Task 3: **Building a Simple Messaging Application.**

.

**1. Message Entity and Repository**

import javax.persistence.\*;  
import java.time.LocalDateTime;  
  
@Entity  
public class Message {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private Long senderId;  
 private Long receiverId;  
  
 @Column(nullable = false)  
 private String content;  
  
 private LocalDateTime timestamp;  
  
 private String status; // e.g., "SENT", "READ"  
  
 // Constructors, getters, and setters  
 public Message() {  
 }  
  
 public Message(Long senderId, Long receiverId, String content, LocalDateTime timestamp, String status) {  
 this.senderId = senderId;  
 this.receiverId = receiverId;  
 this.content = content;  
 this.timestamp = timestamp;  
 this.status = status;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public Long getSenderId() {  
 return senderId;  
 }  
  
 public void setSenderId(Long senderId) {  
 this.senderId = senderId;  
 }  
  
 public Long getReceiverId() {  
 return receiverId;  
 }  
  
 public void setReceiverId(Long receiverId) {  
 this.receiverId = receiverId;  
 }  
  
 public String getContent() {  
 return content;  
 }  
  
 public void setContent(String content) {  
 this.content = content;  
 }  
  
 public LocalDateTime getTimestamp() {  
 return timestamp;  
 }  
  
 public void setTimestamp(LocalDateTime timestamp) {  
 this.timestamp = timestamp;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 @Override  
 public String toString() {  
 return "Message{" +  
 "id=" + id +  
 ", senderId=" + senderId +  
 ", receiverId=" + receiverId +  
 ", content='" + content + '\'' +  
 ", timestamp=" + timestamp +  
 ", status='" + status + '\'' +  
 '}';  
 }  
}

import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
  
import java.util.List;  
  
public interface MessageRepository extends JpaRepository<Message, Long> {  
  
 @Query("SELECT m FROM Message m WHERE m.receiverId = :userId")  
 List<Message> findAllByReceiverId(@Param("userId") Long userId);  
  
 // Optionally, for more complex queries (e.g., messages by sender and receiver)  
 // List<Message> findAllBySenderIdAndReceiverId(Long senderId, Long receiverId);  
}

**2. Message Controller**

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.validation.Valid;  
import java.time.LocalDateTime;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/messages")  
public class MessageController {  
  
 @Autowired  
 private MessageRepository messageRepository;  
  
 // Sending a message  
 @PostMapping  
 public ResponseEntity<Message> sendMessage(@Valid @RequestBody MessageRequest messageRequest) {  
 // Get the currently logged-in user's ID  
 Authentication authentication = SecurityContextHolder.getContext().getAuthentication();  
 String currentUsername = authentication.getName();  
 // User currentUser = userRepository.findByUsername(currentUsername);  
 Long currentUserId = 1L; // Replace with actual logic to get user ID  
  
 Message message = new Message();  
 message.setSenderId(currentUserId);  
 message.setReceiverId(messageRequest.getReceiverId());  
 message.setContent(messageRequest.getContent());  
 message.setTimestamp(LocalDateTime.now());  
 message.setStatus("SENT");  
  
 // Validation - Example (using Spring Validation in MessageRequest)  
  
 // if (userRepository.findById(message.getReceiverId()).isEmpty()) {  
 // return new ResponseEntity<>(HttpStatus.NOT\_FOUND);  
 // }  
  
 Message savedMessage = messageRepository.save(message);  
 return new ResponseEntity<>(savedMessage, HttpStatus.CREATED);  
 }  
  
 // Retrieving all messages for a specific user  
 @GetMapping  
 public ResponseEntity<List<Message>> getMessagesForCurrentUser() {  
 // Get the current user's ID  
 Authentication authentication = SecurityContextHolder.getContext().getAuthentication();  
 String currentUsername = authentication.getName();  
 // User currentUser = userRepository.findByUsername(currentUsername);  
 Long currentUserId = 1L; // Replace with actual logic to get user ID  
  
 List<Message> messages = messageRepository.findAllByReceiverId(currentUserId);  
 return ResponseEntity.ok(messages);  
 }  
  
 // Marking a message as READ  
 @PutMapping("/{id}/read")  
 public ResponseEntity<Message> markMessageAsRead(@PathVariable Long id) {  
 // Authorization check: Ensure the current user is the receiver  
 // Get the current user's ID  
 Authentication authentication = SecurityContextHolder.getContext().getAuthentication();  
 String currentUsername = authentication.getName();  
 // User currentUser = userRepository.findByUsername(currentUsername);  
 Long currentUserId = 1L; // Replace with actual logic to get user ID  
  
 Message message = messageRepository.findById(id).orElse(null);  
 if (message == null) {  
 return ResponseEntity.notFound().build();  
 }  
  
 if (!message.getReceiverId().equals(currentUserId)) {  
 return ResponseEntity.status(HttpStatus.FORBIDDEN).build(); // Unauthorized  
 }  
  
 message.setStatus("READ");  
 Message updatedMessage = messageRepository.save(message);  
 return ResponseEntity.ok(updatedMessage);  
 }  
  
 // Exception Handling (example)  
 @ExceptionHandler(Exception.class)  
 public ResponseEntity<String> handleException(Exception e) {  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("An error occurred: " + e.getMessage());  
 }  
}

import lombok.Data;  
  
import javax.validation.constraints.NotEmpty;  
import javax.validation.constraints.NotNull;  
  
@Data  
public class MessageRequest {  
  
 @NotNull(message = "Receiver ID is required")  
 private Long receiverId;  
  
 @NotEmpty(message = "Content is required")  
 private String content;  
}