UNILAK-Kigali

Faculty: CIS

Academic Year: 2024/2025

Module: Advanced Programming

Level of the study: III/Evening

Category of Assessment: Individual Final project

Maximum: 40 marks

Due Date: 15/05/2025,17:59 PM

Lecturer: NGIRIMANA Schadrack

Instructions:
⚠ **Important tips:**

- Missing the following class design requirements will result in a **-1 mark penalty per omission**.
- Plagiarism: **0 marks**
- Late submission: Even 1 minute late = **0 marks**
- JavaDoc or comment annotations in each class indicating your name and registration number
- Class names must be in Capital letters
- Version your API and use meaningful resource names
- Use the appropriate HTTP method and status
- Submit as a zipped project in my DM on Slack

As you embark on developing RESTful APIs using Spring Boot, please adhere to the following guidelines to ensure your project is well-structured, secure, and aligns with best practices:

1. **Implement Security with JSON Web Tokens (JWT):**
    - ○ Use Spring Security to implement authentication and authorization in your application.
    - ○ Ensure JWT tokens are generated and validated securely to protect sensitive resources.
    - ○ Use access tokens for stateless communication between clients and the server.
2. **Password Encryption:**
    - ○ Always encrypt passwords before storing them in the database.
3. **Project Structure and Packages:**
    - ○ Organize your application into well-defined packages for clarity and maintainability. At a minimum, include:
        - ■ controllers: Handle HTTP requests and responses.
        - ■ entities: Represent database tables as Java objects.
        - ■ services: contain business logic.

        - repositories: Interact with the database.
        - payloads: Define request and response data objects.
        - utils: Define utility classes.

4. **Use appropriate HTTP status codes:**
    - Return meaningful status codes for API responses.
5. **API Versioning:**
    - Incorporate API versioning to ensure backward compatibility as your application evolves.
    - Use URI versioning (e.g., /api/v1/resource) or other strategies discussed in class.
6. **Follow Best Practices:**

- Write clean and readable code with proper comments and documentation. Use meaningful names for variables, methods, and endpoints.
- Validate user inputs to prevent security vulnerabilities like SQL injection.
- Employ exception handling to provide informative error messages without exposing sensitive details.
- Get the user ID from the token
- During presentation, if you fail one task, you can't proceed to the next

Tutorials: one of these tutorials can help

1. https://www.geeksforgeeks.org/spring-boot-3-0-jwt-authentication-with-spring-security-using-mysql-database/
2. https://medium.com/@tericcabrel/implement-jwt-authentication-in-a-spring-boot-3-application-5839e4fd8fac
3. https://dev.to/abhi9720/a-comprehensive-guide-to-jwt-authentication-with-spring-boot-117p
4. https://www.youtube.com/watch?v=KxqlJblhzfI

Lost and Found

Project Overview

In today's fast-paced, digitally connected world, losing personal belongings no longer means relying solely on bulletin boards or local announcements. Just as businesses have transitioned from traditional media to digital platforms to reach wider audiences, lost and found systems have also evolved.

Create API endpoints

API Response Specification

The API endpoints should respond with the appropriate HTTP response ***status*** code and a JSON

object that contains either a ***data*** property (on success) or an ***error*** property (on failure). When Presently, the ***data*** property is always an ***object*** *or an* ***array***.

On Success

```
{
  "status": integer,
“message”:string
  "data": {...} | [...]
}
```

On Error

```
{
  “status”  integer
  “error”  “relevant-error-message”
}
```

Entity Specification

Users

```
{
  "id": integer,
  "email": string,
  "first_name": string,
  "last_name": string,
  "password": string,
  "phone_number": string,
  "address": string,
  "is_admin": boolean,
  "is_banned": boolean,
  "created_at": datetime,
  "updated_at": datetime
}
```

Found item

```
{
 "id": integer,
 "user_id": integer,
 "title": string,
 "description": string,
 "category": string,
 "location": string,
 "image_url": string,
 "found_date": datetime,
 "storage_location": string,
 "status": string, // "pending", "active", "claimed", "rejected"
 "created_at": datetime,
 "updated_at": datetime
}
```

Lost items

```
{
 "id": integer,
 "user_id": integer,
 "title": string,
 "description": string,
 "category": string,
 "location": string,
 "image_url": string,
 "lost_date": datetime,
 "status": string, // "pending", "active", "claimed", "rejected"
 "created_at": datetime,
 "updated_at": datetime
}
```

API Endpoint Specification

1. **Endpoint**: POST /auth/signup

   Create user account

Request spec:

```
{
        "email": string,
        "first_name": string,
        "last_name": string,
        "phone_number": string,
        "address": string,
}
```

Response spec:

```
{
  “status”   integer
 “data”   {
      "id": integer,
       "email": string,
       "first_name": string,
       "last_name": string,
       "phone_number": string,
       "address": string,
       "is_admin": boolean,
       "is_banned": boolean,
       "created_at": datetime,
       "updated_at": datetime
   }
}
```

2. **Endpoint**: POST /auth/signin

Login a user

Request spec:

```
{
        "email": string,
        "password": string,
}
```

Response spec:

```
{
  “status”   integer
  “token”   “45erkjherht45495783”
  “data”   {
     "id": integer,
      "email": string,
      "first_name": string,
      "last_name": string,
      "phone_number": string,
      "address": string,
      "is_admin": boolean,
      "is_banned": boolean,
      "created_at": datetime,
      "updated_at": datetime
   }
}
```

3. **Endpoint**: POST /lostitems

User can create a lost item.

Request spec:

```
{
            "title": string,
            "description": string,
            "category": string,
            "location": string,
            "image_url": string,
            "lost_date": datetime,
            "status": string,
            "created_at": datetime,
            "updated_at": datetime
}
```

Response spec:

```
{
  “status”  integer
  "message": string,
  “data”  {
          "id": integer,
           "user_id": integer,
           "title": string,
           "description": string,
           "category": string,
           "location": string,
           "image_url": string,
           "lost_date": datetime,
           "status": string, // "pending", "active", "claimed", "rejected"
           "created_at": datetime,
           "updated_at": datetime
   }
}
```

4. **Endpoint**: POST /founditems

   User can create a found item.

Request spec:

```
{
          "title": string,
          "description": string,
          "category": string,
          "location": string,
          "image_url": string,
          "found_date": datetime,
          "storage_location": string,
          "status": string, // "pending", "active", "claimed", "rejected"
}
```

Response spec:

```
{
  “status”  integer
  "message": string,
  “data”  {
          "id": integer,
           "user_id": integer,
           "title": string,
           "description": string,
           "category": string,
           "location": string,
           "image_url": string,
           "found_date": datetime,
           "storage_location": string,
           "status": string, // "pending", "active", "claimed", "rejected"
           "created_at": datetime,
           "updated_at": datetime
   }
}
```

5. **Endpoint**: PATCH /found-items/<:id>

   User can update found item data.

   Note: Include any field you will like to update in your request object and only update those fields.

Response spec:

```
{
  “status”  ‘integer’
  “data”  {
          "id": integer,
           "user_id": integer,
           "title": string,
           "description": string,
           "category": string,
           "location": string,
           "image_url": string,
           "found_date": datetime,
           "storage_location": string,
           "status": string,
           "created_at": datetime,
           "updated_at": datetime
   }
}
```

6. **Endpoint**: GET /found-items/<:id>

   User can get details for the found items

Response spec:

```
{
  “status”  ‘integer’
  “data”  {
          "id": integer,
           "user_id": integer,
           "title": string,
           "description": string,
           "category": string,
           "location": string,
           "image_url": string,
           "found_date": datetime,
           "storage_location": string,
           "status": string,
           "created_at": datetime,
           "updated_at": datetime
  }
}
```

7. **Endpoint**: PATCH /lost-items/<:id>

   User can update lost item data.

   Note: Include any field you would like to update in your request object and only update those fields.

Response spec:

```
{
  “status”  ‘integer’
  “data”  {
          "id": integer,
           "user_id": integer,
           "title": string,
           "description": string,
           "category": string,
           "location": string,
           "image_url": string,
           "found_date": datetime,
           "status": string,
           "created_at": datetime,
           "updated_at": datetime
  }
}
```

8. **Endpoint**: GET /lost-items/<:id>

User can get details for the lost item

Response spec:

```
{
  “status”  ‘integer’
  “data”  {
          "id": integer,
           "user_id": integer,
           "title": string,
           "description": string,
           "category": string,
           "location": string,
           "image_url": string,
           "found_date": datetime,
           "status": string,
           "created_at": datetime,
           "updated_at": datetime
  }
}
```

9. **Endpoint**: DELETE /found-items/<:idt>

User can selete a found item..

Response spec:

```
{
  “status”  integer
  “data”  {
    “message”  String
  }
}
```

10. **Endpoint**: DELETE /lost-items/<:idt>

User can delete a lost item..

Response spec:

```
{
  “status”  integer
  “data”  {
    “message”  String
  }
}
```

11. Endpoint: GET

/search?type=lost|found&string&location=string&&start_date=datetime&end_date=datetime

User can search lost/found items using location, date range

Response spec:

```
{
  “status”  Integer
  “data”  [{
        Lost or found item object
  }, {
  Lost or found item object
  }]
}
```

12. **Endpoint**: PATCH /admin/users/<:id>?

Admin can update user (ban/unban)

Request spec:

```
{
     "is_banned": boolean
}
```

Response spec:

```
{
  “status”  integer
  “data”  {
    "id": integer,
     "email": string,
     "first_name": string,
     "last_name": string,
     "phone_number": string,
     "address": string,
     "is_admin": boolean,
     "is_banned": boolean,
     "created_at": datetime,
     "updated_at": datetime
  }
```

13. **Endpoint**: PATCH /admin/items/<:id>?

Admin can approve/reject the item

Request spec:

```
{
        "status": "approved"|"rejected",
         "type": "lost"|"found"
}
```

Response spec:

```
{
  “status”  integer
  “data”  {
     "id": integer,
      "user_id": integer,
      "title": string,
      "description": string,
      "category": string,
      "location": string,
      "image_url": string,
      "found_date": datetime,
      "status": string,
      "created_at": datetime,
      "updated_at": datetime
  }
```

14. **Endpoint**: GET /admin/reports

Admin can get system reports

Response spec:

```
{
  “status”  integer
  “message”  String
  “data”  {
        "total_users": integer,
          "active_users": integer,
          "banned_users": integer,
          "total_lost_items": integer,
          "claimed_lost_items": integer,
          "total_found_items": integer,
          "claimed_found_items": integer
        }
}
```