Marvellous Portal

---

1. Objective

To demonstrate a REST API using Spring Boot for managing "batch entries". The application provides standard CRUD (Create, Read, Update, Delete) operations for batch details.

---

2. Technology Stack * Backend: Spring Boot

- **Database**: MongoDB
- **Language**: Java
- **Build Tool**: Maven
- **Libraries**:
    - spring-boot-starter-web
    - spring-boot-starter-data-mongodb
    - lombok

---

3. Folder Structure & Components The project follows a standard Spring Boot layered architecture:

- **MarvellousPortal\src\main\java\com\marvellous\MarvellousPortal (Base Package)**
    - **MarvellousPortalApplication.java**: The main entry point of the application.
    - **controller**:
        - **BatchEntryController.java**: Handles all REST API requests for batch entries.

        - **HealthCheck.java**: A simple controller for checking the application's health.
    - **Entity**:
        - **BatchEntry.java**: The data model for a batch entry, mapped to the "BatchDetails" MongoDB collection.
    - **Repository**:
        - **BatchEntryRepository.java**: The data access layer for BatchEntry.
    - **Service**:
        - **BatchEntryService.java**: Contains the business logic for batch entry operations.
- **resources**:
    - **application.properties**: Configuration file for the application, including database connection details.
- **pom.xml**: Maven configuration for dependencies and project build.

---

4. Dependencies & Application Properties The pom.xml file includes dependencies for spring-boot-starter-web, spring-boot-starter-data-mongodb, and lombok.

The application.properties file configures the MongoDB connection:

- spring.application.name=MarvellousPortal
- spring.data.mongodb.host = localhost
- spring.data.mongodb.port = 27017
- spring.data.mongodb.database = MarvellousFullStack

---

5. Running the Application 1. Ensure you have a running MongoDB instance on localhost:27017 with a database named MarvellousFullStack.

2. Use Maven to build the project.
3. Run the MarvellousPortalApplication.java file. The application will start on its default port, usually 8080.

---

6. API Endpoint Documentation The base URL for all batch-related endpoints is /batches.

HTTP Method	Endpoint	Description
GET	/HealthCheck	Checks if the application is running.
GET	/batches	Retrieves all batch entries.
POST	/batches	Creates a new batch entry.
GET	/batches/id/{myid}	Retrieves a single batch entry by its ObjectId.
PUT	/batches/id/{myid}	Updates an existing batch entry identified by its ObjectId.
DELETE	/batches/id/{myid}	Deletes a batch entry by its ObjectId.

---

7. Request & Response JSON Examples #### POST /batches (Create) Request Body:

JSON

```
{
  "name": "Full Stack Development",
  "fees": 15000
}
```

Successful Response (201 Created):

JSON

```
{
  "id": "60c72b2f9b8d2a0e2c24c7f0",
  "name": "Full Stack Development",
  "fees": 15000
}
```

GET /batches/id/{myid} (Read) Successful Response (200 OK):

JSON

```
{
  "id": "60c72b2f9b8d2a0e2c24c7f0",
  "name": "Full Stack Development",
  "fees": 15000
}
```

Not Found Response (404 Not Found): No body is returned.

PUT /batches/id/{myid} (Update) Request Body:

JSON

```
{
  "name": "Web Development",
  "fees": 12000
}
```

Successful Response (200 OK):

JSON

```
{
  "id": "60c72b2f9b8d2a0e2c24c7f0",
  "name": "Web Development",
  "fees": 12000
}
```

---

8. Error Handling Format * HTTP 404 NOT FOUND: Returned when a requested resource (e.g., a batch entry by ID) is not found. This status is also used when the list of all batches is empty.

- **HTTP 400 BAD REQUEST**: Returned for a POST request if there is an error while saving the batch entry.
- **HTTP 204 NO CONTENT**: Returned for a successful DELETE request, with an empty response body.