DOCUMENTATATION

DBD LABORATORY: STUDENT MANAGEMENT APPLICATION

STUDENT NAME: PELLE ANDREI
GROUP: 30441

TABLE OF CONTENTS

1. Objectives

1. Choose your own domain and correspondingly build your own NoSQL database in the MongoDB environment. Define projections, filters, and aggregation pipelines. Display the corresponding NoSQL queries.

2. Build a Graphical User Interface (GUI), preferably web interface, for your NoSQL database, using HTML and PHP (recommended). Connect to your MongoDB database, build an authentication form, and also adequate interfaces in order to perform the CRUD operations.

2. Problem analysis, modelling

The Student Management Application is a full-stack web application developed using Spring Boot for the backend, React for the frontend, and MongoDB as the NoSQL database. It provides a platform for managing students, courses, and user accounts.

Backend Development

Spring Boot

Spring Boot is used for rapid backend development, simplifying the process of setting up and configuring Spring applications. It provides defaults and conventions while allowing customization.

MongoDB Integration

MongoDB, a NoSQL database, is integrated with Spring Boot. MongoDB offers flexibility in data modeling, which is particularly useful for hierarchical data, variable formats, and rapid development where the schema may evolve over time.

MongoDB Entities

1. **Student**: Represents the students in the system.
    - **id**: Unique identifier for each student.
    - **personalDetails**: Embeds personal details like name, email, date of birth, and enrollment status.
    - **enrollments**: A list of **Enrollment** objects representing the courses in which the student is enrolled.
2. **Course**: Represents courses offered.
    - **id**: Unique identifier for each course.
    - **courseCode**: A unique code for the course.
    - **courseName**: Name of the course.
    - **courseDescription**: A brief description of the course.
    - **credits**: Number of credits for the course.
    - **prerequisites**: List of course IDs that are prerequisites.
3. **User**: Manages user accounts for authentication.
    - **id**: Unique identifier for each user.
    - **username**: Unique username for user login. Indexed to enforce uniqueness.
    - **name**: Full name of the user.
    - **password**: Encrypted password for the user.
    - **role**: Role of the user (e.g., admin, student).

Spring Data MongoDB

Spring Data MongoDB is used for integrating MongoDB with Spring. It provides a way to perform CRUD operations on MongoDB documents using simple Java methods.

- **Repositories**: For each entity, a repository interface is defined extending **MongoRepository**, providing methods like **findAll()**, **findById()**, **save()**, and **deleteById()**.

REST API

The application exposes a RESTful API, with endpoints to manage students, courses, and users. Spring Web's **@RestController** is used for creating these endpoints.

Frontend Development

React

React is used for building the frontend, offering a responsive user interface to interact with the backend.

Components

1. **Students**: Displays student data and forms for adding or editing students.
2. **Courses**: Manages course information.
3. **Users**: Handles user authentication and management.

State Management

State management is handled using React's **useState** and **useEffect** for fetching data from the backend and updating the UI accordingly.

Database Development

MongoDB

MongoDB is chosen for its flexibility in handling unstructured data and its ability to scale.

Database Design

- **Collections**: Separate collections for students, courses, and users.
- **Documents**: Each record in the collection is stored as a BSON document.

Indexes

- Unique indexes (e.g., on **username** in **users**) ensure data integrity and improve query performance.

Queries

MongoDB queries are used for CRUD operations, typically executed through Spring Data MongoDB. Aggregation queries are also utilized for more complex operations like counting students by enrollment status.

// CRUD students

Find all students:

```
db.students.find({})
```

Find student by id:

```
db.students.find({ "_id": id })
```

```
db.students.insert({
   "_id": student.id,
   "personalDetails": {
      "firstName": student.personalDetails.firstName,
      "lastName": student.personalDetails.lastName,
      "email": student.personalDetails.email,
      "dateOfBirth": student.personalDetails.dateOfBirth,
      "enrollmentStatus": student.personalDetails.enrollmentStatus
   },
   "enrollments": student.enrollments.map(enrollment => ({
      "courseId": enrollment.courseId,
      "enrollmentDate": enrollment.enrollmentDate,
      "status": enrollment.status,
      "grade": enrollment.grade
   }))
})
```

```
db.students.updateOne(
   { "_id": id },
   {
      $set: {
         "personalDetails.firstName": studentDetails.personalDetails.firstName,
         "personalDetails.lastName": studentDetails.personalDetails.lastName,
         "personalDetails.email": studentDetails.personalDetails.email,
         "personalDetails.dateOfBirth": studentDetails.personalDetails.dateOfBirth,
         "personalDetails.enrollmentStatus": studentDetails.personalDetails.enrollmentStatus,
         "enrollments": studentDetails.enrollments.map(enrollment => ({
            "courseId": enrollment.courseId,
            "enrollmentDate": enrollment.enrollmentDate,
            "status": enrollment.status,
            "grade": enrollment.grade
         }))
      }
   }
)

db.students.deleteOne({ "_id": id })
```

Aggregation to count students by enrollement status:

```
db.students.aggregate([
  { $group: {
     _id: "$personalDetails.enrollmentStatus",
    count: { $sum: 1 }
  }}
])
```

Find only part of student (projection):

```
db.students.find({}, {
   "personalDetails.firstName": 1,
   "personalDetails.lastName": 1,
   "personalDetails.email": 1,
   "_id": 0
})
```

Find all enrolled students:

```
db.students.find({
   "personalDetails.enrollmentStatus": "ENROLLED"
})
```

// CRUD Courses

Get All Courses (getAllCourses()):

```
db.courses.find({})
```

This query retrieves all documents from the courses collection.

Get Course By ID (getCourseById(String id)):

```
db.courses.findOne({ "_id": id })
```

This query finds a single document in the courses collection where the _id field matches the specified id.

Create Course (createCourse(Course course)):

```
db.courses.insert({
   "_id": course.id, // Auto-generated if not provided
   "courseCode": course.courseCode,
   "courseName": course.courseName,
   "courseDescription": course.courseDescription,
   "credits": course.credits,
   "prerequisites": course.prerequisites // Array of course IDs
})
```

This query inserts a new document into the courses collection with the provided course details.

Update Course (updateCourse(String id, Course courseDetails)):

```
db.courses.updateOne(
   { "_id": id },
   {
      $set: {
         "courseCode": courseDetails.courseCode,
         "courseName": courseDetails.courseName,
         "courseDescription": courseDetails.courseDescription,
         "credits": courseDetails.credits,
            "prerequisites": courseDetails.prerequisites
        }
    }
)
```

This query updates the document in the courses collection where the

_id matches the specified id. It sets the new values for courseCode, courseName, courseDescription, credits, and prerequisites.

Delete Course (deleteCourse(String id)):

```
db.courses.deleteOne({ "_id": id })
```

This query removes the document from the courses collection where the _id matches the specified id.

// CRUD Users

Get All Users (getAllUsers()):

```
db.users.find({})
```

This query fetches all documents from the users collection.

Get User By ID (getUserById(String id)):

```
db.users.findOne({ "_id": ObjectId(id) })
```

Retrieves a single user document where the _id matches the specified id.

Create User (createUser(User user)):

```
db.users.insert({
   "_id": user.id, // Auto-generated if not provided
   "username": user.username,
   "name": user.name,
   "password": user.password,
   "role": user.role
})
```

Inserts a new user document into the users collection. MongoDB will automatically handle unique index constraint for the username.

Update User (updateUser(String id, User userDetails)):

```
db.users.updateOne(
   { "_id": ObjectId(id) },
   {
      $set: {
         "username": userDetails.username,
         "name": userDetails.name,
         "password": userDetails.password,
         "role": userDetails.role
      }
   }
)
```

Updates the user document in the users collection where the _id matches the specified id with the new details.

Handles unique index constraint for the username.

Delete User (deleteUser(String id)):

db.users.deleteOne({ "_id": ObjectId(id) })

Removes the user document from the users collection where the _id matches the specified id.

db.users.createIndex({ "username": 1 }, { unique: true })

db.createCollection("users")

db.createCollection("students")

db.createCollection("courses")

Security and Authentication

- User authentication is implemented, likely using Spring Security for securing endpoints.
- Passwords are stored securely, preferably encrypted in the database.

Deployment and Maintenance

- The application can be containerized using Docker for ease of deployment.
- Regular database backups and maintenance activities should be scheduled.

Challenges and Considerations

- **Schema Design**: While MongoDB is schema-less, thoughtful design is crucial for performance and maintainability.
- **Data Consistency**: MongoDB's eventual consistency model might be a consideration in certain use cases.
- **Scalability**: While MongoDB scales well, proper indexing and query optimization are key.
- **Enrollment Status**: Automatically updates enrollment status to 'DROPPED' when a student's status changes to 'ON_LEAVE', 'WITHDRAWN', or 'SUSPENDED'.
- **Log Course Updates**: Captures and logs changes made to courses, facilitating effective change tracking.

3. Conclusion

The Student Management Application demonstrates a modern approach to web application development using a Spring Boot backend, a React frontend, and MongoDB. This stack offers flexibility, ease of development, and scalability, making it suitable for a wide range of applications.