Documentation for the “Brain Rush” Project

Team Brain Rush (Alieva Nargiza and Kydykbekova Aidai comes - 23)
December 11, 2024

This document provides a detailed overview of the Brain Rush project. It covers the conceptual, logical, and physical design of the system, the database structure, the backend (Java Spring Boot) and frontend (Thymeleaf) implementation details, and provides code snippets, usage instructions, and various aspects related to system setup and operation.

Contents

1. **Project Overview**
   - 1.1 Title
   - 1.2 Objective:
2. **System Architecture**
   - 2.1 Technologies Used
3. **Conceptual Design**
   - 3.1 Entities and Attributes
   - 3.2 Relationships
4. **Logical Design**
   - 4.1 Database Tables
   - 4.2 Database Constraints
5. **Database Setup**
   - 5.1 Database Management System
   - 5.2 SQL Scripts
     - 5.2.1 DDL Scripts
     - 5.2.2 DML Scripts
6. **Backend Implementation (Java Spring Boot)**
   - 6.1 Overview
   - 6.2 Core Features
   - 6.3 Sample RESTful API Endpoints
     - 6.3.1 Category API
     - 6.3.2 Course API
     - 6.3.3 Student API
     - 6.3.4 Feedback API
7. **How to run project**
8. **How to Maintenance project**
9. **Conclusion**

1. Project Overview

1.1 Title:

Smart E-Learning Management System

1.2 Objective:

About the Project

The Brain Rush application is an advanced online learning platform designed to facilitate interactive and engaging educational experiences. Its main purpose is to provide a comprehensive environment where users—whether students, instructors, or administrators—can access, manage, and participate in a wide range of courses and learning resources. The platform aims to bridge the gap between traditional education and modern digital learning by offering features that enhance user experience, promote effective learning, and support diverse learning styles.

Purpose of the Project: The primary purpose of the Brain Rush application is to democratize access to high-quality education by providing a centralized platform.

The Brain Rush application is built with the goal of creating a seamless and intuitive user experience, allowing users to focus on their educational goals without the distraction of complex navigation and functionality. It leverages modern technologies and design principles to offer a responsive, accessible, and scalable platform for online learning, making it suitable for use in various educational contexts, from individual learning to large-scale institutional deployments.

2. System Architecture

2.1 Technologies Used:

- ***Backend***: Java with Spring Boot (Spring MVC, Spring Data JPA, Spring web, Flyway)
- ***Frontend***: Thymeleaf templates for dynamic web pages
- ***Database***: PostgreSQL
- ***API***: RESTful services to manage data exchange between client and server

3. Conceptual Design

This section provides an overview of the main entities involved in the Brain Rush application and their respective attributes. Understanding these entities and their relationships is crucial for designing and developing the system effectively.

3.1 Entities and Attributes:

1. **Course**:
    - **ID (PK)**: Unique identifier for each course.
    - **Title**: The title of the course.
    - **Description**: A brief overview or syllabus of the course.
    - **Duration**: The length of the course in hours or weeks.
    - **Price**: Cost to enroll in the course.
    - **Category**: The category or subject to which the course belongs (e.g., Programming, Business, Arts).
    - **InstructorID (FK)**: Foreign key linking to the Instructor entity, identifying the course's instructor.
    - **TotalEnrollments (calculated)**: The total number of students who have enrolled in the course.
    - **AverageRating (calculated)**: The average rating given by students who have completed the course.
2. **Student**:
    - **ID (PK)**: Unique identifier for each student.
    - **Name**: The full name of the student.
    - **Email**: Contact email of the student.
    - **Bio**: A brief biography or description of the student.
    - **EnrollmentHistory**: A list of course enrollments by the student (linked to Enrollment entity).
    - **TotalEnrollments (calculated)**: The total number of courses the student has enrolled in.
    - **AverageRating (calculated)**: The average rating received from feedback across all courses.
3. **Instructor**:
    - **ID (PK)**: Unique identifier for each instructor.
    - **Name**: The full name of the instructor.
    - **Email**: Contact email of the instructor.
    - **Bio**: A brief biography or background of the instructor.
    - **Rating (calculated)**: The average rating given by students across all courses taught by the instructor.
4. **Enrollment**:
    - **ID (PK)**: Unique identifier for each enrollment record.
    - **StudentID (FK)**: Foreign key linking to the Student entity, identifying the enrolled student.
    - **CourseID (FK)**: Foreign key linking to the Course entity, identifying the course.
    - **EnrollmentDate**: The date when the student enrolled in the course.
    - **CompletionStatus**: The current status of the course for the student (e.g., in-progress, completed).
5. **Feedback**:
    - **ID (PK)**: Unique identifier for each feedback record.
    - **StudentID (FK)**: Foreign key linking to the Student entity, identifying the student giving the feedback.

    - **CourseID (FK)**: Foreign key linking to the Course entity, identifying the course related to the feedback.
    - **Rating**: The rating given by the student (on a scale, e.g., 1 to 5).
    - **Comment**: Additional comments provided by the student about the course.
    - **FeedbackDate**: The date when the feedback was submitted.
6. **Category**:
    - **ID (PK)**: Unique identifier for each category.
    - **Name**: The name of the category (e.g., Programming, Business, Arts).

This conceptual design establishes the core entities and their attributes, which serve as the foundation for the database structure and business logic of the Brain Rush application. Each entity is interrelated, enabling seamless data interaction and analysis across the platform, from course offerings and student progress to instructor evaluations and feedback mechanisms.

3.2 Relationships

- **One-to-Many Relationship**:
    - **Instructor to Course**: An instructor can teach multiple courses. The Instructor_ID in the Course table references the ID in the Instructor table. If an instructor is deleted, the Instructor_ID in all related courses will be set to NULL.
    - **Category to Course**: A course belongs to a specific category (e.g., Programming, Data Science). The Category_ID in the Course table references the ID in the Category table. If a category is deleted, the Category_ID in all related courses will be set to NULL.
- **Many-to-One Relationship**:
    - **Student to Enrollment**: A student can be enrolled in multiple courses. The Student_ID in the Enrollment table references the ID in the Student table. If a student is deleted, all related enrollments will be deleted due to the cascade delete constraint.
    - **Course to Enrollment**: A course can have multiple students enrolled. The Course_ID in the Enrollment table references the ID in the Course table. If a course is deleted, all related enrollments will be deleted due to the cascade delete constraint.
    - **Student to Feedback**: A student can give feedback for multiple courses. The Student_ID in the Feedback table references the ID in the Student table. If a student is deleted, all related feedback entries will be deleted due to the cascade delete constraint.
    - **Course to Feedback**: A course can receive feedback from multiple students. The Course_ID in the Feedback table references the ID in the Course table. If a course is deleted, all related feedback entries will be deleted due to the cascade delete constraint.

4. Logical Design

4.1 Database Tables:

- Course Table: Stores course-related details.
- Student Table: Stores student-related details.
- Instructor Table: Stores instructor details.
- Enrollment Table: Maps students to their enrolled courses.
- Feedback Table: Stores feedback left by students for courses.
- Category Table: Categorizes courses into different groups like Tech- nology, Business, etc.

4.2 Database Constraints

- **Primary Keys**:
    - ID in Instructor, Category, Course, Student, Enrollment, Feedback tables.
- **Foreign Key Constraints**:
    - Instructor_ID in Course table references ID in Instructor table.
    - Category_ID in Course table references ID in Category table.
    - Student_ID in Enrollment table references ID in Student table (Cascade delete).
    - Course_ID in Enrollment table references ID in Course table (Cascade delete).
    - Student_ID in Feedback table references ID in Student table (Cascade delete).
    - Course_ID in Feedback table references ID in Course table (Cascade delete).
- **Unique Constraints**:
    - Name in Instructor table.
    - Email in Instructor and Student tables.
    - Name in Category table.
- **Check Constraints**:
    - Total_Enrollments in Course table must be non-negative.
    - Average_Rating in Course table must be between 0 and 5.
    - Rating in Feedback table must be between 0 and 5.

5. Database Setup

5.1 Database Management System:

PostgreSQL is used as the relational database management system.

5.2 SQL Scripts

5.2.1 DDL Scripts

```sql
CREATE DATABASE brain_rush;

CREATE TABLE IF NOT EXISTS Instructor (
   ID BIGSERIAL PRIMARY KEY,
   Name VARCHAR(100) UNIQUE NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL,
   Bio TEXT,
   Registration_Date DATE DEFAULT CURRENT_DATE);

CREATE TABLE IF NOT EXISTS Category (
   ID BIGSERIAL PRIMARY KEY,
   Name VARCHAR(100) UNIQUE NOT NULL,
   Description TEXT);

CREATE TABLE IF NOT EXISTS Course (
   ID BIGSERIAL PRIMARY KEY,
   Title VARCHAR(200) NOT NULL,
   Description TEXT,
   Duration INT NOT NULL,
   Price INT NOT NULL,
   Instructor_ID BIGINT REFERENCES Instructor(ID) ON DELETE SET NULL,
   Category_ID BIGINT REFERENCES Category(ID) ON DELETE SET NULL,
   Total_Enrollments INT DEFAULT 0 CHECK (Total_Enrollments >= 0),
   Average_Rating FLOAT DEFAULT 0.0 CHECK (Average_Rating BETWEEN 0 AND 5),
   Creation_Date DATE DEFAULT CURRENT_DATE);

CREATE TABLE IF NOT EXISTS Student (
   ID BIGSERIAL PRIMARY KEY,
   Name VARCHAR(100) UNIQUE NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL,
   Date_Of_Birth DATE NOT NULL,
   Registration_Date DATE DEFAULT CURRENT_DATE);

CREATE TABLE IF NOT EXISTS Enrollment (
   ID BIGSERIAL PRIMARY KEY,
   Student_ID BIGINT REFERENCES Student(ID) ON DELETE CASCADE,
   Course_ID BIGINT REFERENCES Course(ID) ON DELETE CASCADE,
   Enrollment_Date DATE DEFAULT CURRENT_DATE,
   Completion_Status BOOLEAN DEFAULT FALSE,
   UNIQUE (Student_ID, Course_ID));
```

```
CREATE TABLE IF NOT EXISTS Feedback (
   ID BIGSERIAL PRIMARY KEY,
   Student_ID BIGINT REFERENCES Student(ID) ON DELETE CASCADE,
   Course_ID BIGINT REFERENCES Course(ID) ON DELETE CASCADE,
   Rating FLOAT NOT NULL CHECK (Rating BETWEEN 0 AND 5),
   Comment TEXT,
   Feedback_Date DATE DEFAULT CURRENT_DATE
);
```

5.2.2 DML Scripts

```
-- Function to update `Total_Enrollments` in `Course` table when a new enrollment is inserted
CREATE OR REPLACE FUNCTION update_total_enrollments_on_insert()
RETURNS TRIGGER AS $$
BEGIN
   UPDATE Course
   SET Total_Enrollments = Total_Enrollments + 1
   WHERE ID = NEW.Course_ID;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
-- Trigger to execute the function `update_total_enrollments_on_insert()` after an INSERT on
`Enrollment`
CREATE TRIGGER after_enrollment_insert
   AFTER INSERT ON Enrollment
   FOR EACH ROW
   EXECUTE FUNCTION update_total_enrollments_on_insert();


-- Function to update `Total_Enrollments` in `Course` table when an enrollment is deleted
CREATE OR REPLACE FUNCTION update_total_enrollments_on_delete()
RETURNS TRIGGER AS $$
BEGIN
   UPDATE Course
   SET Total_Enrollments = Total_Enrollments - 1
   WHERE ID = OLD.Course_ID;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;
-- Trigger to execute the function `update_total_enrollments_on_delete()` after a DELETE on
`Enrollment`
CREATE TRIGGER after_enrollment_delete
   AFTER DELETE ON Enrollment
   FOR EACH ROW
   EXECUTE FUNCTION update_total_enrollments_on_delete();


-- Function to update `AverageRating` in `Course` table when feedback is inserted
CREATE OR REPLACE FUNCTION update_average_rating_on_insert()
RETURNS TRIGGER AS $$
BEGIN
   UPDATE Course
```

```
    SET Average_Rating = (
        SELECT COALESCE(AVG(Rating), 0)
        FROM Feedback
        WHERE Course_ID = NEW.Course_ID
    )
    WHERE ID = NEW.Course_ID;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
-- Trigger to execute the function `update_average_rating_on_insert()` after an INSERT on
`Feedback`
CREATE TRIGGER after_feedback_insert
    AFTER INSERT ON Feedback
    FOR EACH ROW
    EXECUTE FUNCTION update_average_rating_on_insert();


-- Function to update `AverageRating` in `Course` table when feedback is deleted
CREATE OR REPLACE FUNCTION update_average_rating_on_delete()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Course
    SET Average_Rating = (
        SELECT COALESCE(AVG(Rating), 0)
        FROM Feedback
        WHERE Course_ID = OLD.Course_ID
    )
    WHERE ID = OLD.Course_ID;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
-- Trigger to execute the function `update_average_rating_on_delete()` after a DELETE on
`Feedback`
CREATE TRIGGER after_feedback_delete
    AFTER DELETE ON Feedback
    FOR EACH ROW
    EXECUTE FUNCTION update_average_rating_on_delete();

-- Inserting data

INSERT INTO Instructor (Name, Email, Bio) VALUES
    ('John Doe', 'john.doe@example.com', 'Experienced data scientist with 10+ years in the
field.'),
    ('Jane Smith', 'jane.smith@example.com', 'Full-stack web developer and coding enthusiast.'),
    ('Michael Brown', 'michael.b@example.com', 'Expert in AI and robotics, PhD in Computer
Science.'),
    ('Laura Wilson', 'laura.w@example.com', 'Creative designer with a passion for user
experiences.'),
    ('Robert Davis', 'robert.d@example.com', 'Certified ethical hacker with 8+ years of
experience.'),
    ('Emily Clark', 'emily.c@example.com', 'PMP-certified professional with a focus on agile
methodologies.'),
    ('David Johnson', 'david.j@example.com', 'AWS and Azure expert with a knack for scalable
solutions.'),
    ('Sophia Martinez', 'sophia.m@example.com', 'Mobile app developer specializing in Android
and iOS.');

INSERT INTO Category (Name, Description) VALUES
    ('Programming', 'Courses related to software development and programming languages.'),
    ('Data Science', 'Courses about data analysis, visualization, and machine learning.'),
    ('UI/UX Design', 'Courses about designing user interfaces and enhancing user experiences.'),
    ('Cybersecurity', 'Courses on information security and ethical hacking.'),
    ('Project Management', 'Courses focused on managing projects and teams.'),
    ('Cloud Computing', 'Courses on cloud platforms like AWS, Azure, and GCP.'),
    ('Mobile Development', 'Courses on building Android and iOS applications.'),
    ('Artificial Intelligence', 'Courses on machine learning, AI, and robotics.');

INSERT INTO Course (Title, Description, Duration, Price, Instructor_ID, Category_ID)
VALUES
    ('Python for Beginners', 'Learn Python programming from scratch.', 30, 99.99, 1, 1),
    ('Machine Learning Basics', 'An introduction to machine learning concepts.', 40, 149.99, 1,
2),
    ('Advanced Web Development', 'Deep dive into full-stack development.', 60, 199.99, 2, 1),
    ('Ethical Hacking 101', 'Basics of cybersecurity and ethical hacking.', 45, 129.99, 5, 4),
    ('Design Thinking', 'Learn the principles of design thinking.', 25, 49.99, 4, 3),
    ('Agile Project Management', 'Master agile project management methodologies.', 35, 79.99,
6, 5),
    ('Cloud Fundamentals', 'Introduction to AWS and Azure platforms.', 50, 119.99, 7, 6),
    ('AI for Beginners', 'Learn the basics of artificial intelligence.', 40, 139.99, 3, 8);

INSERT INTO Student (Name, Email, Date_Of_Birth) VALUES
    ('Alice Johnson', 'alice.j@example.com', '2002-05-15'),
    ('Bob Williams', 'bob.w@example.com', '1999-11-25'),
    ('Charlie Brown', 'charlie.b@example.com', '2003-02-10'),
    ('Diana Clark', 'diana.c@example.com', '2000-07-22'),
    ('Ethan Davis', 'ethan.d@example.com', '2001-09-18'),
    ('Fiona Martinez', 'fiona.m@example.com', '2000-12-03'),
    ('George Wilson', 'george.w@example.com', '1998-06-28'),
    ('Hannah Smith', 'hannah.s@example.com', '2002-01-14');

INSERT INTO Enrollment (Student_ID, Course_ID, Completion_Status) VALUES
    (1, 1, FALSE),
    (1, 2, TRUE),
    (2, 3, FALSE),
    (3, 4, TRUE),
    (4, 5, FALSE),
    (5, 6, FALSE),
    (6, 7, TRUE),
    (7, 8, FALSE);

INSERT INTO Feedback (Student_ID, Course_ID, Rating, Comment) VALUES
    (1, 1, 4.5, 'Great introduction to Python programming.'),
    (1, 2, 5.0, 'Very detailed and easy to follow.'),
    (2, 3, 3.8, 'Good course but could use more examples.'),
    (3, 4, 4.9, 'Fantastic and highly informative.'),
(4, 5, 4.2, 'Well-structured with useful insights.'),
(5, 6, 3.5, 'Average course, needs improvement.'),
(6, 7, 5.0, 'Excellent content and practical examples.'),
(7, 8, 4.7, 'Great start for AI beginners.');
```

6. Backend Implementation (Java Spring Boot)

6.2 Overview:

The backend is built with Spring Boot and follows the RESTful API prin- ciples. It communicates with the PostgreSQL database using Spring Data JPA and Flyway.

1. Main Components

- **Database**: The backend interacts with a relational database to store and retrieve data. Key entities include Course, Student, Instructor, Enrollment, Feedback, and Category.
- **APIs**: The backend exposes RESTful APIs for client interactions, allowing frontend components to access course listings, enrollments, feedback, and user profiles.
- **Error Handling**: Robust error handling is implemented to manage issues such as missing data, incorrect input formats, and connection failures.

2. Key Functionalities

1. **Course Management**:
    - **Endpoints**:
        - /course/get-course-by-name/{courseName}: This endpoint retrieves a list of courses by their name. It returns a ResponseEntity with a Response object containing the course data or an error message.
        - /course/get-all-courses: This endpoint retrieves all courses from the database. It returns a ResponseEntity with a Response object containing a list of all courses or an error message.
        - /course/get-course-by-category-name/{categoryName}: This endpoint retrieves courses associated with a specific category name.
        - /course/get-course-by-instructor-name/{instructorName}: This endpoint retrieves courses offered by a specific instructor.
        - /course/get-course-by-student-name/{studentName}: This endpoint retrieves courses associated with a specific student.
        - /course/create-course: This endpoint creates a new course based on the provided data in a CourseDtoRequest object.
        - /course/update-course: This endpoint updates an existing course.
        - /course/sort-by-duration: This endpoint retrieves and sorts all courses by their duration.
        - /course/sort-by-price: This endpoint retrieves and sorts all courses by their price.
        - /course/sort-by-enrollments: This endpoint retrieves and sorts all courses by the number of enrollments.
        - /course/sort-by-rating: This endpoint retrieves and sorts all courses by their rating.
2. **Student Management**:
    - **Endpoints**:
        - /student/get-student-by-name/{studentName}: Retrieves a list of students based on their name.
        - /student/get-student-by-course-id/{courseId}: Retrieves students by course ID.
        - /student/get-all-student: Retrieves all students from the database.
        - /student/create-student: Creates a new student using the provided StudentDto.

        - /student/update-student: Updates an existing student using the provided StudentDto.
        - /student/sort-by-name: Retrieves and sorts students by their name.
3. **Instructor Management**:
    - **Endpoints**:
        - /instructor/get-instructor-by-name/{instructorName}: Retrieves a list of instructors based on their name.
        - /instructor/get-all-instructors: Retrieves all instructors from the database.
        - /instructor/create-instructor: Creates a new instructor using the provided InstructorDto.
        - /instructor/update-instructor: Updates an existing instructor using the provided InstructorDto.
        - /instructor/sort-by-name: Retrieves and sorts instructors by their name.
4. **Enrollment Management**:
    - **Endpoints**:
        - /enrollment/get-enrollment-by-course-id/{courseId}: This endpoint retrieves a list of enrollments associated with a specific course ID.
        - /enrollment/get-enrollment-by-student-id/{studentId}: This endpoint retrieves a list of enrollments for a specific student ID.
        - /enrollment/get-all-enrollment: This endpoint retrieves all enrollments from the database.
        - /enrollment/create-enrollment: This endpoint creates a new enrollment based on the provided data in an EnrollmentDto object.
        - /enrollment/delete-enrollment/{courseId}/{studentName}: This endpoint deletes an enrollment based on the course ID and student name.
        - /enrollment/sort-by-date: This endpoint sorts enrollments by their enrollment date.
        - /enrollment/sort-by-completion-status: This endpoint sorts enrollments by completion status.
5. **Feedback System**:
    - **Endpoints**:
        - /feedback/get-feedback-by-course-name/{courseName}: This endpoint retrieves a list of feedbacks associated with a specific course name.
        - /feedback/get-feedback-by-student-name/{studentName}: This endpoint retrieves a list of feedbacks for a specific student name.
        - /feedback/get-all-feedback: This endpoint retrieves all feedbacks from the database.
        - /feedback/create-feedback: This endpoint creates a new feedback based on the provided data in a FeedbackDto object.
        - /feedback/update-feedback: This endpoint updates an existing feedback based on the provided FeedbackDto data.
        - /feedback/delete-feedback/{feedbackId}: This endpoint deletes a feedback based on the provided feedback ID.
        - /feedback/sort-by-rating: This endpoint sorts feedbacks by their rating.
        - /feedback/sort-by-date: This endpoint sorts feedbacks by the feedback date.
6. **Category Management**:
    - **Endpoints**:
        - /category/create-category This endpoint creates a new category. It expects a CategoryDto object in the request body and returns a ResponseEntity with a Response object containing the created category data or an error message.

        - /category/get-category-by-name/{categoryName}: This endpoint retrieves a category by its name. It returns a ResponseEntity with a Response object containing the category data or an error message.
        - /category/get-all-category: This endpoint retrieves all categories from the database. It returns a ResponseEntity with a Response object containing a list of all categories or an error message.
        - /category/update-category: This endpoint updates an existing category. It expects a CategoryDto object in the request body and returns a ResponseEntity with a Response object containing the updated category data or an error message./category/delete/{id}: Removes a category from the system.
        - /category/sort-by-name: This endpoint sorts all categories by their name.

3. Error Handling

- **Failed Requests**:
    - **404 Not Found**: Returned when a resource does not exist.
    - **400 Bad Request**: Returned for invalid input or missing required fields.
    - **500 Internal Server Error**: Returned for server-side issues; users are advised to contact support if this persists.

7. How to run our project

Prerequisites

1. **Download PostgreSQL 16**:
   - Visit the PostgreSQL download page.
   - Select the appropriate installer for your operating system (Windows, macOS, or Linux).
   - Install PostgreSQL 16 by following the on-screen instructions.
2. **Set Up PostgreSQL Database**:
   - **Open SQL PowerShell** or your preferred PostgreSQL client (like pgAdmin).
   - **Create a new database** named brain_rush:

     ```
     CREATE DATABASE brain_rush;
     ```

Running the Brain Rush Project with Flyway

1. **Open the Project**:
   - Open your favorite Integrated Development Environment (IDE) like IntelliJ IDEA, Eclipse, or Visual Studio Code.
   - Import or open the Brain Rush project in your IDE.
2. **Configure Database Connection**:
   - Locate the application.properties file (usually under src/main/resources).
   - Modify the database connection properties to match your PostgreSQL setup:

     properties
     Копировать код
     ```
     spring.datasource.url=jdbc:postgresql://localhost:5432/brain_rush
     spring.datasource.username=your_username
     spring.datasource.password=your_password
     spring.datasource.driver-class-name=org.postgresql.Driver
     flyway.locations=classpath:/db/migration
     ```

3. This configuration tells Flyway to look for migration scripts in the src/main/resources/db/migration directory.
4. **Run the Project**:
   - In your IDE, locate the main class of the Brain Rush project.
   - Run the project as a Java application.
   - The server should start. During startup, Flyway will automatically run any pending database migrations based on the scripts found in the db/migration folder.
5. **Verify Migrations**:
   - You can verify the migrations have been applied by checking the Flyway status:

     ```
     ./mvnw flyway:info
     ```

   - If you want to apply migrations manually or check the applied migrations:

     ```
     ./mvnw flyway:migrate
     ```

6. **Testing**:
   - Go to this link http://localhost:8888/course

7. **Handling Issues**:
   - If you encounter any issues with the database connection, ensure that your PostgreSQL server is running and that your user has the appropriate permissions.
   - Verify that the database brain_rush exists and that the connection details in application.properties are correct.
   - Check the logs for any Flyway-related errors to diagnose issues with migration scripts or database access.
8. Maintanse project

   Database Maintenance

   1. **Regular Backups**:
      - Use pg_dump to back up your database regularly (daily recommended).
        pg_dump -U your_username -d brain_rush -F c -b -v -f /path/to/backup/brain_rush_backup.sql

   2. **Optimize Database Performance**:
      - Regularly run VACUUM FULL; and ANALYZE; in SQL.
      - Set PostgreSQL's auto_vacuum setting to on:
        ALTER DATABASE brain_rush SET auto_vacuum = on;

   3. **Monitor Database Health**:
      - Check disk space and review logs for errors.
      - Use:
        df -h
        tail -f /var/log/postgresql/postgresql-16-main.log

   4. **Flyway Migrations**:
      - Check for pending migrations:
        ./mvnw flyway:info

      - Apply migrations:
        ./mvnw flyway:migrate

   Application Maintenance

   1. **Update Dependencies**:
      - Check for updates:
        ./mvnw versions:display-dependency-updates

      - Update dependencies:
        ./mvnw versions:use-latest-releases

   2. **Log Monitoring**:
      - Use logback for structured logging:
        tail -f /path/to/logs/application.log

   3. **Health Checks**:
      - Expose a health check endpoint:
        GET http://localhost:8080/actuator/health

4. **Regular Restarts**:
    - Periodically restart the application server:
      ./mvnw spring-boot:stop
      ./mvnw spring-boot:start

5. **Backup Configuration Files**:
    - Regularly back up application configuration files:
      cp /path/to/config /path/to/backup/

By following these routine maintenance tasks, the Brain Rush project will run efficiently and remain secure.

9. Conclusion

The Brain Rush project aims to provide a modern, scalable, and secure e-learning management system. With the use of Java Spring Boot, PostgreSQL, and Thymeleaf, the system offers essential functionalities such as course management, student enrollment, feedback collection, and advanced analytics. The modular architecture and robust security ensure smooth and efficient operation for users, instructors, and administrators alike.