<dependencies>

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- JWT -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<!-- JPA and Database -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

</dependencies>

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

@JsonIgnore

private String password;

// Getters and setters

}

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

}

@RestController

@RequestMapping("/api")

public class AuthController {

@Autowired

private UserRepository userRepository;

@Autowired

private PasswordEncoder passwordEncoder;

@PostMapping("/register")

public ResponseEntity<?> registerUser(@RequestBody User user) {

if(userRepository.existsByUsername(user.getUsername())) {

return ResponseEntity

.badRequest()

.body("Error: Username is already taken!");

}

user.setPassword(passwordEncoder.encode(user.getPassword()));

userRepository.save(user);

return ResponseEntity.ok("User registered successfully!");

}

}

@Entity

public class Task {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String description;

private String status;

private String priority;

private LocalDate dueDate;

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false)

private User user;

private LocalDateTime createdAt;

private LocalDateTime updatedAt;

// Getters and setters

}

@Repository

public interface TaskRepository extends JpaRepository<Task, Long> {

List<Task> findByUserId(Long userId);

// Add custom queries for filtering and searching if needed

}

@RestController

@RequestMapping("/api/tasks")

public class TaskController {

@Autowired

private TaskRepository taskRepository;

@Autowired

private UserRepository userRepository;

@PostMapping

public Task createTask(@RequestBody Task task, @AuthenticationPrincipal User user) {

task.setUser(user);

task.setCreatedAt(LocalDateTime.now());

task.setUpdatedAt(LocalDateTime.now());

return taskRepository.save(task);

}

@GetMapping

public List<Task> getTasks(@AuthenticationPrincipal User user) {

return taskRepository.findByUserId(user.getId());

}

@PutMapping("/{id}")

public ResponseEntity<Task> updateTask(@PathVariable Long id, @RequestBody Task taskDetails, @AuthenticationPrincipal User user) {

Task task = taskRepository.findByIdAndUserId(id, user.getId())

.orElseThrow(() -> new ResourceNotFoundException("Task not found"));

task.setTitle(taskDetails.getTitle());

task.setDescription(taskDetails.getDescription());

task.setStatus(taskDetails.getStatus());

task.setPriority(taskDetails.getPriority());

task.setUpdatedAt(LocalDateTime.now());

final Task updatedTask = taskRepository.save(task);

return ResponseEntity.ok(updatedTask);

}

@DeleteMapping("/{id}")

public ResponseEntity<Map<String, Boolean>> deleteTask(@PathVariable Long id, @AuthenticationPrincipal User user) {

Task task = taskRepository.findByIdAndUserId(id, user.getId())

.orElseThrow(() -> new ResourceNotFoundException("Task not found"));

taskRepository.delete(task);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return ResponseEntity.ok(response);

}

}

// Example for filtering tasks by status, priority, and due date

@Query("SELECT t FROM Task t WHERE t.user.id = :userId AND t.status = :status AND t.priority = :priority AND t.dueDate <= :dueDate")

List<Task> findTasks(@Param("userId") Long userId, @Param("status") String status, @Param("priority") String priority, @Param("dueDate") LocalDate dueDate);

version: '3'

services:

app:

image: your\_image\_name

build:

context: .

dockerfile: Dockerfile

ports:

- "8080:8080"

depends\_on:

- db

db:

image: mysql:8.0

environment:

MYSQL\_ROOT\_PASSWORD: password

MYSQL\_DATABASE: task\_management\_db

ports:

- "3306:3306"

spring.datasource.url=jdbc:mysql://localhost:3306/task\_management\_db

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true