolesAll.USER
        );
        //User newAdmin = new User("admin",
"admin", RolesAll.ADMIN);
        User user =
```

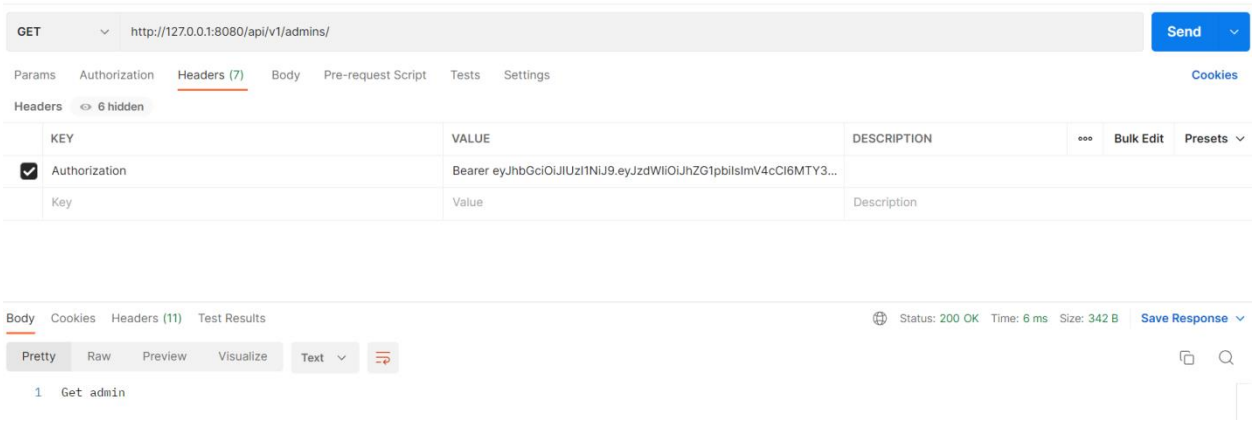
```
userCRUDService.saveUser(newUser);

Assertions.assertThat(user).isEqualTo(newUser);
    }
    @Test
    void saveAdmin() {
        User newAdmin = new User(
                "admin",

"$2a$12$tPu4fDXy3qjFvM2o4.0MCOTOdX0IMHKP/krZ8dPmxBFAlPU2P8BNi",
                RolesAll.ADMIN
        );
        User user =
userCRUDService.saveUser(newAdmin);

Assertions.assertThat(user).isEqualTo(newAdmin);
    }
}
```

17. Postman request settings

Get token for user



Get request with bearer token


GET
http://127.0.0.1:8080/api/v1/admins/
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
6 hidden
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
Authorization
Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTY3...
Key
Value
Description

Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 6 ms
Size: 342 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1 Get admin