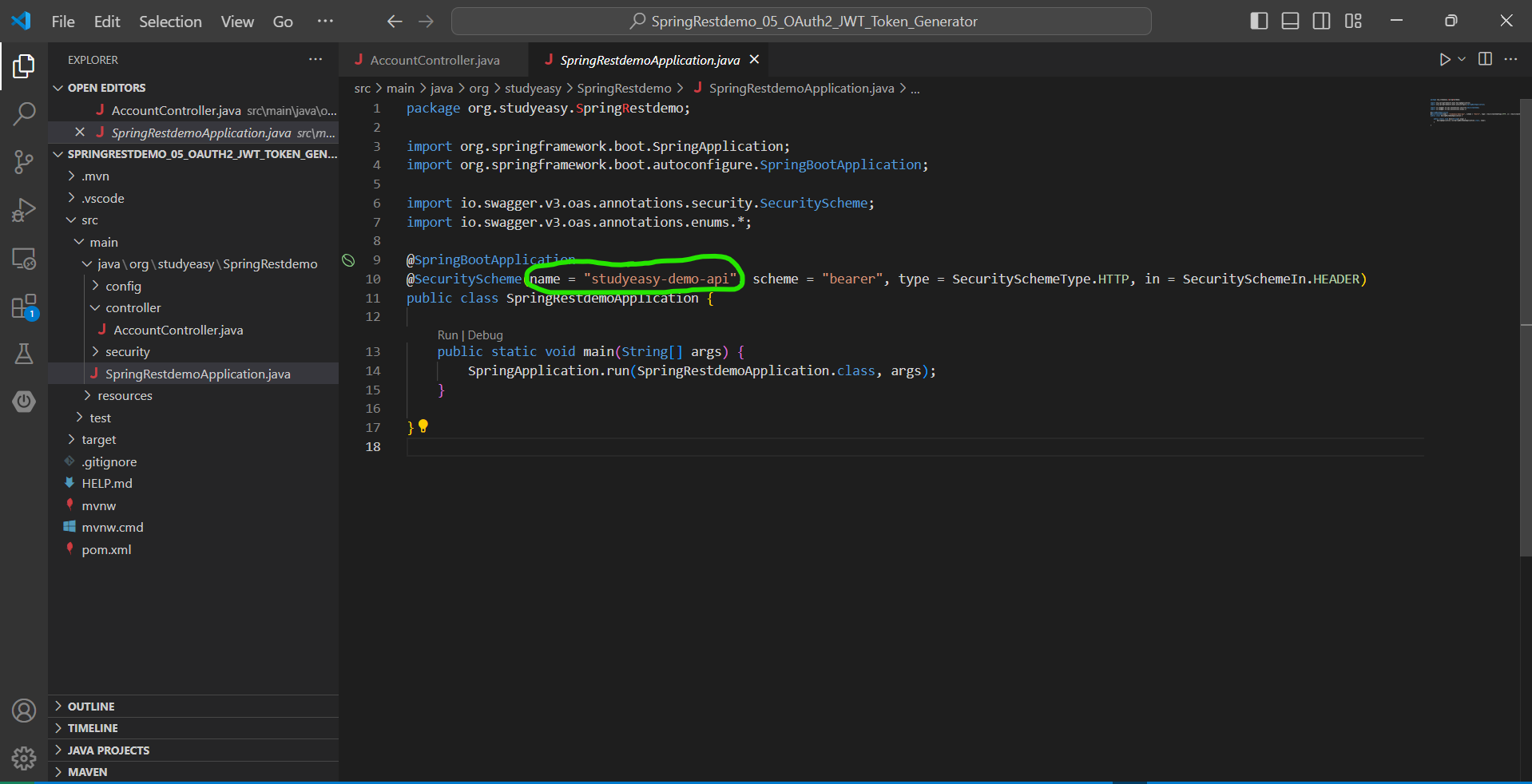
**Java Spring Boot Restful – OAuth2 – JWT – Token Generator - Continues**

How to secure?

 @SecurityRequirement(name = "studyeasy-demo-api")

****

**Go to AccountController -> modify code**

@GetMapping("/test")

    @SecurityRequirement(name = "studyeasy-demo-api")

    public String test(){

        return "Test api..";

    }

**Tag – Add some description or name to a specific tag.**

@Tag(name = "Test", description = "The Test API.")

 @GetMapping("/test")

    @Tag(name = "Test", description = "The Test API.")

    @SecurityRequirement(name = "studyeasy-demo-api")

    public String test(){

        return "Test api..";

    }

**Authentication Manager**

@Bean

public AuthenticationManager authManager(UserDetailsService userDetailsService) {

      var authProvider = new DaoAuthenticationProvider();

       authProvider.setUserDetailsService(userDetailsService);

       return new ProviderManager(authProvider);

}

**Create a controller in controller folder -> AuthController.java and in order to create a token you need to generate a service as well -> new folder -> service**

**New file -> TokenService.java**

**TokenService.java**

package org.studyeasy.SpringRestdemo.service;

import java.time.Instant;

import java.time.temporal.ChronoUnit;

import java.util.stream.Collectors;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.oauth2.jwt.JwtClaimsSet;

import org.springframework.security.oauth2.jwt.JwtEncoder;

import org.springframework.security.oauth2.jwt.JwtEncoderParameters;

public class TokenService {

    private final JwtEncoder encoder;

    public TokenService(JwtEncoder encoder){

        this.encoder = encoder;

    }

    public String generateToken(Authentication authentication){

        Instant now = Instant.now();

        String scope = authentication.getAuthorities().stream()

                       .map(GrantedAuthority::getAuthority)

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

        JwtClaimsSet claims = JwtClaimsSet.builder()

                              .issuer("self")

                              .issuedAt(now)

                              .expiresAt(now.plus(1, ChronoUnit.HOURS))

                              .subject(authentication.getName())

                              .claim("scope", scope)

                              .build();

        return this.encoder.encode(JwtEncoderParameters.from(claims)).getTokenValue();

    }

}

**AuthController.java**

package org.studyeasy.SpringRestdemo.controller;

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

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

@RestController

public class AuthController {

    @PostMapping("/token")

    public String token(){

    }

}

**----------------------------------------------------xxxxxxxx---------------------------------------------**

**Using TokenService -> Implement AuthService**

**Payload -> Create payload folder -> Create auth folder -> create file -> UserLogin.java -> and make it as record**

**UserLogin.java**

package org.studyeasy.SpringRestdemo.payload.auth;

public record UserLogin(String username, String password) {

}

**Payload -> Create payload folder -> Token.java and make it as record**

package org.studyeasy.SpringRestdemo.payload;

public record Token(String token) {

}

**Change the type to token and return statement**

@PostMapping("/token")

@ResponseBody

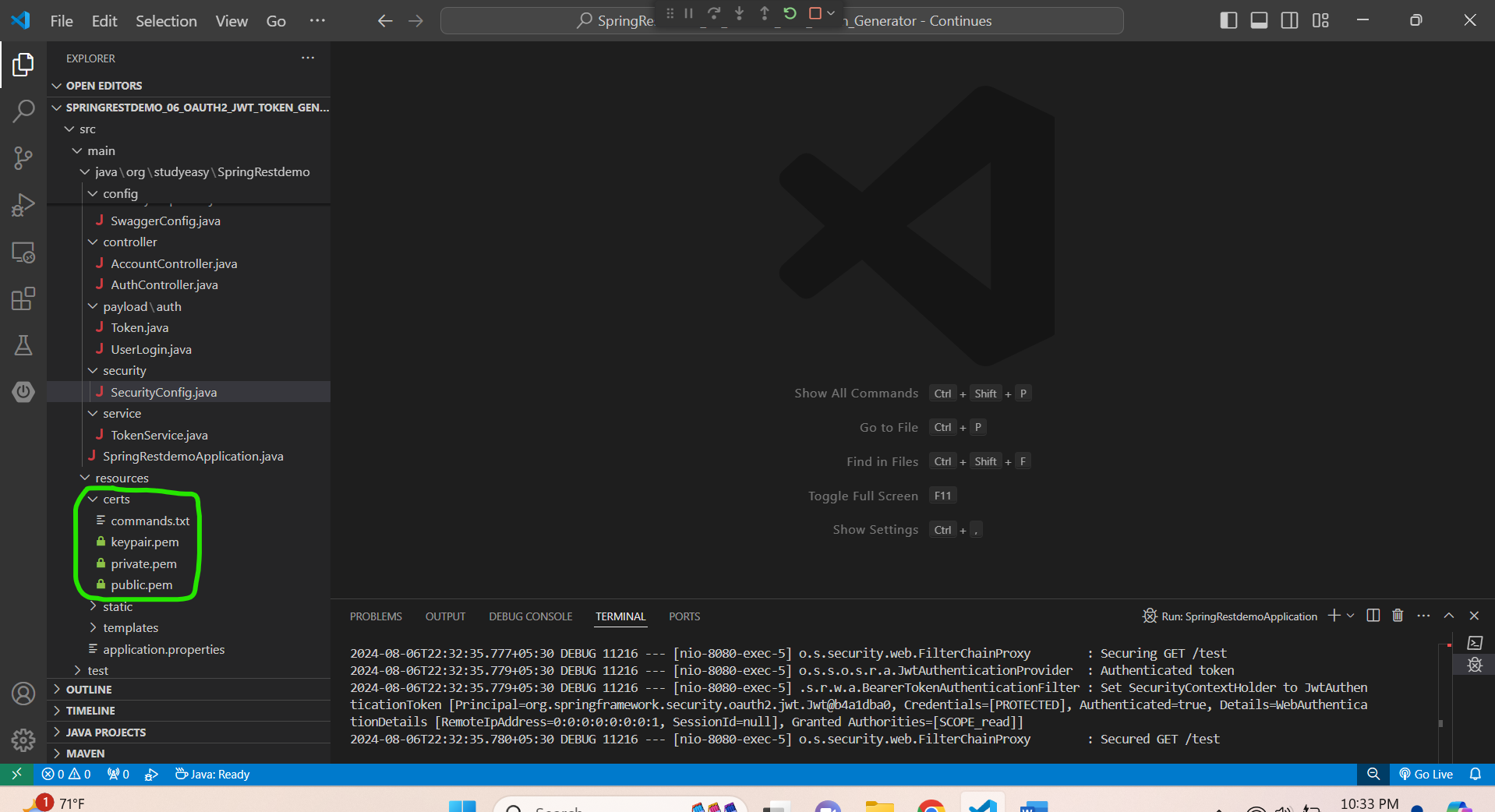
public String token(@RequestBody UserLogin userLogin) throws AuthenticationException{

    Authentication authentication = authenticationManager

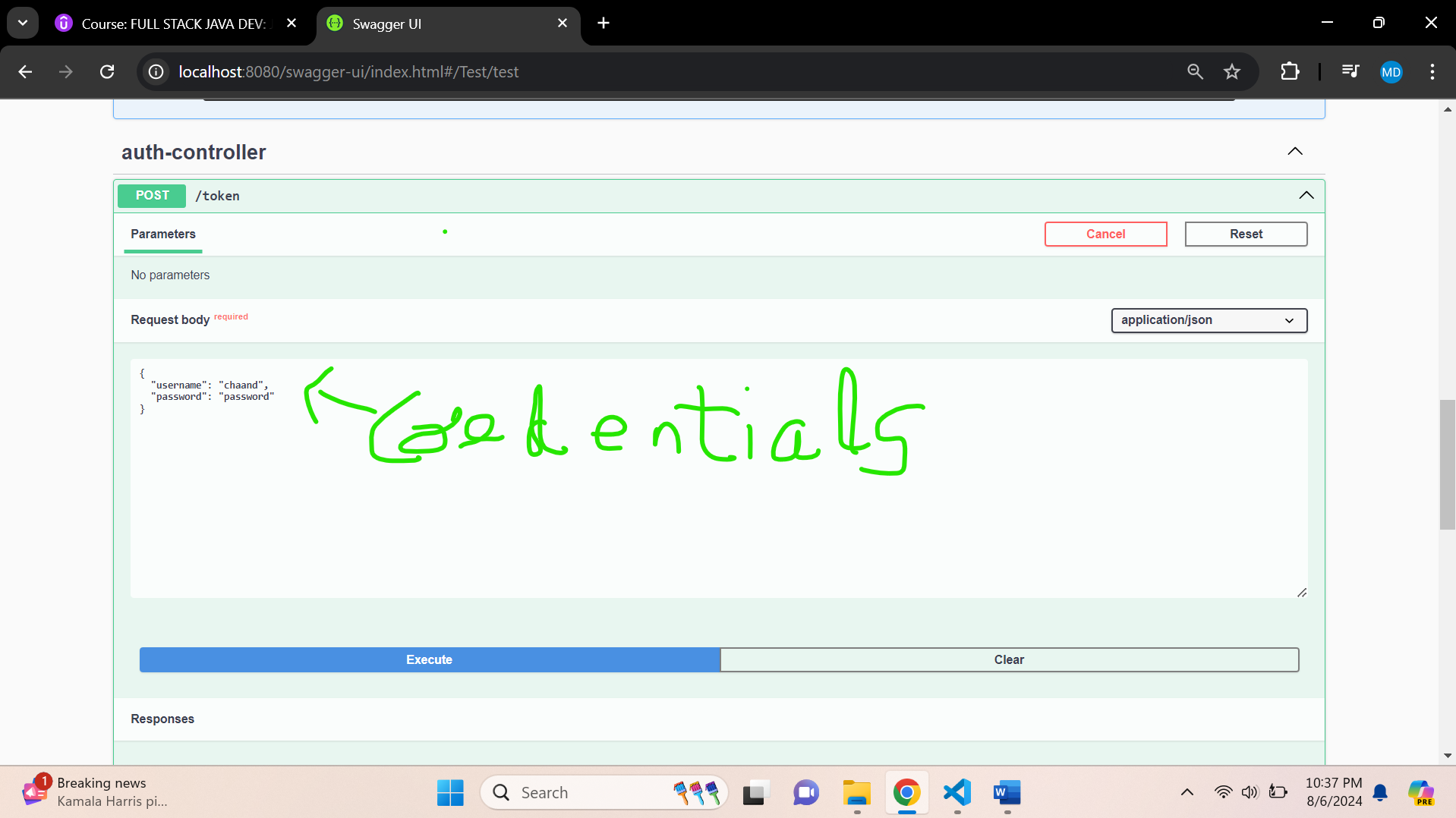
   .authenticate(new UsernamePasswordAuthenticationToken(userLogin.username(), userLogin.password()));

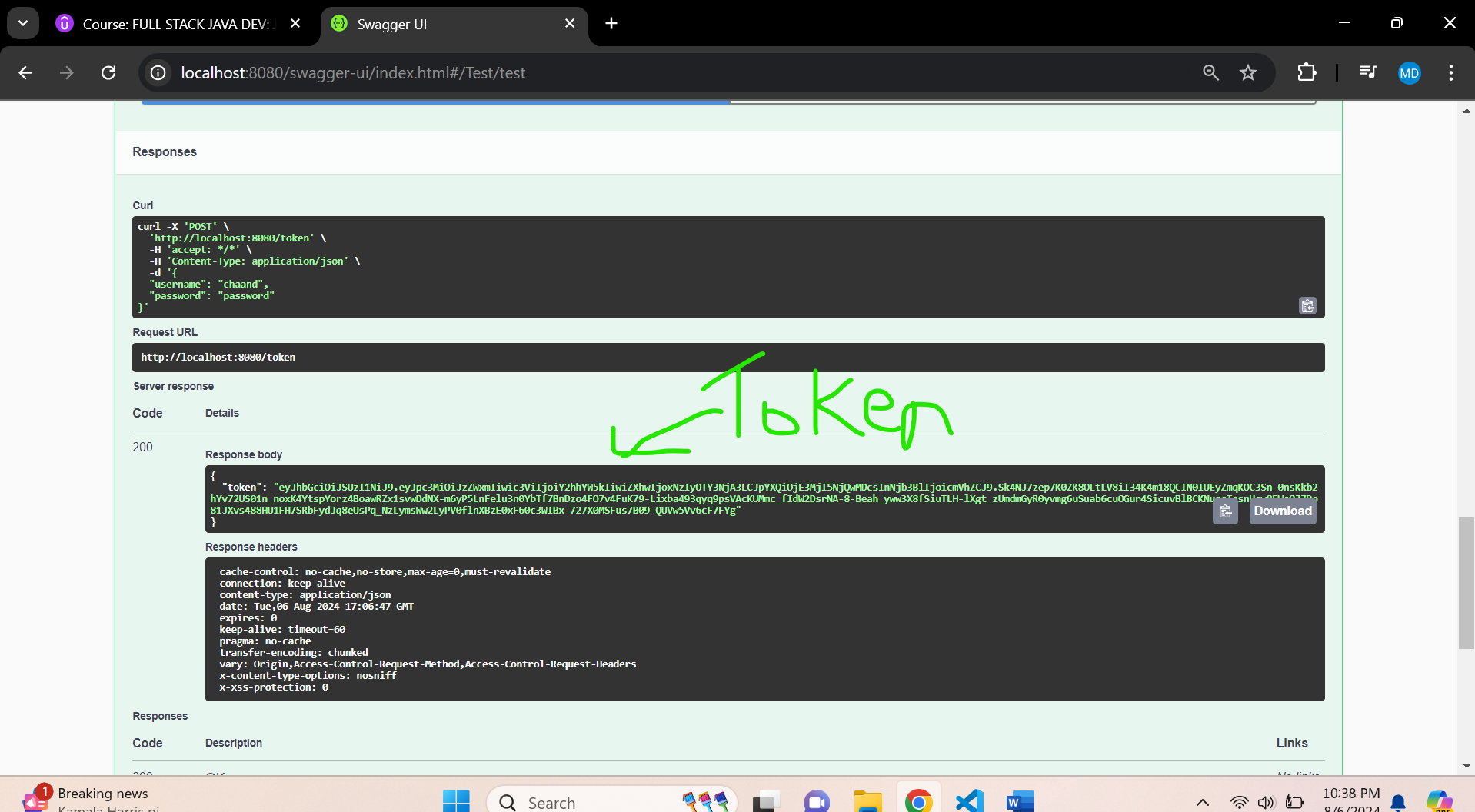
    }

**Output**

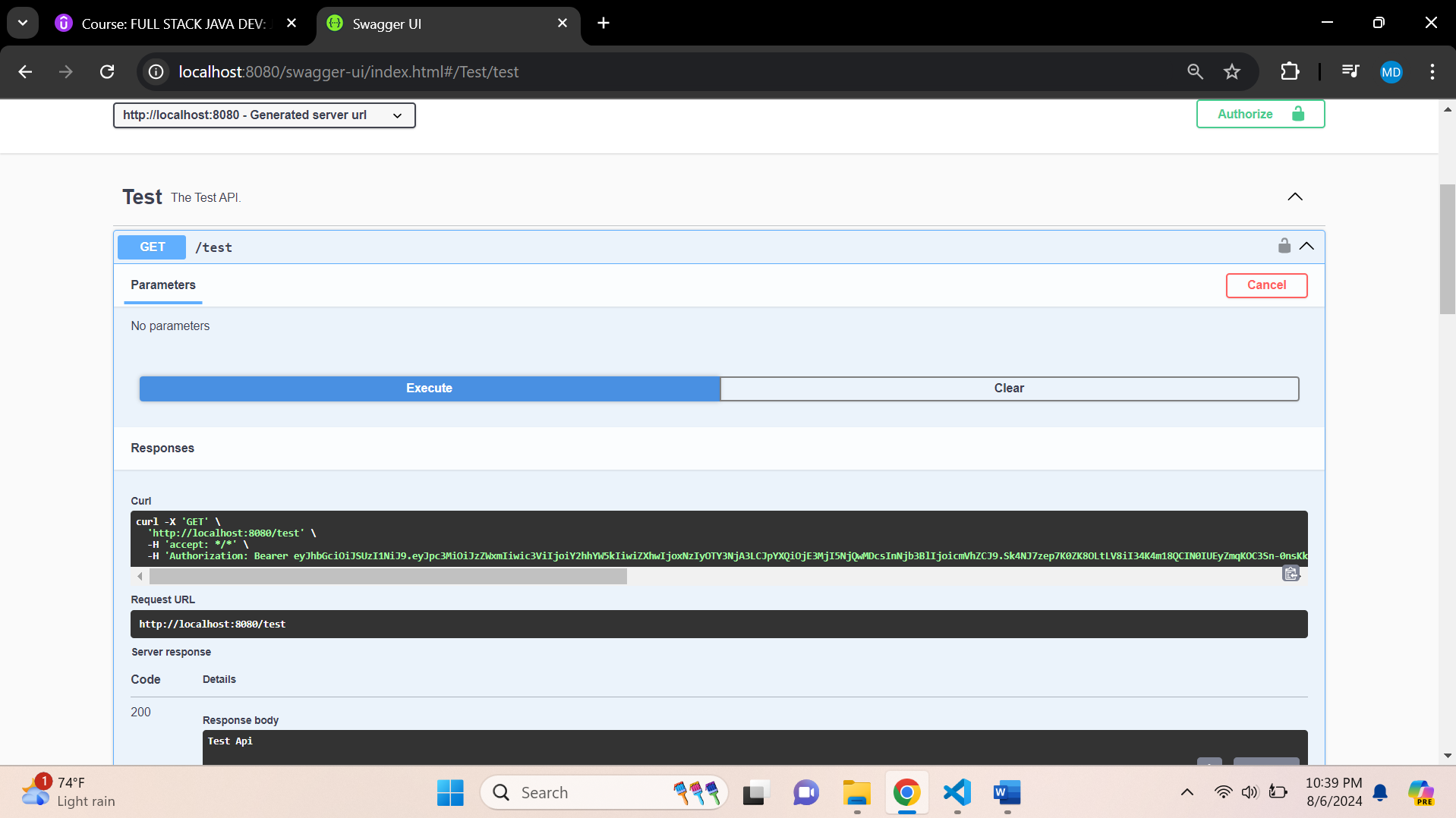
****

**Auth Controller**

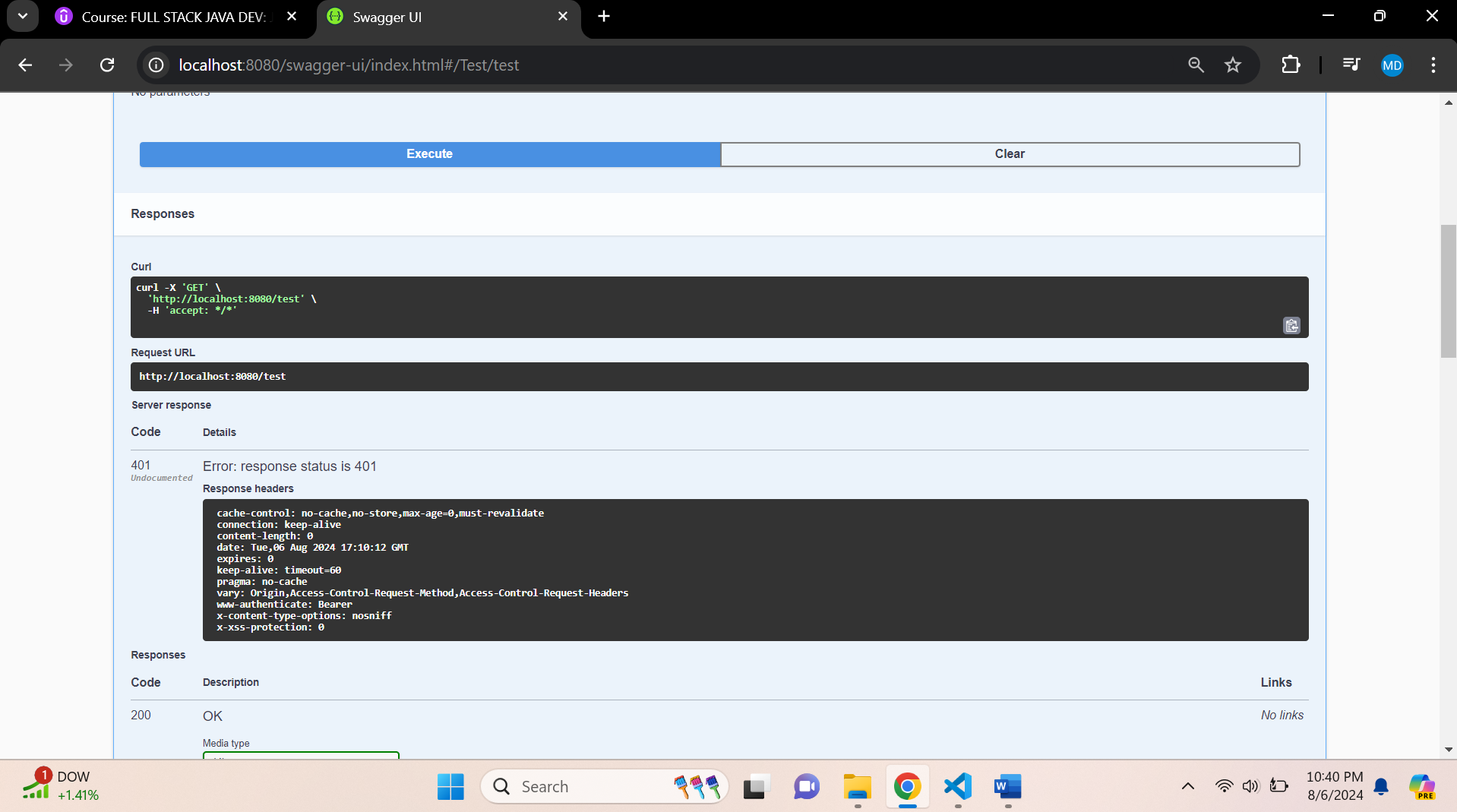
****

****

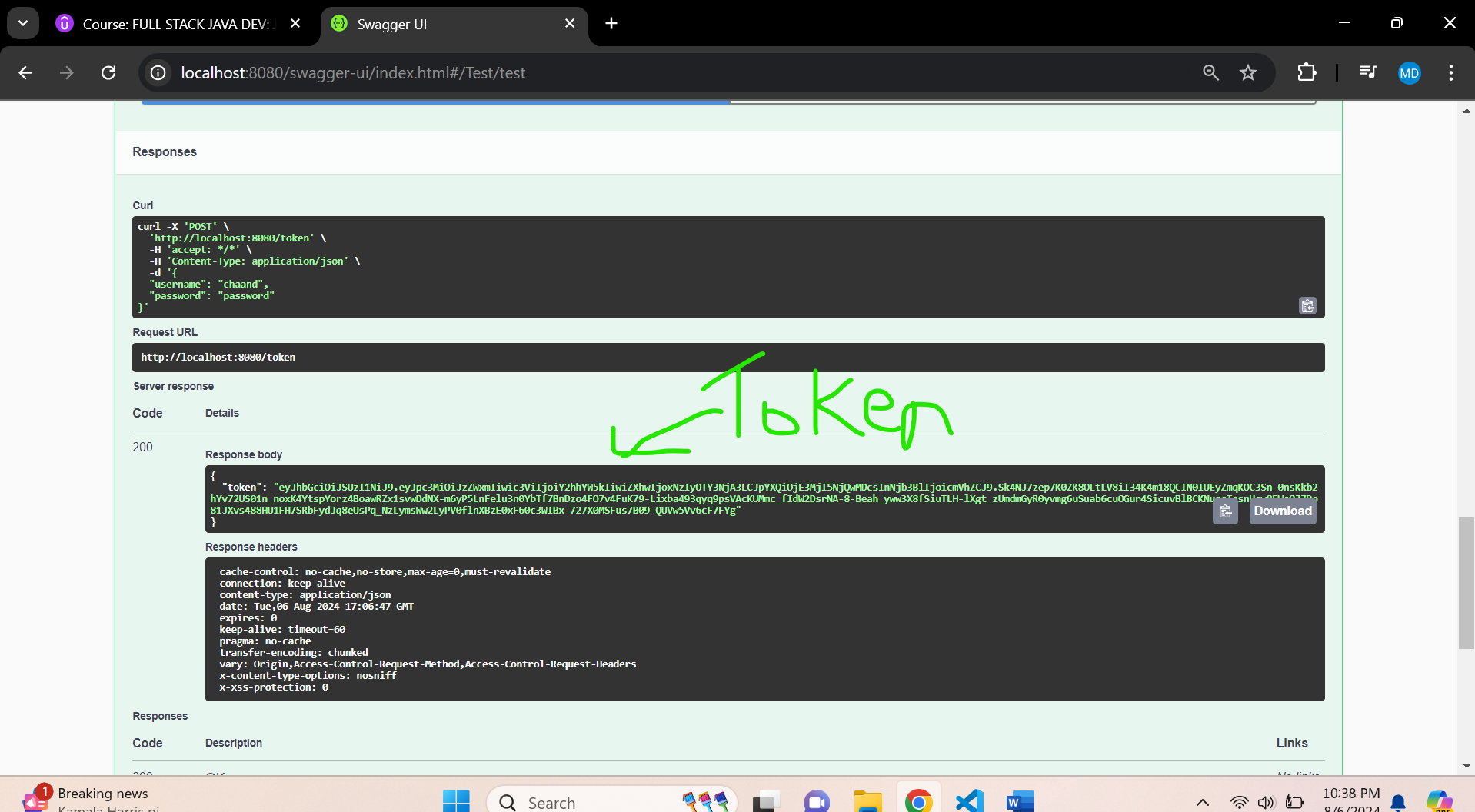
**Click on execute button -> with no credentials -> and ypu get 401**

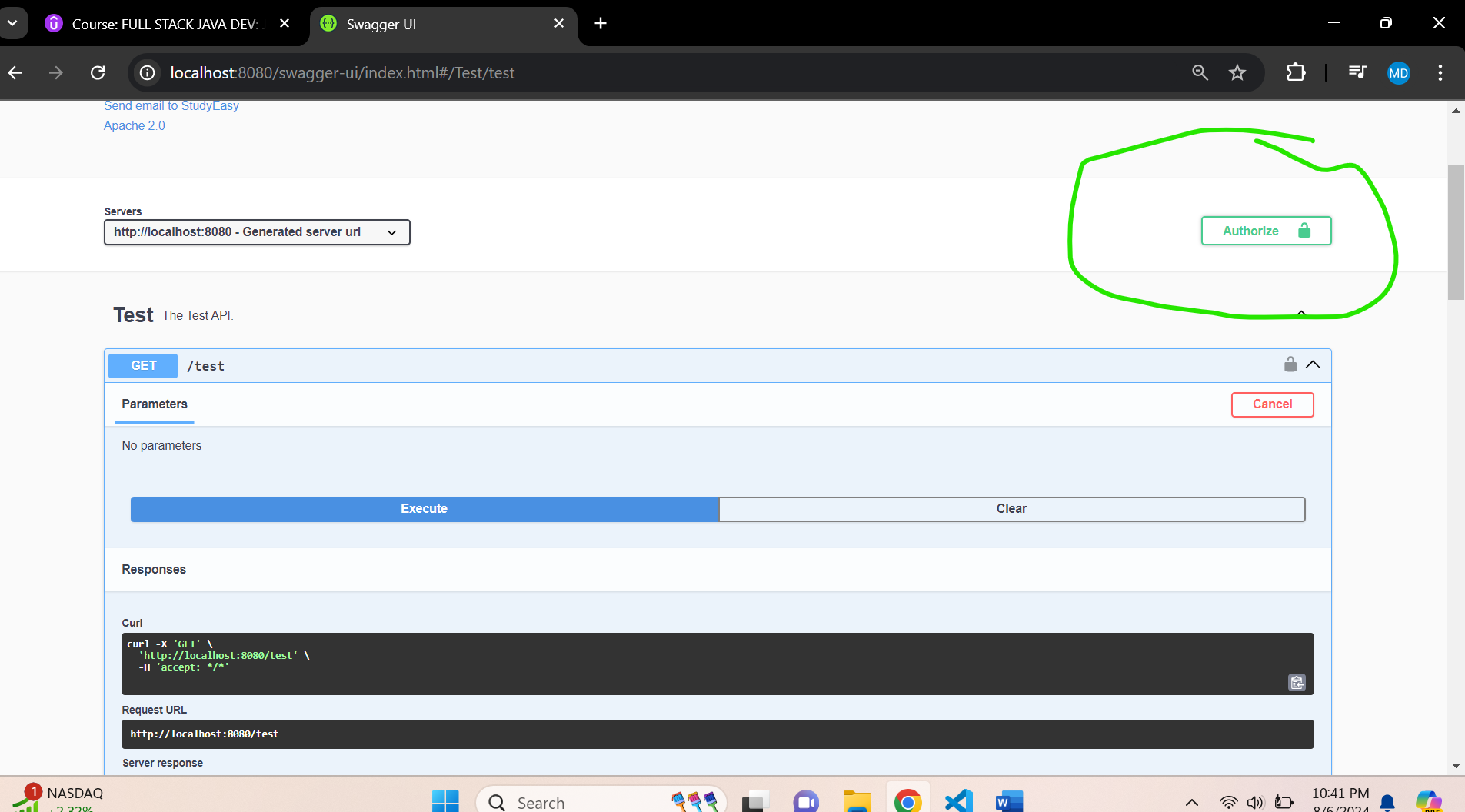
****

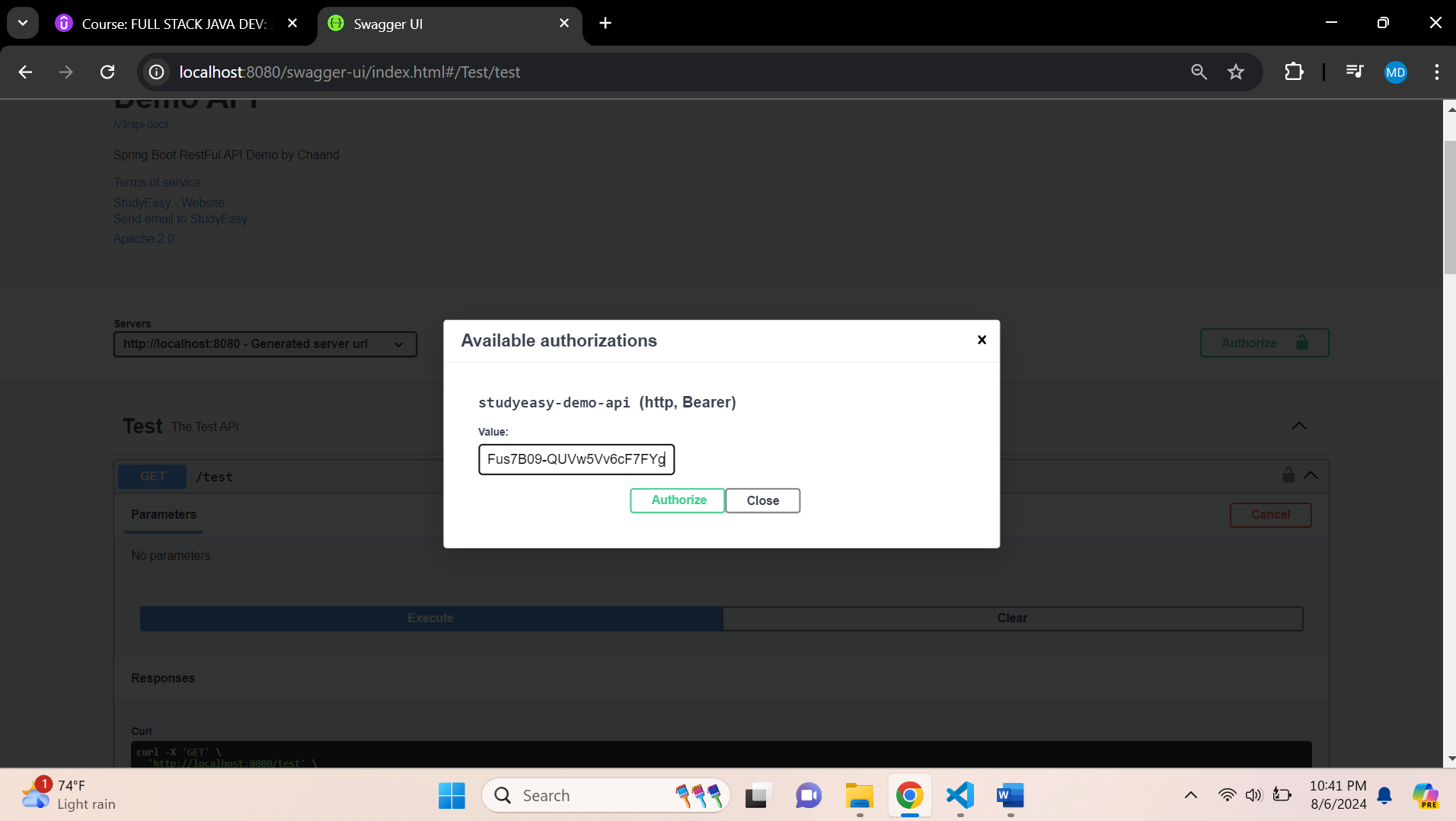
**You get error**

****

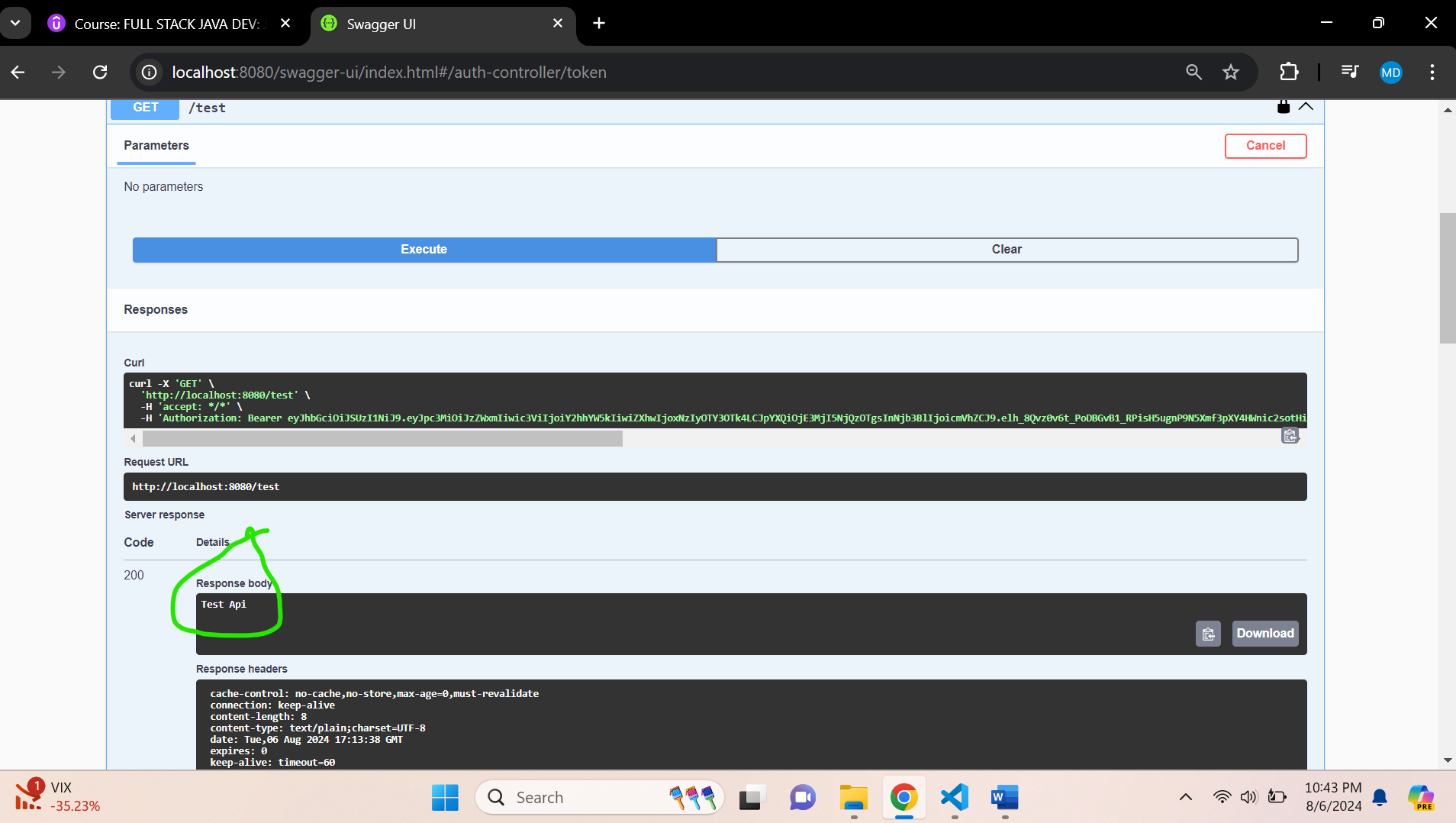
**Use the token generated in the below and paste it in authorize field like below**

****

****

****

**Click on authorize button**

****

**The major error was caused due to invalid password like below**

 @Bean

 public InMemoryUserDetailsManager users() {

    return new InMemoryUserDetailsManager(

          User.withUsername("chaand")

        .password("{noop}password")

          .authorities("read")

          .build());

    }

**{noop} -> no encoding**

**SecurityConfig.java**

package org.studyeasy.SpringRestdemo.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.ProviderManager;

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

import org.springframework.security.config.Customizer;

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

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

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

import org.springframework.security.core.userdetails.User;

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

import org.springframework.security.oauth2.jwt.JwtDecoder;

import org.springframework.security.oauth2.jwt.JwtEncoder;

import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;

import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

import org.studyeasy.SpringRestdemo.config.RsaKeyProperties;

import com.nimbusds.jose.jwk.JWK;

import com.nimbusds.jose.jwk.JWKSet;

import com.nimbusds.jose.jwk.RSAKey;

import com.nimbusds.jose.jwk.source.ImmutableJWKSet;

import com.nimbusds.jose.jwk.source.JWKSource;

import com.nimbusds.jose.proc.SecurityContext;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    private final RsaKeyProperties rsaKeys;

    public SecurityConfig(RsaKeyProperties rsaKeys) {

        this.rsaKeys = rsaKeys;

    }

    @Bean

    public InMemoryUserDetailsManager users() {

        return new InMemoryUserDetailsManager(

                User.withUsername("chaand")

                        .password("{noop}password")

                        .authorities("read")

                        .build());

    }

    @Bean

    public AuthenticationManager authManager(UserDetailsService userDetailsService) {

        var authProvider = new DaoAuthenticationProvider();

        authProvider.setUserDetailsService(userDetailsService);

        return new ProviderManager(authProvider);

    }

    @Bean

    JwtDecoder jwtDecoder() {

        return NimbusJwtDecoder.withPublicKey(rsaKeys.publicKey()).build();

    }

    @Bean

    JwtEncoder jwtEncoder() {

        JWK jwk = new RSAKey.Builder(rsaKeys.publicKey()).privateKey(rsaKeys.privateKey()).build();

        JWKSource<SecurityContext> jwks = new ImmutableJWKSet<>(new JWKSet(jwk));

        return new NimbusJwtEncoder(jwks);

    }

    @Bean

    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http

                .authorizeHttpRequests(authorize -> authorize

                        .requestMatchers("/token").permitAll()

                        .requestMatchers("/").permitAll()

                        .requestMatchers("/swagger-ui/\*\*").permitAll()

                        .requestMatchers("/v3/api-docs/\*\*").permitAll()

                        .requestMatchers("/test").authenticated())

                .oauth2ResourceServer(oauth2 -> oauth2

                        .jwt(Customizer.withDefaults()))

                .sessionManagement(session -> session

                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))

                .csrf(csrf -> csrf.disable())

                .headers(headers -> headers

                        .frameOptions(frameOptions -> frameOptions.disable()));

        return http.build();

    }

}