Project Report: Kamero Research Base

Table of Contents

Introduction

System Architecture - Backend

Backend

The backend is built using **Node.js** and **Express.js**. It follows a RESTful API design to handle HTTP requests and manage data interactions between the frontend and the database. Key components include:

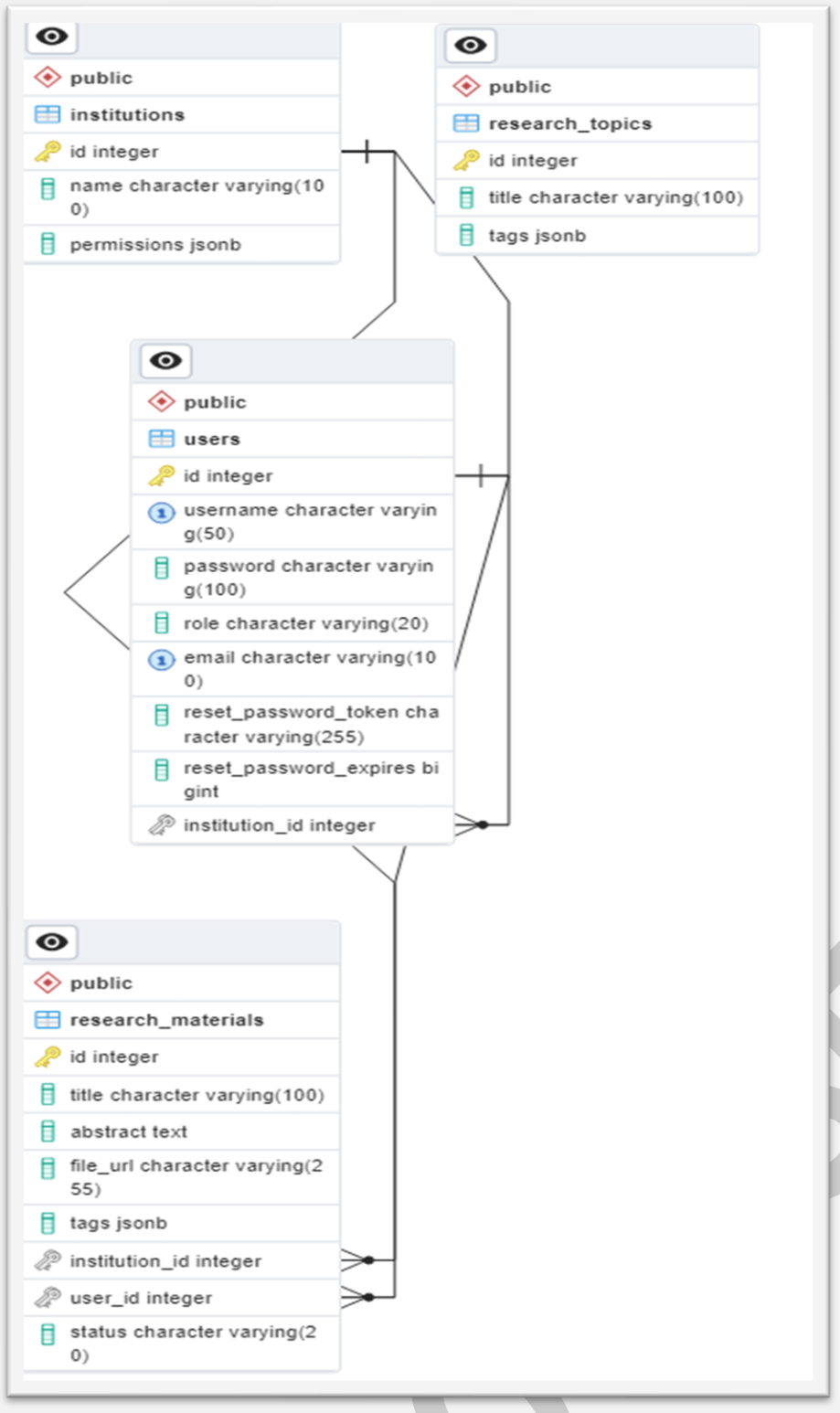
- **Controllers**: Handle the core business logic. For example, `lecturerController` manages lecturer-related functionalities.
- **Middleware**: Includes authentication and authorization logic, such as `authenticateToken`.
- **Database Interaction**: Uses `pg` (node-postgres) to interact with a PostgreSQL database called Kamero-research-base.

Database

The system uses **PostgreSQL** as the relational database management system (RDBMS). The schema includes tables for `users`, `institutions`, `research_materials`, and `research_topics`.

- **Institutions Table**: Stores information about institutions.
- **Research Materials Table**: Stores uploaded research files and metadata.
- **Users Table**: Stores user credentials and roles.
- **Research Topics Table**: Stores different research topics.

See ER – Diagram Below



Design Patterns and Principles

The project follows several design principles:

- **Separation of Concerns**: Different functionalities are separated into distinct modules (e.g., controllers, routes, services).
- **DRY (Don't Repeat Yourself)**: Common code is abstracted into reusable functions.
- **MVC (Model-View-Controller)**: The project loosely follows the MVC pattern, separating the data layer (models), the business logic (controllers), and the presentation layer (views).
- **RESTful Services**: The backend adheres to REST principles, making the API stateless and resource-based.

---

1. Process Flow

Overview:

- **Users:** Admin, Lecturer, Student.
- **Components:** Controllers (`adminController`, `lecturerController`, `studentController`), Routes (Views/Responses), Models (User, Research Material, Institution), Database.

2. Process Flow for Each User

- **Admin:**
    1. **Login/Register** - Admin logs into the system using credentials.
    2. **Manage Users** - Admin manages users (create, read, update, delete).
    3. **Manage Institutions** - Admin manages institutions.
    4. **View System Analytics** - Admin accesses system-wide analytics.
- **Lecturer:**
    1. **Login/Register** - Lecturer logs into the system.
    2. **View Assigned Institutions** - Lecturer views the institutions they are associated with.
    3. **Upload Research** - Lecturer uploads research materials to the system.
    4. **Review Student Uploads** - Lecturer reviews research materials uploaded by students.
    5. **View Institution Research** - Lecturer views all research materials associated with their institution.
- **Student:**
    1. **Login/Register** - Student logs into the system.
    2. **View Available Research** - Student views available research materials.
    3. **Upload Research** - Student uploads their research materials.

3. Process Flow Diagram

To visualize this:

- **Admin**:
    - Connects to `authController` -> Validates credentials.
    - Interacts with `adminController` -> Calls specific methods (e.g., `manageUsers`, `manageInstitutions`, `viewSystemAnalytics`).
    - `adminController` communicates with corresponding Models (e.g., `User`, `Institution`).
    - Models interact with the Database for CRUD operations.
    - Responses sent back to the admin (Success/Error).
- **Lecturer**:
    - Connects to `authController` -> Validates credentials.
    - Interacts with `lecturerController` -> Calls specific methods (e.g., `viewAssignedInstitutions`, `uploadResearch`, `reviewStudentUploads`, `viewInstitutionResearch`).
    - `lecturerController` communicates with Models (`Institution`, `ResearchMaterial`).
    - Models interact with the Database for data retrieval or updates.
    - Responses sent back to the Lecturer (Success/Error).
- **Student**:
    - Connects to `authController` -> Validates credentials.
    - Interacts with `studentController` -> Calls specific methods (e.g., `viewAvailableResearch`, `uploadResearch`).
    - `studentController` communicates with `ResearchMaterial` model.
    - Models interact with the Database for data retrieval or uploads.
    - Responses sent back to the student (Success/Error).

Testing

How the testing was conducted

Unit Testing

Unit tests are written to validate individual components of the system, such as controller methods and utility functions. The tests ensure that each function behaves as expected in isolation. See the test cases covered below.

adminController.test.js:

Test Case	Expected Response	Reason for Testing
manageUsers - create user	Status 201, created user object	Verify user creation functionality
manageUsers - read user	Status 200, user object	Verify user retrieval functionality
manageUsers - read non-existent user	Status 404, error message	Verify proper handling of non-existent users
manageUsers - update user	Status 200, updated user object	Verify user update functionality
manageUsers - delete user	Status 204	Verify user deletion functionality
manageUsers - invalid action	Status 400, error message	Verify handling of invalid actions
manageUsers - error handling	Status 500, error message	Verify proper error handling
manageInstitutions - create institution	Status 201, created institution object	Verify institution creation functionality
manageInstitutions - read institution	Status 200, institution object	Verify institution retrieval functionality
manageInstitutions - read non-existent institution	Status 404, error message	Verify proper handling of non-existent institutions
manageInstitutions - update institution	Status 200, updated institution object	Verify institution update functionality
manageInstitutions - delete institution	Status 204	Verify institution deletion functionality
manageInstitutions - invalid action	Status 400, error message	Verify handling of invalid actions
manageInstitutions - error handling	Status 500, error message	Verify proper error handling
manageResearchTopics - create topic	Status 201, created topic object	Verify research topic creation functionality
manageResearchTopics - read topic	Status 200, topic object	Verify research topic retrieval functionality
manageResearchTopics - read non-existent topic	Status 404, error message	Verify proper handling of non-existent topics
manageResearchTopics - update topic	Status 200, updated topic object	Verify research topic update functionality
manageResearchTopics - delete topic	Status 204	Verify research topic deletion functionality
manageResearchTopics - invalid action	Status 400, error message	Verify handling of invalid actions
manageResearchTopics - error handling	Status 500, error message	Verify proper error handling
viewSystemAnalytics	Status 200, analytics object	Verify system analytics retrieval functionality
viewSystemAnalytics - error handling	Status 500, error message	Verify proper error handling

```
PS C:\Users\Greenfield\Desktop\Kamero research base> jest AdminController.test.js
  console.error
    Error in manageUsers: Something went wrong

      51 |         }
      52 |     } catch (error) {
    > 53 |         console.error('Error in manageUsers:', error.message);
         |                 ^
      54 |         res.status(500).json({ message: error.message});
    √ should update a user (1 ms)
    √ should delete a user (1 ms)
    √ should return 400 for invalid action
    √ should handle errors (30 ms)
  manageInstitutions
    √ should create an institution
    √ should read an institution (1 ms)
    √ should return 404 if institution not found
    √ should update an institution
    √ should delete an institution (1 ms)
    √ should return 400 for invalid action (1 ms)
    √ should handle errors
  manageResearchTopics
    √ should create a research topic (1 ms)
    √ should read a research topic (1 ms)
    √ should return 404 if research topic is not found
    √ should update a research topic (1 ms)
    √ should delete a research topic (1 ms)
    √ should return 400 for invalid action
    √ should handle errors (1 ms)
  viewSystemAnalytics
    √ should return system analytics (1 ms)
    √ should handle errors

Test Suites: 1 passed, 1 total
Tests:       23 passed, 23 total
Snapshots:   0 total
Time:        0.479 s, estimated 3 s
Ran all test suites matching /AdminController.test.js/i.
PS C:\Users\Greenfield\Desktop\Kamero research base>
```

---

authController.test.js:

Test Case	Expected Response	Reason for Testing
forgotPassword - generate reset token	Status 200, reset token	Verify password reset token generation
forgotPassword - error handling	Status 500, error message	Verify proper error handling
resetPassword - invalid token	Status 400, error message	Verify handling of invalid reset tokens
resetPassword - valid token	Status 200, success message	Verify password reset functionality
resetPassword - error handling	Status 500, error message	Verify proper error handling
login - user not found	Status 401, error message	Verify handling of non-existent users
login - successful	JWT token, user object	Verify successful login functionality
login - error handling	Status 500, error message	Verify proper error handling
register - successful	Status 201, user object	Verify user registration functionality
register - error handling	Status 500, error message	Verify proper error handling

```
PS C:\Users\Greenfield\Desktop\Kamero research base> jest authController.test.js
 PASS  TESTS/controllers/authController.test.js
  Auth Controller
    forgotPassword
      √ should generate a reset token and update user in the database (4 ms)
      √ should return 500 in case of an error (1 ms)
  resetPassword
    √ should return 400 if the token is invalid or expired
    √ should reset the user's password and remove the token
    √ should return 500 in case of an error (1 ms)
  login
    √ should return 401 if the user is not found (1 ms)
    √ should return a JWT token on successful login
    √ should return 500 in case of an error
  register
    √ should create a new user and return 201
    register
      √ should return 500 in case of an error

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        0.528 s, estimated 3 s
Ran all test suites matching /authController.test.js/i.
```

---

lecturerController.test.js:

Test Case	Expected Response	Reason for Testing
GET /lecturer/institutions	Status 200, institutions array	Verify retrieval of assigned institutions
GET /lecturer/institutions - error	Status 500, error message	Verify proper error handling
POST /lecturer/research	Status 201, research object	Verify research upload functionality
POST /lecturer/research - error	Status 500, error message	Verify proper error handling
GET /lecturer/student-uploads	Status 200, uploads array	Verify retrieval of student uploads
GET /lecturer/student-uploads - error	Status 500, error message	Verify proper error handling
GET /lecturer/research	Status 200, research array	Verify retrieval of institution research
GET /lecturer/research - error	Status 500, error message	Verify proper error handling

```
PS C:\Users\Greenfield\Desktop\Kamero research base> jest lecturerController.test.js
 PASS  TESTS/controllers/lecturerController.test.js
  Lecturer Controller
    GET /lecturer/institutions
      √ should return assigned institutions for the lecturer (16 ms)
      √ should handle errors when fetching institutions (5 ms)
    POST /lecturer/research
      √ should allow lecturer to upload research (18 ms)
      √ should handle errors when uploading research (3 ms)
    GET /lecturer/student-uploads
      √ should return student uploads for review (4 ms)
      √ should handle errors when fetching student uploads (4 ms)
    GET /lecturer/research
      √ should return research related to the lecturer's institution (4 ms)
      √ should handle errors when fetching institution research (3 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        0.676 s, estimated 4 s
Ran all test suites matching /lecturerController.test.js/i.
```

---

studentController.test.js:

Test Case	Expected Response	Reason for Testing
GET /api/research	Status 200, research array	Verify retrieval of available research

GET /api/research - error	Status 500, error message	Verify proper error handling
POST /api/research - valid PDF	Status 201, research object	Verify research upload functionality
POST /api/research - invalid file type	Status 400, error message	Verify handling of invalid file types
POST /api/research - database error	Status 500, error message	Verify proper error handling

```
PS C:\Users\Greenfield\Desktop\Kamero research base> jest studentController.test.js
 PASS  TESTS/controllers/studentController.test.js
  GET /api/research
    √ should return a list of available research (15 ms)
    √ should return 500 if there is a server error (4 ms)
  POST /api/research
    √ should upload a PDF file and save the research details (11 ms)
    √ should return 400 if file type is not PDF (5 ms)
    √ should return 500 if there is a database error (7 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.971 s, estimated 4 s
Ran all test suites matching /studentController.test.js/i.
PS C:\Users\Greenfield\Desktop\Kamero research base>
```

Test Coverage

All functions were covered.

Coding Conventions

The project adheres to the following coding conventions:

- **Indentation and Spacing**: Proper indentation and spacing are maintained for readability.
- **Linting**: **ESLint** enforces coding standards and catches potential issues early.
- **File Naming**: Consistent naming conventions are used across the project (e.g., camelCase for variables, PascalCase for classes).
- **Error Handling**: Proper error handling mechanisms are in place, ensuring that the system gracefully handles unexpected scenarios.
- **Function Length**: Functions are kept short and focused on a single task.
- **Comments**: Important functions and complex logic are well-documented with comments.

Documentation

Code Documentation

- **JSDoc** is used for documenting functions, classes, and methods.
- Inline comments explain complex logic or non-obvious code snippets.

Deployment:

The project could be set up for deployment on cloud platforms like AWS, G-Cloud, or Azure.

Continuous Integration/Continuous Deployment (CI/CD) pipelines have already been established using GitHub actions to automate testing, and could be extended to deployment.

Future Additions and Enhancements

Several features can be added to improve the system:

1. **Role-Based Access Control (RBAC)**: Implement finer-grained permissions based on user roles.
2. **Notifications System**: Add real-time notifications for lecturers and students.
3. **Advanced Search**: Implement a search functionality that allows filtering research materials by tags, title, or institution.
4. **Analytics Dashboard**: Provide lecturers with a dashboard to view statistics about research submissions and reviews.
5. **Multi-Language Support**: Add internationalization (i18n) to support multiple languages.

Conclusion

This report provides a comprehensive overview of Kamero Research Base, covering the system architecture, design patterns, testing strategies, coding conventions, and future improvements. Following the best practices outlined in this report, the system can be maintained and scaled effectively, ensuring its long-term success.