**1. Giải thích về Entity**

**Khái niệm Entity**

* **Entity** trong Spring Boot là một lớp Java đại diện cho một bảng trong cơ sở dữ liệu.
* Mỗi thuộc tính của Entity tương ứng với một cột trong bảng đó.
* Entity chứa các annotation để thực hiện ánh xạ giữa đối tượng Java và cơ sở dữ liệu.

**Annotations thường dùng trong Entity**

* **@Entity**: Đánh dấu một lớp Java là một thực thể trong JPA.
* **@Table**: Chỉ định tên bảng trong cơ sở dữ liệu mà Entity ánh xạ tới. Ví dụ: @Table(name = "users").
* **@Id**: Đánh dấu thuộc tính là khóa chính của Entity.
* **@GeneratedValue**: Chỉ định chiến lược tạo giá trị tự động cho khóa chính. Các tùy chọn bao gồm:
  + GenerationType.IDENTITY: Tự động tăng giá trị khóa chính.
  + GenerationType.SEQUENCE: Sử dụng sequence để tạo giá trị khóa chính.
  + GenerationType.AUTO: Để JPA tự động chọn chiến lược phù hợp.
* **@Column**: Chỉ định thuộc tính ánh xạ tới cột nào trong bảng, có thể chỉ định thêm các thuộc tính như nullable, unique, length.
* **@ManyToOne**, **@OneToMany**, **@ManyToMany**, **@OneToOne**: Định nghĩa các quan hệ giữa các thực thể với nhau.

**Ví dụ về Entity User**

**Ví dụ về class BaseEntity**

* // Đánh dấu đây là class cha cho tất cả các entity kế thừa  
  @MappedSuperclass  
  @Data // tự tạo getter, setter  
  public class BaseEntity {  
    
   @Id // xác định đây là khóa chính  
   @GeneratedValue(strategy = GenerationType.*IDENTITY*) // config khóa chính tự động tăng  
   public Long id;  
    
   @Column(name = "created\_date") // chỉ định thuộc tính này của  
   // java sẽ mapping với cột created\_date trong table ứng với entity  
   private LocalDateTime createdDate;

**Ví dụ về class UserEntity**

// Đánh dấu class này là một entity ứng với một table trong database  
@Entity  
@Table(name = "users") // chỉ định UserEntity ứng với table users trong db  
@Getter  
@Setter  
public class UserEntity extends BaseEntity{  
  
 */\*\*  
 Khai báo các thuộc tính ứng với các cột trong bảng  
 users  
 \*/* private String username;

* **Lombok Annotations**:
  + **@Data**: Tự động tạo getter, setter, toString(), equals() và hashCode() cho các thuộc tính.
  + **@NoArgsConstructor**: Tạo constructor không tham số.
  + **@AllArgsConstructor**: Tạo constructor có tham số cho tất cả các thuộc tính.

**2. Giải thích về Repository**

**Khái niệm Repository**

* **Repository** là một lớp hoặc interface trong Spring Data JPA giúp cho ứng dụng Java có thể kết nối và thực hiện các công việc đọc(truy vấn), ghi(insert), cập nhật(update)... vào cơ sở dữ liệu thông qua các hàm đã được xây dựng sẵn bởi Repository và Entity
* Repository sử dụng các phương thức của JPA để truy vấn và thao tác dữ liệu mà không cần viết nhiều mã SQL phức tạp.
* Có thể tự định nghĩa ra các hàm riêng việt để xử lý với các câu sql đặc biệt

**Cấu trúc của JpaRepository<T, ID>**

* **JpaRepository<T, ID>**: Là một interface cung cấp các phương thức cơ bản để thao tác với dữ liệu trong bảng.
  + **T**: Loại của Entity mà Repository sẽ làm việc.
  + **ID**: Loại của khóa chính của Entity đó.

**Các phương thức thường dùng trong Repository**

* **CRUD Methods**:
  + **save(S entity)**: Lưu hoặc cập nhật một thực thể vào cơ sở dữ liệu.
  + **findById(ID id)**: Tìm một thực thể dựa trên khóa chính.
  + **findAll()**: Lấy tất cả các thực thể từ cơ sở dữ liệu.
  + **deleteById(ID id)**: Xóa một thực thể dựa trên khóa chính.
  + **delete(T entity)**: Xóa một thực thể cụ thể.
  + **count()**: Đếm tổng số thực thể trong bảng.
* **Paging and Sorting**:
  + **findAll(Sort sort)**: Trả về tất cả các thực thể theo sắp xếp.
  + **findAll(Pageable pageable)**: Trả về các thực thể theo phân trang và sắp xếp.
* **Query Methods**:
  + **findByUsername(String username)**: Tìm một thực thể dựa trên tên đăng nhập (ví dụ tùy chỉnh).

**Ví dụ về Repository: UserRepository**

import com.t3h.insuranceclaim.entity.UserEntity;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UserRepository extends JpaRepository<UserEntity, Integer> {  
  
 UserEntity findByUsername(String username);  
}

**Ví dụ về Repository: ClaimRepository và hàm tự định nghĩa trong ClaimRepository**

import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
import org.thymeleaf.util.Validate;  
@Repository  
public interface ClaimRepository extends JpaRepository<ClaimEntity, Long> {  
  
 @Query("SELECT c FROM ClaimEntity c " +  
 "WHERE (:#{#request.claimCode} = '' OR LOWER(c.code) LIKE LOWER(CONCAT('%', :#{#request.claimCode}, '%'))) " +  
 "AND (:#{#request.statusCode} = '' OR c.claimStatusEntity.code = :#{#request.statusCode}) " +  
 "AND (:#{#request.fromDateSearch} IS NULL OR c.claimDate >= :#{#request.fromDateSearch}) " +  
 "AND (:#{#request.toDateSearch} IS NULL OR c.claimDate <= :#{#request.toDateSearch}) AND (c.deleted=false OR c.deleted is NULL) ")  
 Page<ClaimEntity> findByCondition(@Param("request") ClaimRequest request, Pageable pageable);

Đây là câu query sử dụng JPQL trong spring data JPA

**3. Giải thích về DTO**

**Khái niệm DTO**

* **DTO (Data Transfer Object)** là một lớp dùng để truyền dữ liệu giữa các tầng trong ứng dụng hoặc giữa các hệ thống mà không gắn với cấu trúc cơ sở dữ liệu cụ thể.
* DTO giúp tối ưu hóa việc truyền tải dữ liệu và bảo mật thông tin nhạy cảm.

**Đặc điểm của DTO**

* **Plain Object**: Chỉ chứa các thuộc tính dữ liệu và không có logic nghiệp vụ.
* **Không chứa annotation JPA**: DTO không liên quan trực tiếp đến cơ sở dữ liệu.
* **Sử dụng thư viện để chuyển đổi**: Có thể sử dụng các thư viện như MapStruct hoặc ModelMapper để chuyển đổi giữa Entity và DTO.

**Ví dụ về DTO: UserDTO**

* @Getter  
  @Setter  
  public class UserDTO {  
    
   private Long id;  
   private String username;  
   private String password;  
   private String code;  
   private String email;  
   private String phone;  
   private String address;  
   private String firstName;  
   private String lastName;  
    
   private LocalDate createdDate;  
   private String createdBy;  
   private LocalDate lastModifiedDate;  
   private String lastModifiedBy;  
   private Boolean deleted;  
    
  }
* **Lombok Annotations**:
  + **@Data**: Tự động tạo getter, setter, toString(), equals() và hashCode() cho các thuộc tính.
  + **@NoArgsConstructor**: Tạo constructor không tham số.
  + **@AllArgsConstructor**: Tạo constructor có tham số cho tất cả các thuộc tính.

**4. Giải thích về Service và cách sử dụng DTO để xử lý nghiệp vụ**

**Khái niệm Service**

* **Service** là một lớp chứa logic nghiệp vụ của ứng dụng. Nó tương tác với Repository để thực hiện các thao tác cơ sở dữ liệu và nhận được dữ liệu từ database dưới dạng các **class ENTITY** và xử lý logic, **convert dữ liệu từ dạng ENTITY sang dạng DTO** trước khi trả về dữ liệu cho Controller.

**Cách sử dụng DTO trong Service**

* **Chuyển đổi Entity thành DTO**: Khi lấy dữ liệu từ Repository, Service phải chuyển đổi dữ liệu từ Entity sang DTO để trả về cho Controller.
* **Chuyển đổi DTO thành Entity**: Khi nhận dữ liệu từ Controller, Service chuyển đổi dữ liệu từ DTO thành Entity trước khi lưu vào cơ sở dữ liệu.

**Ví dụ về Service: UserService và UserServiceImpl**

import java.util.List;  
@Service// đánh dấu đây là tầng service  
public class UserServiceImpl implements IUserService {  
 @Autowired  
 private UserRepository userRepository;  
 @Override  
 public List<UserDTO> getAllUser() {  
 // query lấy danh sách user từ database sử dụng repository   
 List<UserEntity> userEntities = userRepository.findAll();  
 // chuyển dữ liệu từ dang entity sang dạng DTO để trả về controller   
 List<UserDTO> userDTOs = new ArrayList<UserDTO>();  
 for (UserEntity userEntity : userEntities) {  
 UserDTO userDTO = new UserDTO();  
 // Set các giá trị từ UserEntity sang UserDTO  
 userDTO.setId(userEntity.getId());  
 userDTO.setUsername(userEntity.getUsername());  
 userDTO.setCode(userEntity.getCode());  
 userDTO.setEmail(userEntity.getEmail());  
 userDTO.setFirstName(userEntity.getFirstName());  
 userDTO.setLastName(userEntity.getLastName());  
 userDTO.setPhone(userEntity.getPhone());  
 userDTO.setAddress(userEntity.getAddress());  
 // Các trường từ BaseEntity sang UserDTO  
 if (userEntity.getCreatedDate() != null){  
 userDTO.setCreatedDate(userEntity.getCreatedDate().toLocalDate());  
 }  
 userDTO.setCreatedBy(userEntity.getCreatedBy());  
 if (userEntity.getLastModifiedDate() != null){  
 userDTO.setLastModifiedDate(userEntity.getLastModifiedDate().toLocalDate());  
 }  
 userDTO.setLastModifiedBy(userEntity.getLastModifiedBy());  
 userDTO.setDeleted(userEntity.getDeleted());  
 userDTOs.add(userDTO);  
 }  
 // trả về danh sách user cho controller  
 return userDTOs;  
 }  
}

**Giải thích:**

* **Trong hàm này của service lấy ra danh sách user từ database sử dụng repositoy. Nhận được kiểu dữ liệu entity và chuyển kiểu dữ liệu dạng entity sang DTO**

**5. Giải thích về Controller**

**Khái niệm Controller**

* **Controller** là một lớp chịu trách nhiệm nhận các yêu cầu HTTP từ client và trả về phản hồi tương ứng. Nó sử dụng các phương thức từ Service để thực hiện logic nghiệp vụ.

**Đặc điểm của Controller**

* **Annotations**:
  + **@RestController**: Đánh dấu lớp là một RESTful Controller, kết hợp giữa @Controller và @ResponseBody.
  + **@RequestMapping**: Định tuyến các yêu cầu HTTP đến các phương thức xử lý.
  + **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping**: Định nghĩa các phương thức xử lý cho các yêu cầu HTTP tương ứng.
  + **@PathVariable**: Lấy dữ liệu từ URI.
  + **@RequestBody**: Ánh xạ nội dung của request body vào một đối tượng Java.

**Ví dụ về Controller: UserController**

import org.springframework.web.bind.annotation.RestController;  
import vn.com.t3h.dto.UserDTO;  
import vn.com.t3h.service.IUserService;  
  
import java.util.List;  
  
@RestController// khai báo đây là một restcontroller chuyên nhận request và trả về response  
@RequestMapping("api/user/") // chỉ định url cho api bắt đầu bằng http://localhost:8080/api/user/  
public class UserResourceController {  
   
 @Autowired // Tiêm bean từ ngoài vào sử dụng design DI  
 private IUserService iUserService; // khai báo tầng service   
  
 @GetMapping("all-user")// định nghĩa url sẽ được mapping. ứng với url http://localhost:8080/api/user/all-user sẽ được hàm getAllUser tiếp nhận và xử lý  
 public ResponseEntity<List<UserDTO>> getAllUser() {  
 // gọi tới tầng service để nhận về dữ liệu DTO   
 List<UserDTO> userDTOList = iUserService.getAllUser();  
 // trả dữ liệu từ service về response để chuyển sang dữ liệu dưới dạng json   
 return ResponseEntity.*ok*(userDTOList);  
 }  
}

**6. Giải thích luồng về Entity, Repository, DTO, Service, Controller**

**Luồng hoạt động từ Controller đến Entity qua DTO**

1. **Controller**:
   * Nhận yêu cầu từ client.
   * Ánh xạ dữ liệu từ request body vào DTO bằng @RequestBody.
   * Gọi phương thức từ tầng Service và truyền dữ liệu DTO vào.
2. **Service**:
   * Nhận DTO từ Controller.
   * Chuyển đổi DTO thành Entity để tương tác với Repository.
   * Gọi phương thức từ Repository để thực hiện các thao tác CRUD.
   * Sau khi nhận được dữ liệu từ Repository, chuyển đổi Entity thành DTO.
   * Trả về DTO cho Controller.
3. **Repository**:
   * Nhận Entity từ Service.
   * Sử dụng các phương thức của JPA để thực hiện các thao tác trên cơ sở dữ liệu.
   * Trả về Entity cho Service.
4. **Entity**:
   * Đại diện cho các bảng trong cơ sở dữ liệu.
   * Được Repository sử dụng để thực hiện các thao tác CRUD.
5. **DTO**:
   * Được sử dụng để truyền dữ liệu giữa Controller và Service.
   * Chứa các thông tin cần thiết mà không gắn với cấu trúc cơ sở dữ liệu cụ thể.

**Lợi ích của việc sử dụng DTO**

* **Giảm thiểu dữ liệu truyền tải**: Chỉ truyền tải những thông tin cần thiết.
* **Bảo mật**: Ngăn ngừa việc truyền tải các dữ liệu nhạy cảm.
* **Tách biệt logic nghiệp vụ và dữ liệu**: DTO giúp tách biệt cấu trúc dữ liệu của API với cấu trúc dữ liệu của Entity.

**7. Giải thích về API**

**Khái niệm API**

* **API (Application Programming Interface)** là một tập hợp các quy tắc và giao thức cho phép các ứng dụng hoặc dịch vụ giao tiếp với nhau.
* API định nghĩa cách một ứng dụng có thể tương tác với một ứng dụng khác.

**Các phương thức HTTP trong API**

* **GET**: Lấy dữ liệu từ server.
* **POST**: Gửi dữ liệu đến server để tạo mới.
* **PUT**: Cập nhật dữ liệu hiện có trên server.
* **DELETE**: Xóa dữ liệu trên server.
* **PATCH**: Cập nhật một phần dữ liệu trên server.

**API trong Spring Boot**

* **Spring Boot** sử dụng các annotation để định nghĩa các API RESTful.
* **REST (Representational State Transfer)** là một kiến trúc phần mềm cho phép giao tiếp giữa client và server thông qua các phương thức HTTP.

**8. Ví dụ về cách làm API lấy ra danh thông tin chi tiết của UserEntity**

**Luồng từ Controller, Service, DTO, Repository, Entity**

* **Entity**: BaseEntity đóng vai trò supper entity cho tất cả các entity khác kế thừa

import jakarta.persistence.\*;  
import lombok.Data;  
  
import java.time.LocalDateTime;  
  
// Đánh dấu đây là class cha cho tất cả các entity kế thừa  
@MappedSuperclass  
@Data // tự tạo getter, setter  
public class BaseEntity {  
  
 @Id // xác định đây là khóa chính  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*) // config khóa chính tự động tăng  
 public Long id;  
  
 @Column(name = "created\_date") // chỉ định thuộc tính này của  
 // java sẽ mapping với cột created\_date trong table ứng với entity  
 private LocalDateTime createdDate;  
  
 // khi không đánh dấu @Column thì tự động JPA sẽ hiểu thuộc tính  
 // java này createdBy đang mapping với cột created\_by trong table  
 private String createdBy;  
 private LocalDateTime lastModifiedDate;  
 private String lastModifiedBy;  
 private Boolean deleted = false;  
}

* **Entity**: UserEntity đóng vai trò mapping dữ liệu từ bảng users và class UserEntity trong Java
* import jakarta.persistence.\*;  
  import lombok.Data;  
  import lombok.Getter;  
  import lombok.Setter;  
    
  import java.util.HashSet;  
  import java.util.Set;  
    
  // Đánh dấu class này là một entity ứng với một table trong database  
  @Entity  
  @Table(name = "users") // chỉ định UserEntity ứng với table users trong db  
  @Getter  
  @Setter  
  public class UserEntity extends BaseEntity{  
    
   */\*\*  
   Khai báo các thuộc tính ứng với các cột trong bảng  
   users  
   \*/* private String username;  
   private String password;  
   private String code;  
   private String email;  
   private String firstName;  
   private String lastName;  
   private String phone;  
   private String address;  
    
   */\*\*  
   config fetch:  
   FetchType.EAGER: khi query user trong database sẽ lấy ra hết  
   danh sách quyền thuộc về user đấy và đưa vào thuộc tính roles  
   FetchType.LAZY: khi query user trong database sẽ không lấy ra  
   ngay danh sách quyền và đưa vào roles, mà sẽ chỉ lấy ra và  
   đưa vào khi gọi tới hàm getRoles()  
   cascade = CascadeType.ALL: khi tạo ra hoặc xóa đi một user  
   -> sẽ có thể tạo ra hoặc xóa đi một role đính kèm  
   \*/* @ManyToMany(fetch = FetchType.*LAZY*,cascade = CascadeType.*ALL*)  
   @JoinTable( // tự động tạo ra bảng phụ user\_roles  
   name = "user\_roles",  
   joinColumns = @JoinColumn(name = "user\_id"), /\* Cấu hình để thuộc tính user\_id trong bảng phụ user\_roles  
   sẽ là khóa phụ tham chiếu tới cột id trong bảng users\*/  
   inverseJoinColumns = @JoinColumn(name = "role\_id")/\* Cấu hình để thuộc tính role\_id trong bảng phụ user\_roles  
   sẽ là khóa phụ tham chiếu tới cột id trong bảng roles\*/  
   )  
   // 1 user có nhiều quyền  
   private Set<RoleEntity> roleEntities = new HashSet<>();  
  }

1. **Repository**: UserRepository đóng vai trò kết nối giữa tầng application java spring boot và database. Thông qua UserEntity
   * Cung cấp sẵn các hàm thao tác CRUD( Create, Read, Update, Delete) trên bảng users.

import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import vn.com.t3h.entity.UserEntity;  
  
*/\*\*  
 là interface được kế thừa từ JpaRepository  
 - khi khai báo UserRepository và cho kế thừa từ JpaRepository<UserEntity, Long>  
 + thì tương đương interface này đóng vai trò là một interface cung cấp các  
 method xử lý và kết nối với database  
 + interface JpaRepository đã cung cấp sẵn tất cả các hàm cơ bản để làm việc với database  
 như  
 getOne : lấy ra data theo id  
 getById: lấy ra data trong databse theo id  
 findAll: lấy ra tất cả dữ liệu trong table  
 saveAll: lưu tất cả list vào dabaatase  
 save: lưu dữ liệu vào database  
 .... và rất nhiều hàm khác  
 mà UserRepository đang kế thừa từ JpaRepository => UserRepository sẽ tự động có tất cả các hàm truy vấn với database của  
 interface cha  
 JpaRepository<T, ID> :  
 T: Entity mapping giữa database và app Java  
 ID: kiểu dữ liệu của cột ID trong Entity của Java  
 vd: JpaRepository<UserEntity, Long>  
 UserEntity: là entity của Java  
 Long: Là kiểu dữ liệu của trường id trong entity UserEntity  
 - Quy trình để kết nối app Java với database sử dụng JDBC khi chưa có Spring data JPA  
 b1) tạo kết nối đến database  
 b2) tạo ra câu query để truy vấn dữ liệu vd: select \* from user\_roles;  
 b3) Sử dụng thư viện JDBC thư thi câu query select \* from user\_roles; vào trong database  
 và nhận được dữ liệu dưới dạng table trong database  
 b4) Chuyển dữ liệu dạng table trong database sang thành dạng class của java  
 mục tiêu cuối cùng nhận về dữ liệu database đã được chuyển thành các object java  
  
 \*/*@Repository // để spring boot hiểu đây là interface repository và tự động tạo và quản lý bean  
public interface UserRepository extends JpaRepository<UserEntity, Long> {  
}

1. **DTO**: UserDTO
   * Truyền dữ liệu giữa Controller và Service.

import lombok.Getter;  
import lombok.Setter;  
  
import java.time.LocalDate;  
  
*/\*\*  
 "id": 1,  
 "username": "admin",  
 "password": "$2a$10$f6p5TLa9sDZzuTzRgnQAFewEMqU843QaSQqjjy56Fmk.UjgNkqTs2",  
 "code": "A0001",  
 "email": "admin@gmail.com",  
 "firstName": "Nguyen",  
 "lastName": "Admin",  
 "phone": "0925334772",  
 "address": "HaNoi",  
 "roles": [  
 {  
 "code": "ADMIN",  
 "name": "Quản trị viên"  
 }  
 ],  
 "createdDate": null,  
 "createdBy": null,  
 "lastModifiedDate": null,  
 "lastModifiedBy": "2024-07-27 12:25:58.000000 +00:00",  
 "deleted": false  
 \*/*@Getter  
@Setter  
public class UserDTO {  
  
 private Long id;  
 private String username;  
 private String code;  
 private String email;  
 private String phone;  
 private String address;  
 private String firstName;  
 private String lastName;  
  
 private LocalDate createdDate;  
 private String createdBy;  
 private LocalDate lastModifiedDate;  
 private String lastModifiedBy;  
 private Boolean deleted;  
}

1. **Service**: UserService
   * Xử lý logic nghiệp vụ và chuyển đổi dữ liệu giữa Entity và DTO.

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import vn.com.t3h.dto.UserDTO;  
import vn.com.t3h.entity.UserEntity;  
import vn.com.t3h.repository.UserRepository;  
import vn.com.t3h.service.IUserService;  
import java.util.ArrayList;  
import java.util.List;  
@Service// đánh dấu đây là tầng service  
public class UserServiceImpl implements IUserService {  
 @Autowired  
 private UserRepository userRepository;  
 @Override  
 public List<UserDTO> getAllUser() {  
 // query lấy danh sách user từ database sử dụng repository  
 List<UserEntity> userEntities = userRepository.findAll();  
 // chuyển dữ liệu từ dang entity sang dạng DTO để trả về controller  
 List<UserDTO> userDTOs = new ArrayList<UserDTO>();  
 for (UserEntity userEntity : userEntities) {  
 UserDTO userDTO = new UserDTO();  
 // Set các giá trị từ UserEntity sang UserDTO  
 userDTO.setId(userEntity.getId());  
 userDTO.setUsername(userEntity.getUsername());  
 userDTO.setCode(userEntity.getCode());  
 userDTO.setEmail(userEntity.getEmail());  
 userDTO.setFirstName(userEntity.getFirstName());  
 userDTO.setLastName(userEntity.getLastName());  
 userDTO.setPhone(userEntity.getPhone());  
 userDTO.setAddress(userEntity.getAddress());  
 // Các trường từ BaseEntity sang UserDTO  
 if (userEntity.getCreatedDate() != null){  
 userDTO.setCreatedDate(userEntity.getCreatedDate().toLocalDate());  
 }  
 userDTO.setCreatedBy(userEntity.getCreatedBy());  
 if (userEntity.getLastModifiedDate() != null){  
 userDTO.setLastModifiedDate(userEntity.getLastModifiedDate().toLocalDate());  
 }  
 userDTO.setLastModifiedBy(userEntity.getLastModifiedBy());  
 userDTO.setDeleted(userEntity.getDeleted());  
 userDTOs.add(userDTO);  
 }  
 // trả về danh sách user cho controller  
 return userDTOs;  
 }  
}

1. **Controller**: UserController
   * Nhận yêu cầu từ client, gọi tới tầng service để xử lý, trả về dữ liệu cho client
2. import org.springframework.beans.factory.annotation.Autowired;  
   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 vn.com.t3h.dto.UserDTO;  
   import vn.com.t3h.service.IUserService;  
     
   import java.util.List;  
     
   @RestController// khai báo đây là một restcontroller chuyên nhận request và trả về response  
   @RequestMapping("api/user/") // chỉ định url cho api bắt đầu bằng http://localhost:8080/api/user/  
   public class UserResourceController {  
     
    @Autowired // Tiêm bean từ ngoài vào sử dụng design DI  
    private IUserService iUserService; // khai báo tầng service  
     
    @GetMapping("all-user")// định nghĩa url sẽ được mapping. ứng với url http://localhost:8080/api/user/all-user sẽ được hàm getAllUser tiếp nhận và xử lý  
    public ResponseEntity<List<UserDTO>> getAllUser() {  
    // gọi tới tầng service để nhận về dữ liệu DTO  
    List<UserDTO> userDTOList = iUserService.getAllUser();  
    // trả dữ liệu từ service về response để chuyển sang dữ liệu dưới dạng json  
    return ResponseEntity.*ok*(userDTOList);  
    }  
   }
3. **Giải thích từng bước**
4. **Nhận yêu cầu từ client**:
   * Ví dụ, khi client gửi yêu cầu GET đến /api/users/{id}, phương thức getUserById trong UserController sẽ được gọi.
5. **Tương tác với Service**:
   * UserController gọi phương thức getUserById từ UserService và truyền vào id từ @PathVariable.
6. **Chuyển đổi dữ liệu**:
   * UserService sử dụng UserRepository để tìm kiếm Entity User theo id.
   * Chuyển đổi Entity User thành DTO UserDTO bằng cách gọi mapToDTO.
7. **Trả về dữ liệu**:
   * UserService trả về đối tượng UserDTO cho UserController.
   * UserController trả về dữ liệu UserDTO cho client thông qua HTTP response.

**Tổng kết**

* **Entity**: Ánh xạ trực tiếp với cơ sở dữ liệu.
* **Repository**: Quản lý truy cập dữ liệu và thao tác CRUD.
* **DTO**: Truyền dữ liệu giữa các tầng và giảm thiểu dữ liệu không cần thiết.
* **Service**: Xử lý logic nghiệp vụ và chuyển đổi dữ liệu giữa Entity và DTO.
* **Controller**: Nhận yêu cầu HTTP và trả về phản hồi cho client.
* **API**: Giao diện giao tiếp giữa các ứng dụng, sử dụng các phương thức HTTP để truyền tải dữ liệu.

## 9. Cách call api sử dụng postmain

1) Sau khi viết api sẽ có đường dẫn

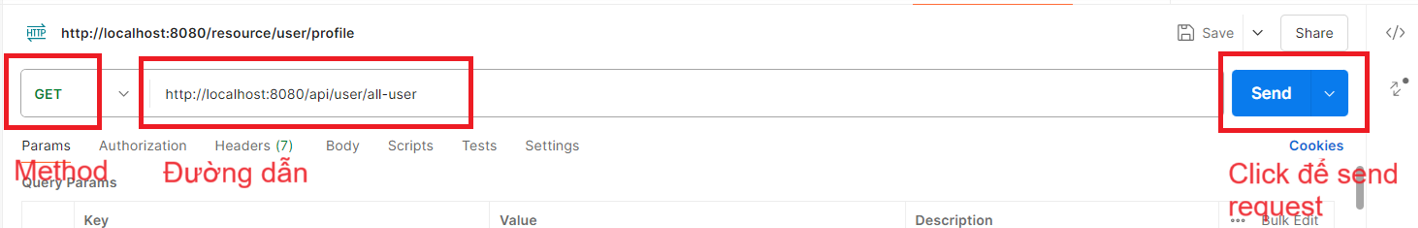
http://localhost:8080/api/user/all-user

2) Mở postman và điền các thông tin sau:

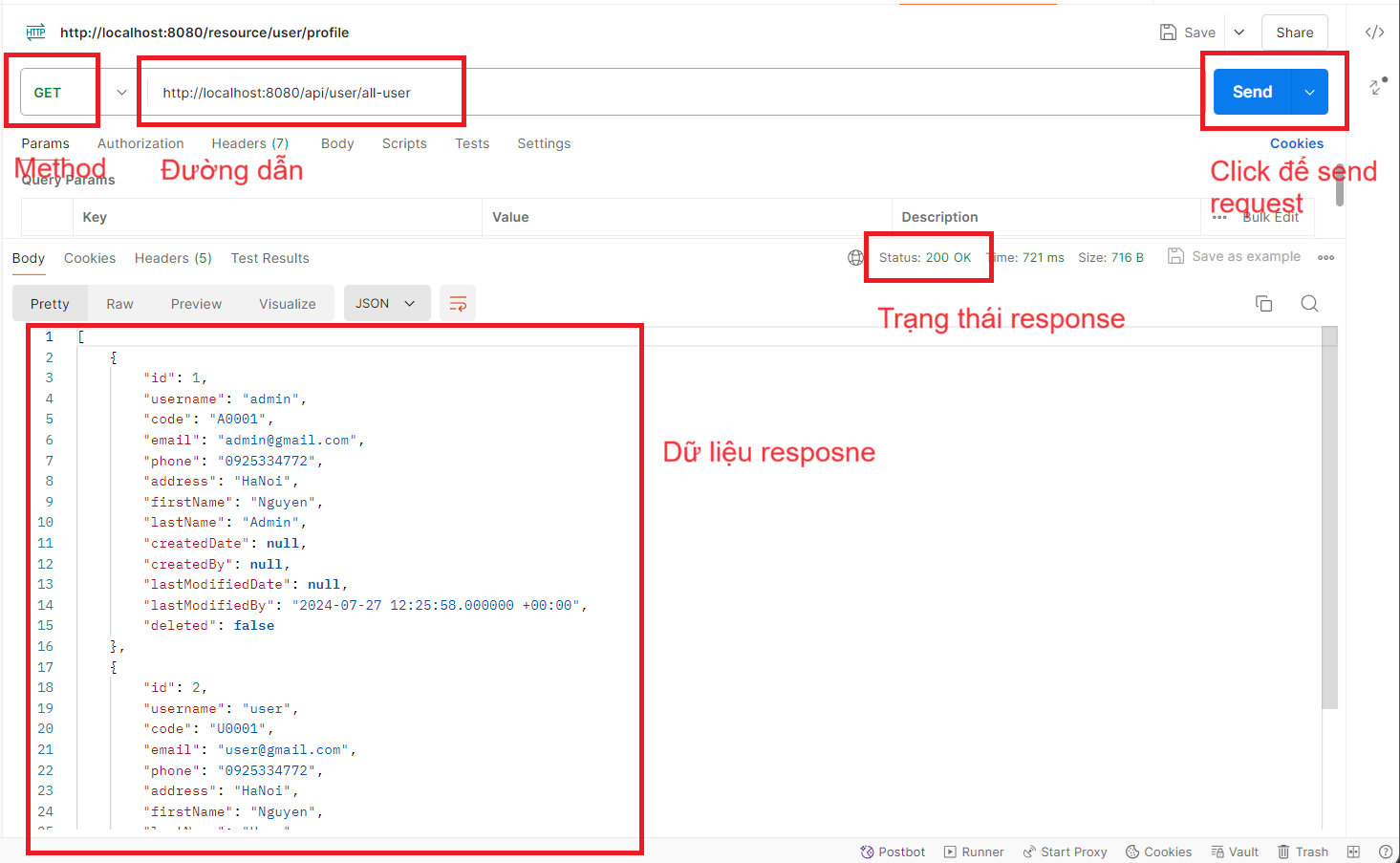
Đường dẫn: http://localhost:8080/api/user/all-user

Method: GET

3) Rồi click button Send để gửi request như ảnh



4) Sau khi click send sẽ nhận được response dưới dạng json như hình. Vậy là đã tạo thành công api



Code mẫu: nằm trong folder LJAVA2402\Buoi19\_Spring\claim-manager

Bài tập: Viết api lấy thông tin user thông qua username, username là param được gửi từ client vào

Gợi ý:

1) Viết hàm lấy thông tin userentity sử dụng user trong UserRepository

@Repository // để spring boot hiểu đây là interface repository và tự động tạo và quản lý bean  
public interface UserRepository extends JpaRepository<UserEntity, Long> {  
  
 UserEntity findByUsername(String username);  
}

2) Sử dụng annotation @Requestparam tại controller để nhận được username từ client

3) thông tin user được trả về dưới dạng json như sau :  
 {

        "id": 1,

        "username": "admin",

        "code": "A0001",

        "email": "admin@gmail.com",

        "phone": "0925334772",

        "address": "HaNoi",

        "firstName": "Nguyen",

        "lastName": "Admin",

        "createdDate": **null**,

        "createdBy": **null**,

        "lastModifiedDate": **null**,

        "lastModifiedBy": "2024-07-27 12:25:58.000000 +00:00",

        "deleted": **false**

    }