**Quản lý sản phẩm (Training Spring Boot)**

Mục lục

[1. Tổng quan 3](#_Toc91492041)

[1.1 Giới thiệu về project 3](#_Toc91492042)

[1.2 Biểu đồ cơ sở dữ liệu 4](#_Toc91492043)

[2. Các chức năng chính 4](#_Toc91492044)

[2.1 JWT Authentication 5](#_Toc91492045)

[2.1.1 JWT là gì ? 5](#_Toc91492046)

[2.1.2 Áp dụng JWT Authenticate vào project 9](#_Toc91492047)

[2.2 Spring JPA 16](#_Toc91492048)

[2.2.1 Spring JPA là gì ? 16](#_Toc91492049)

[2.2.2 Tại sao nên sử dụng Spring JPA ? 16](#_Toc91492050)

[2.2.3 Thêm các dependencies vào file pom.xml 17](#_Toc91492051)

[2.2.4 Tạo các class tương ứng với các bảng trong database 17](#_Toc91492052)

[2.2.5 Tạo các Repository tương ứng 19](#_Toc91492053)

[ProductRepository 19](#_Toc91492054)

[OrderRepository 19](#_Toc91492055)

[2.2.6 Tạo các Service 20](#_Toc91492056)

[2.2.7 Tạo ra các RestController 22](#_Toc91492057)

[2.3 Logging 24](#_Toc91492058)

[2.3.1 Logging là gì ? 24](#_Toc91492059)

[2.3.2 Các thông tin trong log 24](#_Toc91492060)

[2.3.3 Các mức độ của log (Logging Level) theo thứ tự tăng dần 24](#_Toc91492061)

[2.3.4 Cấu hình logback 24](#_Toc91492062)

[2.4 Swagger 26](#_Toc91492063)

[2.4.1 Swagger là gì ? 26](#_Toc91492064)

[2.4.2 Cấu hình Swagger 26](#_Toc91492065)

[2.5 Email 28](#_Toc91492066)

[2.5.1 Thêm các dependencies 28](#_Toc91492067)

[2.5.2 Cấu hình thông tin trong qua class MyConstants 29](#_Toc91492068)

[2.5.3 Tạo email service 29](#_Toc91492069)

[2.5.4 Tạo email controller 30](#_Toc91492070)

[3. Tài liệu tham khảo 30](#_Toc91492071)

## Tổng quan

### Giới thiệu về project

* Project quản lý sản phẩm với các chức năng cơ bản như:
* Thêm, sửa, xóa sử dụng spring JPA
* Xác thực sử dụng JWT và spring security
* Email
* Swagger
* Logging

### Biểu đồ cơ sở dữ liệu

## Các chức năng chính

### JWT Authentication

#### 2.1.1 JWT là gì ?

* JWT là viết tắt của Json Web Token, là một tiêu chuẩn mở (RFC 7519) định nghĩa một cách gọn nhẹ và khép kín cho việc truyền gửi dữ liệu an toàn giữa các bên như một đối tượng JSON. Thông tin này có thể được kiểm chứng và đáng tin cậy bởi chữ ký số của nó. Client sẽ cần phải xác minh với Server bằng việc sử dụng chứng chỉ 1 lần. Trong suốt thời gian Server xác minh chứng chỉ và gửi trả cho Client JWT. Cho tất cả những yêu cầu tới, Client có thể xác thực bản thân với Server bằng việc sử dụng JWT và nó không cần phải gửi chứng chỉ đi như username và password.
* Ví dụ về JWT Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjEzODY4OTkxMzEsImlzcyI6ImppcmE6MTU0ODk1OTUiLCJxc2giOiI4MDYzZmY0Y2ExZTQxZGY3YmM5MGM4YWI2ZDBmNjIwN2Q0OTFjZjZkYWQ3YzY2ZWE3OTdiNDYxNGI3MTkyMmU5IiwiaWF0IjoxMzg2ODk4OTUxfQ.uKqU9dTB6gKwG6jQCuXYAiMNdfNRw98Hw\_IWuA5MaMo

* Cấu trúc của một JWT

<base64-encoded header>.<base64-encoded payload>.<base64-encoded signature>

* Header bao gồm hai phần chính: loại token (mặc định là JWT - Thông tin này cho biết đây là một Token JWT) và thuật toán đã dùng để mã hóa (HMAC SHA256 - HS256 hoặc RSA).

{

"alg": "HS256",

"typ": "JWT"

}

* Payload chứa các claims. Claims là một các biểu thức về một thực thể (chẳng hạn user) và một số metadata phụ trợ.

{

"iss": "jira:1314039", //issuer

"iat": 1300819370, //issued at time

"exp": 1300819380, //expiration time

"qsh": "8063ff4ca1e41df7bc90c8ab6d0f6207d491cf6dad7c66ea797b4614b71922e9",

"sub": "batman", //subject

"context": {

"user": {

"userKey": "batman",

"username": "bwayne",

"displayName": "Bruce Wayne"

}

}

}

* Chữ ký Signature trong JWT là một chuỗi được mã hóa bởi header, payload cùng với một chuỗi bí mật theo nguyên tắc sau:

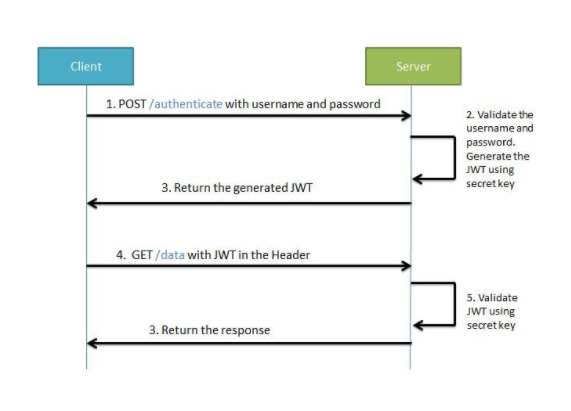
HMACSHA256(

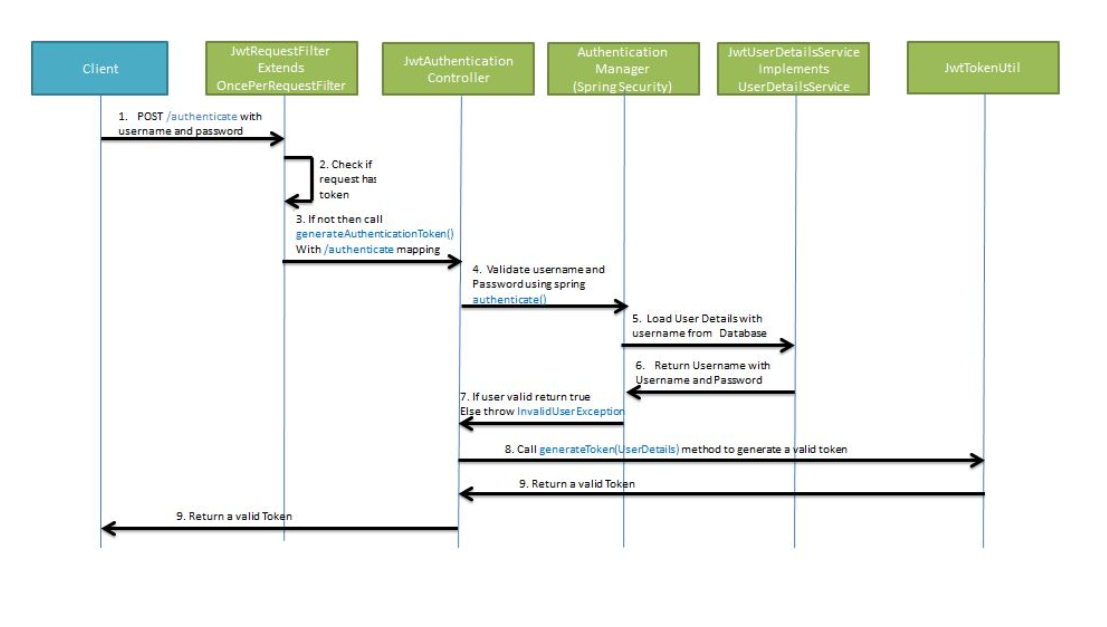
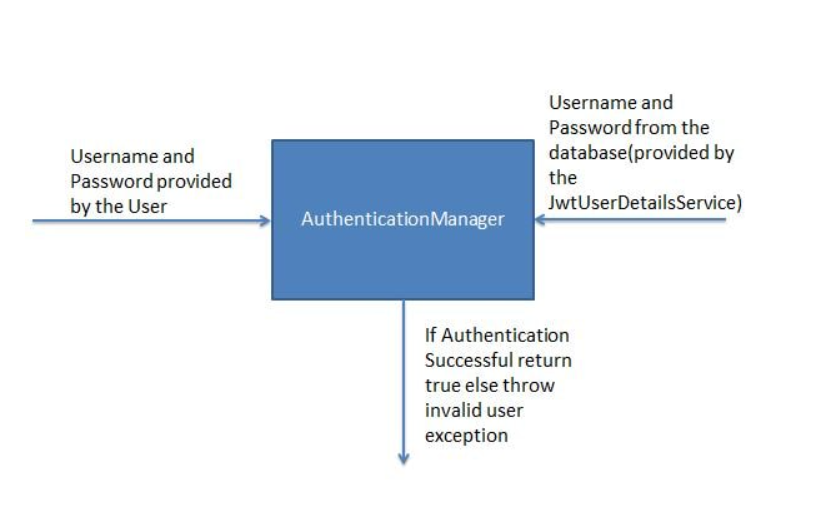
base64UrlEncode(header) + "." +

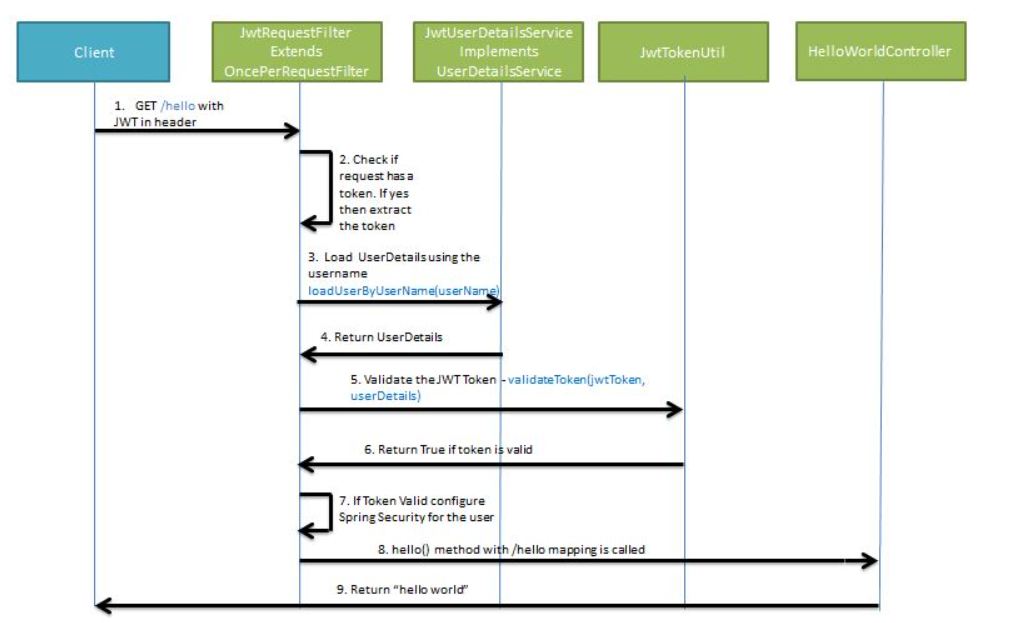
base64UrlEncode(payload),

secret)

* Mô hình hoạt động của JWT Authenticate



* Khởi tạo JWT
* Xác minh JWT



#### 2.1.2 Áp dụng JWT Authenticate vào project

* Thêm các dependencies vào file pom.xml

//spring security  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 <version>RELEASE</version>  
 <scope>compile</scope>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.1</version>  
</dependency>  
//library of @Valid  
<dependency>  
 <groupId>javax.validation</groupId>  
 <artifactId>validation-api</artifactId>  
 <version>2.0.1.Final</version>  
</dependency>

* Định nghĩa secret key trong application.properties. Secret key kết hợp với header và payload để tạo 1 hash riêng. Ta chỉ có thể xác minh cái hash này nếu như có secret key

jwt.secret=aladitech

* Tạo class JwtTokenUtil: class này chịu trách nhiệm cho việc thực hiện hoạt động của JWT như khởi tạo và xác minh

@Component  
public class JwtTokenUtil implements Serializable {  
   
 private static final long *serialVersionUID* = -2550185165626007488L;  
  
 //expired time of token  
 public static final long *JWT\_TOKEN\_VALIDITY* = 5\* 60\*60 ;  
  
 //get value from application.properties  
 @Value("${jwt.secret}")  
 private String secret;  
  
 //get username from jwt token  
 public String getUsernameFromToken(String token){  
 return getClaimFromToken(token, Claims::getSubject);  
 }  
  
 //get expiration date from jwt token  
 public Date getExpirationFromToken(String token){  
 return getClaimFromToken(token, Claims::getExpiration);  
  
 }  
  
 //get claim from token  
 public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver){  
 final Claims claims=getAllClaimsFromToken(token);  
 return claimsResolver.apply(claims);  
 }

//for retrieving any information from token will need the secret key  
 private Claims getAllClaimsFromToken(String token){  
// System.out.println("Body: "+Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody());  
 return Jwts.*parser*().setSigningKey(secret).parseClaimsJws(token).getBody();  
 }  
  
 //check if the token has expired  
 private Boolean isTokenExpired(String token){  
 final Date expiration=getExpirationFromToken(token);  
 return expiration.before(new Date());  
 }  
  
 //generate token for user  
 public String generateToken(UserDetails userDetails){  
 Map<String, Object> claims= new HashMap<>();  
 return doGenerateToken(claims,userDetails.getUsername());  
 }  
  
 */\*\*  
 \* While creating the token:  
 \* 1. Define claims of the token, like Issuer, Expiration, Subject,ID  
 \* 2. Sign the JWT using the HS512 algorithm and secret key  
 \* 3. According to JWT Compact Serialization(http://tool.ietf.org/html/draft-ietf-jose-json-web-singature-41#section3.1  
 \* compaction of the JWT to a URL-safe string  
 \*  
 \*\*/* public String doGenerateToken(Map<String,Object> claims, String subject){  
// System.out.println("do generateToken: "+Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))  
// .setExpiration(new Date(System.currentTimeMillis()+JWT\_TOKEN\_VALIDITY\*1000))  
// .signWith(SignatureAlgorithm.HS512,secret).compact());  
 return Jwts.*builder*().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.*currentTimeMillis*()))  
 .setExpiration(new Date(System.*currentTimeMillis*()+*JWT\_TOKEN\_VALIDITY*\*1000))  
 .signWith(SignatureAlgorithm.*HS512*,secret).compact();  
 }  
  
 //validate token  
 public Boolean validateToken(String token, UserDetails userDetails){  
 final String username=getUsernameFromToken(token);  
 return (username.equals(userDetails.getUsername()));  
 }  
}

* Tạo class JWTUserDetailsService: class này kế thừa từ interface UserDetailServices của Spring Security. Nó sẽ Override phương thức loadUserByUsername để lấy thông tin của người dùng từ trong database qua username.

@Service  
public class JwtUserDetailsService implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userDao;  
  
 //get user from database using username  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 DAOUser user = userDao.findByUsername(username);  
 if (user == null) {  
 throw new UsernameNotFoundException("Username not found with username: " + username);  
 }  
 return new User(user.getUsername(), user.getPassword(), new ArrayList<>());  
 }  
  
 public DAOUser save(UserDTO user) {  
 DAOUser newUser = new DAOUser();  
 newUser.setUsername(user.getUsername());  
 newUser.setPassword(new BCryptPasswordEncoder().encode(user.getPassword()));  
 newUser.setAddress(user.getAddress());  
 newUser.setAge(user.getAge());  
 newUser.setFirstName(user.getFirstName());  
 newUser.setLastName(user.getLastName());  
 return userDao.save(newUser);  
 }  
  
}

* Tạo class JWT Request: dùng để gửi username và password từ Client

package com.example.ss\_jwt\_mysql\_helloworld.model;  
  
import lombok.Data;  
  
import java.io.Serializable;  
  
//using lombok   
@Data  
public class JwtRequest implements Serializable {  
 private static final long *serialVersionUID* = 5926468583005150707L;  
 private String username;  
 private String password;  
}

* Tạo class JwtResponse để nhận token trả về

package com.example.ss\_jwt\_mysql\_helloworld.model;  
  
import lombok.Data;  
//using lombok  
@Data  
public class JwtResponse {  
 private static final long *serialVersionUID* = -8091879091924046844L;  
 private final String jwtToken;  
}

* Tạo class JwtAuthenticationController: tạo api để người dùng request authenticate

@RestController  
@CrossOrigin  
public class JwtAuthenticationController {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private JwtTokenUtil jwtTokenUtil;  
  
 @Autowired  
 private JwtUserDetailsService userDetailsService;  
  
  
 @Operation(summary = "Create token")  
 @RequestMapping(value = "/authenticate", method = RequestMethod.*POST*)  
 public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) throws Exception {  
  
 authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());  
  
 //get user from database  
 final UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());  
  
 //generate token  
 final String token = jwtTokenUtil.generateToken(userDetails);  
  
 //return token  
 return ResponseEntity.*ok*(new JwtResponse(token));  
 }  
  
 //register new user -> save to database  
 //return user has already saved  
 @Operation(summary = "Register user")  
 @RequestMapping(value = "/register", method = RequestMethod.*POST*)  
 public ResponseEntity<?> saveUser(@RequestBody UserDTO user) throws Exception {  
 return ResponseEntity.*ok*(userDetailsService.save(user));  
 }  
  
 //athenticate  
 private void authenticate(String username, String password) throws Exception {  
 try {  
 authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));  
 } catch (DisabledException e) {  
 throw new Exception("USER\_DISABLED", e);  
 } catch (BadCredentialsException e) {  
 throw new Exception("INVALID\_CREDENTIALS", e);  
 }  
 }  
}

* Tạo class JwtRequestFilter: kế thừa Spring Web Filter OncePerRequestFilter class. Với tất cả những request được gửi đến thì class Filter này sẽ thực thi. Nó sẽ kiểm tra request này có JWT Token hợp lệ hay không. Nếu hợp lệ thì nó sẽ set Authentication trong context nhằm chỉ ra người dùng hiện tại này đã được xác minh

@Component  
public class JwtRequestFilter extends OncePerRequestFilter {  
  
 @Autowired  
 private JwtUserDetailsService jwtUserDetailService;  
  
 @Autowired  
 private JwtTokenUtil jwtTokenUtil;  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  
  
 final String requestTokenHeader = request.getHeader("Authorization");  
  
 //initial username, jwtToken  
 String username = null;  
 String jwtToken = null;  
  
 //JWT Token is in the form "Bearer token". Remove Bearer word and get only the Token  
 if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {  
 jwtToken = requestTokenHeader.substring(7);  
 try {  
 username = jwtTokenUtil.getUsernameFromToken(jwtToken);  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Unable to get JWT Token");  
 } catch (ExpiredJwtException e) {  
 System.*out*.println("JWT Token has expired");  
 }  
 } else {  
 logger.warn("JWT Token does not begin with Bearer String");  
 }  
  
 //Once we get the token validate it.  
 if (username != null && SecurityContextHolder.*getContext*().getAuthentication() == null) {  
 UserDetails userDetails = this.jwtUserDetailService.loadUserByUsername(username);  
  
 //if token is valid configure Spring Security to manually set authentication  
 if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {  
 UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(  
 userDetails, null, userDetails.getAuthorities());  
 usernamePasswordAuthenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
  
 //after setting the Authentication in the context, we specify that the current user is authenticated. So it  
 //pass the Spring Security Configurations successfully.  
 SecurityContextHolder.*getContext*().setAuthentication(usernamePasswordAuthenticationToken);  
  
 }  
 }  
 filterChain.doFilter(request, response);  
 }  
}

* Tạo class JwtAuthenticationEntryPoint: kế thừa AuthenticationEntryPoint của Sping. Nó sẽ trả về tất cả những request chưa được xác minh và trả về lỗi 401

`@Component  
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {  
  
 private static final long *serialVersionUID* = -7858869558953243875L;  
  
 @Override  
 public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {  
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, "Unauthorized");  
 }  
}

* Tạo class WebSecurityConfig: kế thừa WebSecurityConfigurerAdapter là class tiện lợi cho việc điều chỉnh WebSecurity và HttpSecurity

@Configuration  
@ConfigurationProperties(prefix = "spring")  
@EnableWebSecurity  
@Order(1)  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;  
  
 @Autowired  
 private UserDetailsService jwtUserDetailsService;  
  
 @Autowired  
 private JwtRequestFilter jwtRequestFilter;  
  
 @Autowired  
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
 // configure AuthenticationManager so that it knows from where to load  
 // user for matching credentials  
 // Use BCryptPasswordEncoder  
 auth.userDetailsService(jwtUserDetailsService).passwordEncoder(new BCryptPasswordEncoder());  
 }  
  
  
 @Bean  
 @Override  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Override  
 protected void configure(HttpSecurity httpSecurity) throws Exception {  
 // disable csrf  
 httpSecurity.csrf().disable()  
 // dont authenticate this particular request  
 .authorizeRequests().antMatchers("/authenticate", "/register","/swagger-ui/\*\*",  
 "/v3/api-docs/\*\*","/swagger-ui.html").permitAll().  
 // all other requests need to be authenticated  
 anyRequest().authenticated().and().  
 // make sure we use stateless session; session won't be used to  
 // store user's state.  
 exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()  
 .sessionCreationPolicy(SessionCreationPolicy.*STATELESS*);  
  
 // Add a filter to validate the tokens with every request  
 httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);  
 }  
}

### Spring JPA

#### 2.2.1 Spring JPA là gì ?

* Spring JPA là một đặc tả của Java dùng để quản lý các dữ liệu có quan hệ trong ứng dụng Java. Nó cho phép chúng ta truy cập và duy trì dữ liệu giữa Object và Class trong Java với cơ sở dữ liệu có quan hệ. JPA tuân theo Object-Relation-Mapping (ORM). Nó là cài đặt của các interface. Nó cũng cung cấp thời gian EntityManage API trong việc xử lý query và các giao dịch trên các đối tượng từ database. Nó sử dụng nền tảng câu lệnh hướng đối tượng không phụ thuộc JPQL (Java Persitent Query Language)

#### 2.2.2 Tại sao nên sử dụng Spring JPA ?

* JPA đơn giản hơn, gọn gàng hơn và ít tốn công sức hơn là JDBC, SQL hay ánh xạ viết tay. JPA thích hợp cho các ứng dụng phức tạp không tuân theo hiệu suất. Ưu điểm chính của JPA so với JDBC là: trong JPA, dữ liệu được biểu thị bằng các object và class trong khi trong JDBC dữ liệu được biểu diễn bằng các bảng và bản ghi. Nó sử dụng POJO để biểu diễn dữ liệu liên tục giúp đơn giản hóa việc lập trình database. Có một số ưu điểm khác của JPA:
* JPA tránh viết Ngôn Ngữ Định Nghĩa Dữ Liệu (DDL) bằng phương ngữ SQL dành riêng cho database. Thay vào đó nó cho phép ánh xạ trong XML hoặc sử dụng các annotation Java.
* JPA cho phép chúng ta tránh viết Ngôn Ngữ Thao Tác Dữ Liệu (DML) bằng phương ngữ SQL dành riêng cho database.
* JPA cho phép chúng ta save và load các object và đồ thị Java mà không cần bất kỳ ngôn ngữ DML nào.
* Khi chúng ta cần thực hiện các truy vấn JPQL, nó cho phép chúng ta viết các truy vấn dưới dạng các entity Java mà không phải là bảng và cột SQL.

#### 2.2.3 Thêm các dependencies vào file pom.xml

//JPA  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>

#### 2.2.4 Tạo các class tương ứng với các bảng trong database

**Class DAOUser**

@Data  
@Entity  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "DungTQ\_User")  
public class DAOUser implements Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Schema(title = "id of user", example = "1")  
 private Long id;  
  
 @Column(name = "username", nullable = false, unique = true)  
 @Schema(title = "username of user", example = "username")  
 private String username;  
  
 @Column(name = "password", nullable = false)  
 @Schema(title = "password of user", example = "password")  
 private String password;  
  
 @Column(name = "firstName")  
 @Schema(title = "firstName of user", example = "John")  
 private String firstName;  
  
 @Column(name = "lastName")  
 @Schema(title = "lastName of user", example = "Dung")  
 private String lastName;  
  
 @Column(name = "age", nullable = false)  
 @Schema(title = "age of user", example = "1")  
 private int age;  
  
 @Column(name = "address")  
 @Schema(title = "address of user", example = "Ha Noi")  
 private String address;  
  
\

**Class Company**

@Data  
@Entity  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Schema(description = "Company's information")  
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})  
@Table(name = "DungTQ\_Company")  
public class Company implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Schema(title = "ID of company", example = "1")  
 private Long id;  
  
 @Column(name = "companyName")  
 @Schema(title = "Name of company", example = "company1")  
 private String companyName;  
  
 @Column(name = "companyAddress")  
 @Schema(title = "Address of company", example = "Viet Nam")  
 private String companyAddress;  
  
 public Company(Long id, String companyName, String companyAddress) {  
 this.id = id;  
 this.companyName = companyName;  
 this.companyAddress = companyAddress;  
 }  
  
 public Company(Long id) {  
 this.id = id;  
 }  
 @JsonIgnore  
 @OneToMany(mappedBy = "company", cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*)  
// @EqualsAndHashCode.Exclude  
// @ToString.Exclude  
 private Collection<Category> categories;  
}

#### 2.2.5 Tạo các Repository tương ứng

##### ProductRepository

@Repository  
public interface ProductRepository extends JpaRepository<Product, Long> {  
}

##### OrderRepository

@Repository  
public interface OrderRepository extends JpaRepository<Order, Long> {  
 @Query(value = "select \* from DungTQ\_Order", nativeQuery = true)  
 List<Order> getAll\_Order();  
  
 @Query(value = "select o.id,o.ORDER\_DATE, p.PRODUCT\_NAME,DU.FIRST\_NAME, DU.LAST\_NAME from DungTQ\_Order o inner join DungTQ\_Product p on o.product\_id =p.id\n" +  
 "inner join DUNGTQ\_USER DU on o.USER\_ID = DU.ID ", nativeQuery = true)  
 List<Order> getAllDetailOrder();  
  
 @Query(value = "Insert into DUNGTQ\_ORDER(order\_date, product\_id, user\_id)\n" +  
 "VALUES ( TO\_DATE(:date,'YYYY/MM/DD'),:product\_id,:user\_id)", nativeQuery = true)  
 Order insertOrder(@Param("date") String date, @Param("product\_id") Long product\_id, @Param("user\_id") Long user\_id);  
  
 @Query(value = "Insert into DUNGTQ\_ORDER(order\_date, product\_id, user\_id)\n" +  
 "VALUES ( (SELECT CURRENT\_TIMESTAMP FROM DUAL),:product\_id,:user\_id)", nativeQuery = true)  
 Order insertOrderCurrentDate(@Param("product\_id") Long product\_id, @Param("user\_id") Long user\_id);  
  
  
 @Procedure  
 void updateOrder(Long id, Long product\_id, Long user\_id);  
}

**UserRepository**

@Repository  
public interface UserRepository extends JpaRepository<DAOUser, Integer> {  
 DAOUser findByUsername(String username);  
}

**CompanyRepository**

@Repository  
public interface CompanyRepository extends JpaRepository<Company, Long> {  
}

#### 2.2.6 Tạo các Service

UserService

@Service  
public class UserService {  
 @Autowired  
 private UserRepository userRepository;  
 @Autowired  
 private MailService mailService;  
  
 //get all Users  
 public List<DAOUser> getAllUsers() {  
 return userRepository.findAll();  
 }  
  
 //add new User  
 public DAOUser addNewUser(DAOUser user) {  
 if ("".equals(user.getUsername().trim()) || user.getPassword().trim().length() == 0 ||  
 user.getUsername().trim().length() == 0 || "".equals(user.getPassword().trim())) {  
 return null;  
 } else {  
 DAOUser user1 = userRepository.save(user);  
 mailService.sendSimpleEmail("Inserted new User: ", user1.getUsername() + "-" + user1.getPassword()  
 );  
 return user1;  
 }  
 }  
  
 //deleteUser  
 public void deleteUser(Long id) {  
 userRepository.deleteById(Integer.*parseInt*(id.toString()));  
 }  
  
 //update user  
 public void updateUser(DAOUser user) {  
 DAOUser user1 = userRepository.findById(user.getId().intValue()).get();  
 user1.setId(user.getId());  
 user1.setUsername(user.getUsername());  
 user1.setPassword(user.getPassword());  
 user1.setFirstName(user.getFirstName());  
 user1.setLastName(user.getLastName());  
 user1.setAge(user.getAge());  
 user1.setAddress(user.getAddress());  
 userRepository.save(user);  
 }  
  
 //get user by id  
 public DAOUser getUserById(Long id) {  
 return userRepository.findById(id.intValue()).get();  
 }  
  
 //get user by username  
 public DAOUser loadUserByUsername(String username) {  
 return userRepository.findByUsername(username);  
 }  
  
}

**OrderService**

@Service  
public class OrderService {  
 @Autowired  
 OrderRepository orderRepository;  
  
 public List<Order> getAllOrder() {  
 return orderRepository.getAll\_Order();  
 }  
  
 public Order insertOrderCurrentDate(Order order) {  
 return orderRepository.insertOrderCurrentDate(order.getProduct\_id(), order.getUser\_id());  
 }  
  
 public Order insertOrderDate(Order order) {  
 return orderRepository.insertOrder(order.getOrderDate().toString(), order.getProduct\_id(), order.getUser\_id());  
 }  
  
 public List<Order> getAllDetailOrder() {  
 return orderRepository.getAllDetailOrder();  
 }  
  
 public List<Order> getAllOrders() {  
 return orderRepository.findAll();  
 }  
  
 public void deleteOrder(Long id) {  
 orderRepository.deleteById(id);  
 }  
  
 public Order getOrderById(Long id) {  
 return orderRepository.findById(id).get();  
 }  
  
// public Order updateOrderUsingProcedure(Long id, Long product\_Id, Long user\_id) {  
// return orderRepository.updateOrderUsingProcedure(id, product\_Id, user\_id);  
// }  
  
 public void updateOrder(Long id, Long product\_id, Long user\_id) {  
 orderRepository.updateOrder(id, product\_id, user\_id);  
 }  
}

//deleteCategory  
 public void deleteCategory(Long id) {  
 categoryRepository.deleteById(id);  
 }  
  
 //update Category  
 public void updateCategory(Category category) {  
 Category category1 = categoryRepository.getById(category.getId());  
 category1.setCategoryName(category.getCategoryName());  
 category1.setCategoryOrigin(category.getCategoryOrigin());  
 category1.setCompany(category.getCompany());  
 categoryRepository.save(category1);  
 }  
  
 //get Category by id  
 public Category getCategoryById(Long id) {  
 return categoryRepository.findById(id).get();  
 }  
}

#### 2.2.7 Tạo ra các RestController

**OrderController**

@RestController  
@RequestMapping("/api/v1")  
public class OrderController {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(CompanyController.class);  
  
 @Value("${error.messages")  
 private String error\_message;  
  
 @Autowired  
 private OrderService orderService;  
  
  
 //get list orders with detail information  
 @GetMapping("/listOrderss")  
 @Operation(summary = "Get list orders with detail information")  
 public List<Order> getAllDetailOrders() {  
 *LOGGER*.info("List orders with detail information:");  
 return orderService.getAllDetailOrder();  
 }  
  
 @Operation(summary = "Get list orders ")  
 @ApiResponses(value = {  
 @ApiResponse(code = 200, message = "OK"),  
 @ApiResponse(code = 401, message = "Unauthorized"),  
 })  
  
 //get list order  
 @GetMapping("/listOrders")  
 public List<Order> getAllOrders() {  
 *LOGGER*.info("List orders: " + orderService.getAllOrder());  
 return orderService.getAllOrder();  
 }  
  
 //get order by id  
 @Operation(summary = "get order by id")  
 @GetMapping("/listOrders/{id}")  
 @Parameter(name = "id", required = true, description = "id of order")  
 public ResponseEntity<Order> getOrderByid(@PathVariable(value = "id") Long id) {  
 Order order = orderService.getOrderById(id);  
 *LOGGER*.info("Order with id: " + id + " is: " + order);  
 return ResponseEntity.*ok*(order);  
 }  
  
 //create order with date  
 @Operation(summary = "Create new order")  
 @PostMapping("/createOrders")  
 public Order createOrder(@Valid @RequestBody Order order) {  
 Order order1 = orderService.insertOrderDate(order);  
 *LOGGER*.info("Added order: " + order1.toString());  
 return order1;  
 }  
  
 //create order with current date  
 @Operation(summary = "Create new order with current date")  
 @PostMapping("/createOrder")  
 public Order createOrderCurrentDate(@Valid @RequestBody Order order) {  
 Order order1 = orderService.insertOrderCurrentDate(order);  
 *LOGGER*.info("Added order: " + order1.toString());  
 return order1;  
 }  
  
 //delete order  
 @Operation(summary = "Delete Order by id")  
 @Parameter(name = "id", required = true, description = "id of order")  
 @DeleteMapping("/listOrders/{id}")  
 public ResponseEntity<?> deleteOrder(@PathVariable(value = "id") Long id) {  
 Order order = orderService.getOrderById(id); //orElseThrow  
 orderService.deleteOrder(id);  
 *LOGGER*.info("Deleted order with id: " + id);  
 return ResponseEntity.*ok*().build();  
 }  
  
 //update order  
 @Operation(summary = "Update Order ")  
 @Parameter(name = "id", required = true, description = "id of order")  
 @PutMapping("/listOrders/{id}")  
 public ResponseEntity<Order> updateOrder(@PathVariable(value = "id") Long orderId, @Valid @RequestBody Order order) {  
 Order order1 = orderService.getOrderById(orderId); //orElseThrow  
 order1.setOrderDate(order.getOrderDate());  
 order1.setProduct\_id(order.getProduct\_id());  
 order1.setUser\_id(order.getUser\_id());  
 *LOGGER*.info("Updated Order: " + order1.toString());  
 return ResponseEntity.*ok*(order1  
 );  
 }  
}

### 2.3 Logging

#### 2.3.1 Logging là gì ?

* Logging là một phần quan trọng của tất cả các ứng dụng và mang lại lợi ích không chỉ cho các nhà phát triển chúng tôi mà còn cho các nhân viên và những người kinh doanh. Các ứng dụng Spring Boot cần nắm bắt dữ liệu nhật ký liên quan để giúp chúng tôi chẩn đoán và khắc phục sự cố cũng như đo lường các chỉ số kinh doanh.
* Các quyết định về việc log và ở đâu thường mang tính chiến lược và được đưa ra sau khi xem xét rằng ứng dụng sẽ hoạt động sai trong môi trường trực tiếp. Logging đóng một vai trò quan trọng trong việc giúp ứng dụng khôi phục nhanh chóng khỏi bất kỳ lỗi nào như vậy và tiếp tục hoạt động bình thường.

#### 2.3.2 Các thông tin trong log

* %d – In ra thời gian xảy ra log message theo SimpleDateFormat cho phép.
* %thread – In ra tên luồng mà log message xảy ra.
* $-5level – In ra mức độ của log.
* %logger{36} – In ra tên package + tên class xảy ra log. 36 là giới hạn độ dài tên package + tên class.
* %M – In ra tên phương thức thực hiện log
* %msg – In ra log message.
* %n – cách dòng.
* %magenta() - đặt màu của đầu ra chứa trong dấu ngoặc vuông thành màu magenta.
* highlight() - đặt màu của đầu ra chứa trong dấu ngoặc vuông dựa trên log level.

#### 2.3.3 Các mức độ của log (Logging Level) theo thứ tự tăng dần

* OFF (Không in ra log)
* ERROR
* WARN
* INFO
* DEBUG
* TRACE

#### 2.3.4 Cấu hình logback

Thêm dependencies

//logging  
<dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>jul-to-slf4j</artifactId>  
</dependency>

Cấu hình file trong application.properties

logging.config=classpath:logback-spring.xml

Tạo và cấu hình file logback-spring.xml

<?xml version="1.0" encoding="UTF-8"?>  
<configuration>  
  
 <!-- Đặt thuộc tính LOGS có giá trị là "/.logs" (folder chứa file logs)-->  
 <property name="LOGS" value="./logs"/>  
 <!-- Cấu hình log ra console-->  
 <appender name="Console"  
 class="ch.qos.logback.core.ConsoleAppender">  
 <layout class="ch.qos.logback.classic.PatternLayout">  
<!-- Dinh dang cua logging-->  
 <Pattern>  
 %white(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}): %msg%n%throwable  
 </Pattern>  
 </layout>  
 </appender>  
 <!-- Cấu hình log ngày hiện tại ra file spring-boot-logger.log-->  
 <appender name="RollingFile"  
 class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS}/spring-boot-logger.log</file>  
 <encoder  
 class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <Pattern>%d %p %C{1.} [%t] %m%n</Pattern>  
 </encoder>  
 <!-- Cấu hình lưu ra file theo mỗi ngày-->  
 <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">  
 <fileNamePattern>  
 ${LOGS}/archived/log\_%d{dd-MM-yyyy}\_%i.log  
 </fileNamePattern>  
 <!--Khi file quá 10MB sẽ tạo thêm 1 file mới với vị trí được đính kèm -->  
 <!--VD: /archived/log\_22-11-2021\_0 quá 10 MB -> tạo thêm /archived/log\_22-12-2021\_1 ...-->  
 <maxFileSize>10MB</maxFileSize>  
 <!-- Sau 30 ngày các file log này sẽ bị xóa-->  
 <maxHistory>30</maxHistory>  
 <!-- Khi tổng file đạt 100MB thì file log cũ nhất sẽ bị xóa-->  
 <totalSizeCap>100MB</totalSizeCap>  
 </rollingPolicy>  
 </appender>  
 <!-- Mức độ in log gốc (log từ mức INFO mới được in ra)-->  
 <root level="info">  
 <appender-ref ref="RollingFile"/>  
 <appender-ref ref="Console"/>  
 </root>  
 <!-- Mức độ in log cho một package nào đó-->  
 <logger name="com/example/ss\_jwt\_mysql\_helloworld/controller" level="debug" additivity="false">  
 <appender-ref ref="RollingFile"/>  
 <appender-ref ref="Console"/>  
 </logger>  
</configuration>

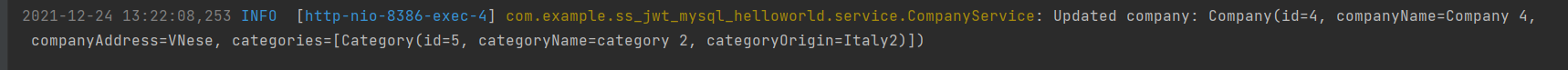
Sử dụng logging (Sử dụng log trong CompanyService)

Tạo một biến log

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(CompanyService.class);

In ra log ở mức INFO với nội dung “TEST API Get all products” mỗi khi gọi đến hàm getAllProducts của class ProductResource

public void updateCompany(Company company) {  
 Company company1 = companyRepository.getById(company.getId());  
 company1.setId(company.getId());  
 company1.setCompanyName(company.getCompanyName());  
 company1.setCompanyAddress(company.getCompanyAddress());  
 companyRepository.save(company1);  
 *LOGGER*.info("Updated company: " + company1);  
}

Kết quả

### 2.4 Swagger

#### 2.4.1 Swagger là gì ?

* Swagger đơn giản hóa việc phát triển API cho người dùng, nhóm và doanh nghiệp với bộ công cụ chuyên nghiệp và mã nguồn mở Swagger. Tìm hiểu cách Swagger có thể giúp bạn thiết kế và lập tài liệu các API của mình trên quy mô lớn.

#### 2.4.2 Cấu hình Swagger

Thêm các dependencies

<dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-ui</artifactId>  
 <version>1.2.32</version>  
</dependency>  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>3.0.0</version>  
</dependency>  
  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 <version>2.9.2</version>  
</dependency>

Thêm các chú thích cho từng phần trong Swagger như API hay Schema

* Với class DAOUser

@Data  
@Entity  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "DungTQ\_User")  
public class DAOUser implements Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Schema(title = "id of user", example = "1")  
 private Long id;  
  
 @Column(name = "username", nullable = false, unique = true)  
 @Schema(title = "username of user", example = "username")  
 private String username;  
  
 @Column(name = "password", nullable = false)  
 @Schema(title = "password of user", example = "password")  
 private String password;  
  
 @Column(name = "firstName")  
 @Schema(title = "firstName of user", example = "John")  
 private String firstName;  
  
 @Column(name = "lastName")  
 @Schema(title = "lastName of user", example = "Dung")  
 private String lastName;  
  
 @Column(name = "age", nullable = false)  
 @Schema(title = "age of user", example = "1")  
 private int age;  
  
 @Column(name = "address")  
 @Schema(title = "address of user", example = "Ha Noi")  
 private String address;

}

* Class Company

@Data  
@Entity  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Schema(description = "Company's information")  
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})  
@Table(name = "DungTQ\_Company")  
public class Company implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Schema(title = "ID of company", example = "1")  
 private Long id;  
  
 @Column(name = "companyName")  
 @Schema(title = "Name of company", example = "company1")  
 private String companyName;  
  
 @Column(name = "companyAddress")  
 @Schema(title = "Address of company", example = "Viet Nam")  
 private String companyAddress;  
  
 public Company(Long id, String companyName, String companyAddress) {  
 this.id = id;  
 this.companyName = companyName;  
 this.companyAddress = companyAddress;  
 }  
  
 public Company(Long id) {  
 this.id = id;  
 }  
 @JsonIgnore  
 @OneToMany(mappedBy = "company", cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*)  
// @EqualsAndHashCode.Exclude  
// @ToString.Exclude  
 private Collection<Category> categories;  
}

* Chú thích cho API

@Operation(summary = "Get all companies")  
@ApiResponses(value = {  
 @ApiResponse(code = 200, message = "OK"),  
 @ApiResponse(code = 401, message = "Unauthorized"),  
})  
@GetMapping("/listCompanies")  
public List<Company> getAllCompanies() {  
 *LOGGER*.info("List companies");  
 return companyService.getAllCompany();  
}

Test thử Swagger (Demo)

### 2.5 Email

#### 2.5.1 Thêm các dependencies

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-mail</artifactId>  
</dependency>

#### 2.5.2 Cấu hình thông tin trong qua class MyConstants

public class MyConstants {  
 // Replace with your email here:  
 public static final String *MY\_EMAIL* = "dominononono@gmail.com";  
  
 // Replace password!!  
 public static final String *MY\_PASSWORD* = "bababbaab";  
  
 // And receiver!  
 public static final String *FRIEND\_EMAIL* = "dungtucococq@aladintech.co";  
}

#### 2.5.3 Tạo email service

@Service  
public class MailService {  
  
 @Autowired  
 public JavaMailSender emailSender;  
  
 // @RequestMapping("/sendSimpleEmail")  
 public String sendSimpleEmail(String subject, String text) {  
  
 //create a simple mailmessage  
 SimpleMailMessage message = new SimpleMailMessage();  
  
 message.setTo(MyConstants.*FRIEND\_EMAIL*);  
 message.setSubject(subject);  
 message.setText(text);  
  
 this.emailSender.send(message);  
 return "email sent";  
 }  
  
 public String sendAttachmenEmail(String subject, String text, String path1, String path2) throws MessagingException {  
 //create mimeMessage  
 MimeMessage message = emailSender.createMimeMessage();  
  
 MimeMessageHelper helper = new MimeMessageHelper(message, true);  
 helper.setFrom(MyConstants.*MY\_EMAIL*);  
 helper.setTo(MyConstants.*FRIEND\_EMAIL*);  
 helper.setSubject(subject);  
 helper.setText(text);  
  
// String path1 = "C:\\Users\\quang\\Desktop\\file1.txt";  
// String path2 = "C:\\Users\\quang\\Desktop\\file2.txt";  
  
 //Attachment 1  
 FileSystemResource file1 = new FileSystemResource(new File(path1));  
 helper.addAttachment("file1", file1);  
  
 //Attachment 2  
 FileSystemResource file2 = new FileSystemResource(new File(path2));  
 helper.addAttachment("file2", file2);  
  
 emailSender.send(message);  
 return "Email sent";  
 }  
}

#### 2.5.4 Tạo email controller

Phần gửi email với nhiều file đính kèm sẽ test ở phần Schedule

@RestController  
@RequestMapping("/api/v1")  
public class MailSenderController {  
  
 @Autowired  
 private MailService mailService;  
  
 @PostMapping("/sendAttachEmail")  
 public String sendAttachEmail(@RequestParam(name = "subject", required = true) String subject,  
 @RequestParam(name = "text", required = true) String text,  
 @RequestParam(name = "path1") String path1,  
 @RequestParam(name = "path2") String path2) throws MessagingException {  
 mailService.sendAttachmenEmail(subject, text, path1, path2);  
 return "Email sent";  
 }  
}

}  
}

## Tài liệu tham khảo

AladinTrainingProgrammer.docx – Tài liệu của công ty Aladin Technology.

<https://www.javainuse.com/webseries/spring-security-jwt/chap5>

<https://loda.me/courses/spring-boot>

https://viblo.asia/p/scheduling-task-trong-spring-boot-yMnKMy2QK7P