Traini8 - Training Center Registry API Documentation

Overview:

This Spring Boot project provides RESTful APIs to manage Government-funded training centers, allowing the creation and retrieval of training center details. The APIs support filtering capabilities for more efficient data retrieval.

Project Structure:

- **Model:** Defines the data structure (TrainingCenter, Address) and includes validation constraints.
- **Repository:** Extends Spring Data JPA for database interactions.
- **Service:** Handles business logic for saving and filtering training centers.
- **Controller:** Exposes the APIs for creating and retrieving training centers.
- **ExceptionHandler**: Catches and formats validation error messages.
- **Database**: PostgreSQL

API Endpoints:

1. POST /api/training-centers - Create Training Center

Description:

This API creates and saves a new training center in the database. The input JSON must conform to the specified schema, with various fields requiring validation. The createdOn field is automatically populated by the server based on the current epoch timestamp.

Request:

- Method: POST
- URL: /api/training-centers
- Content-Type: application/json

Request Body Schema:

```
json
{
"centerName": "Tech Training Center",
"centerCode": "ABC123XYZ456",
"address": {
"detailedAddress": "123 Street",
"city": "New York",
"state": "NY",
"pincode": "100001"
},
"studentCapacity": 500,
```

```
"coursesOffered": ["Java", "Spring Boot", "Microservices"],
"contactEmail": "center@example.com",
"contactPhone": "1234567890"
}
```

Field Descriptions:

Field	Type	Description	Validation
centerName	*String*	*Name of the training center*	*Required, max length 40 characters*
centerCode	*String*	*Unique alphanumeric center code (12 characters)*	*Required, exactly 12 alphanumeric characters*
address	*Object*	*Contains detailed address, city, state, and pincode*	*Required fields inside Address (detailedAddress, city, state, pincode)*
studentCapacity	*Integer*	*Maximum number of students the center can accommodate*	*Optional, must be positive*
coursesOffered	*List of Strings*	*Courses offered by the training center*	*Optional*
contactEmail	*String*	*Email address of the center (if provided)*	*Optional, must be a valid email format*
contactPhone	*String*	*| Contact phone number (10-digit number)*	*Required, must be a valid 10-digit phone number*
createdOn	*Long*	*Server-generated epoch timestamp when the center is created*	*Auto-generated by the server based on the current epoch time*

Response:

- Status: 201 Created
- Body: JSON representation of the saved training center with all details, including the createdOn timestamp generated by the server.
- Example Response:

```
json
{
"centerName": "Tech Training Center",
"centerCode": "ABC123XYZ456",
"address": {
"detailedAddress": "123 Street",
"city": "New York",
"state": "NY",
```

```
"pincode": "100001"
},
"studentCapacity": 500,
"coursesOffered": ["Java", "Spring Boot", "Microservices"],
"contactEmail": "center@example.com",
"contactPhone": "1234567890"
}
```

Error Handling:

- Status: 400 Bad Request
- Response Body: A JSON object showing the validation error message.
- Example Error Response:

```
json
{
"centerCode": "CenterCode must be exactly 12 alphanumeric characters",
"contactPhone": "ContactPhone must be a valid 10-digit number"
}
```

2. GET /api/training-centers - Get All Training Centers (with Optional Filters)

Description:

This API retrieves a list of all saved training centers. It supports filtering based on various optional query parameters. If no filters are provided, it will return all centers.

Request:

- Method: GET
- URL: /api/training-centers
- Content-Type: application/json

Optional Query Parameters:

Parameter	Type	Description
city	*String*	*Filter by city*
state	*String*	*Filter by state*
capacity	*Integer*	*Filter by minimum student capacity*
course	*String*	*Filter by a specific course offered*

Example Requests:

1. Get all training centers:

GET /api/training-centers

2. Get centers filtered by city and student capacity:

GET /api/training-centers?city=NewYork&capacity=100

3. Get centers filtered by course:

GET /api/training-centers?course=Java

Response:

- Status: 200 OK
- Body: A JSON array containing all matching training centers.
- *Example Response*:

```
json
[
{
"id": 1,
"centerName": "Tech Training Center",
"centerCode": "ABC123XYZ456",
"address": {
"detailedAddress": "123 Street",
"city": "New York",
"state": "NY",
"pincode": "100001"
},
"studentCapacity": 500,
"coursesOffered": ["Java", "Spring Boot", "Microservices"],
"createdOn": 1691234567,
"contactEmail": "center@example.com",
"contactPhone": "1234567890"
}
]
```

- Empty Result: If no centers are found based on the filters, the response will be an empty list.

```
json
[]
```

Validation Rules:

1. Center Name*: Max 40 characters

2. Center Code*: Exactly 12 alphanumeric characters

3. Address: Detailed address, city, state, and pincode are mandatory; Pincode should be a valid 6-digit number

4. Student Capacity (Optional): Should be a positive integer

5. Courses Offered (Optional): Accepts a list of course names

6. Contact Email (Optional): Should be a valid email format if provided

7. Contact Phone*: Should be a valid 10-digit phone number

8. Created On: Auto-generated by the server

Error Handling:

The project uses a *Global Exception Handler* to catch validation errors and return meaningful messages. Errors are returned with a 400 Bad Request status code and include details about the specific fields that failed validation.

Example Error Response:

```
json
{
"centerName": "CenterName is mandatory",
"centerCode": "CenterCode must be exactly 12 alphanumeric characters",
"contactPhone": "ContactPhone must be a valid 10-digit number"
}
```

Database Configuration:

The project uses PostgreSQL database. PostgreSQL Configuration (application.properties):

```
spring.datasource.url=jdbc:postgresql://localhost:5432/traini8db
spring.datasource.username=traini8user
spring.datasource.password=yourpassword
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
```

Testing APIs:

You can use tools like *Postman* or *cURL* to test the APIs.

1. POST Example (cURL):

```
bash
curl -X POST http://localhost:8080/api/training-centers \
   -H "Content-Type: application/json" \
   -d '{
"centerName": "Tech Training Center",
```

```
"centerCode": "ABC123XYZ456",
"address": {
"detailedAddress": "123 Street",
"city": "New York",
"state": "NY",
"pincode": "100001"
},
"studentCapacity": 500,
"coursesOffered": ["Java", "Spring Boot"],
"contactEmail": "center@example.com",
"contactPhone": "1234567890"
}'
```

2. GET Example (cURL):

You can use cURL to retrieve all training centers or filter them based on the provided query parameters.

A. Get All Training Centers:

```
bash
curl -X GET http://localhost:8080/api/training-centers
```

B. Get Training Centers with Filters:

a. Filter by city and student capacity:

```
bash
curl -X GET http://localhost:8080/api/training-centers?city=New%20York&capacity=300
```

b. Filter by course offered:

```
bash
curl -X GET http://localhost:8080/api/training-centers?course=Java
```

c. Filter by multiple criteria:

```
bash
curl -X GET http://localhost:8080/api/training-centers?city=New%20York&state=NY&course=Spring%20Boot
```

Project Setup Instructions:

Step 1: Clone the Project

bash

git clone <your-repo-url>

cd traini8

Step 2: Configure PostgreSQL Database

Make sure PostgreSQL is installed and running on your machine. Create a new database and configure the application.properties file with the correct database credentials.

PostgreSQL Database Setup:

1) Open your PostgreSQL command line or GUI (e.g., pgAdmin) and create a new database:

   sql

   a) CREATE DATABASE traini8db;
2) Create a new PostgreSQL user and grant privileges:

   sql

   a) CREATE USER traini8user WITH PASSWORD 'yourpassword';
   b) GRANT ALL PRIVILEGES ON DATABASE traini8db TO traini8user;
3) Update the database configurations in the src/main/resources/application.properties file:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/traini8db
spring.datasource.username=traini8user
spring.datasource.password=yourpassword
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
```

Step 3: Build the Project

You can use *Maven* to build the project. If you're using IntelliJ IDEA or Eclipse, you can also import the project directly into the IDE and build it from there.

bash

mvn clean install

Step 4: Run the Application

Once the project is built successfully, you can run the Spring Boot application:

bash

mvn spring-boot:run

Step 5: Access the APIs

The application will start on port *8080* by default. You can then access the APIs via:

- ***POST API*: http://localhost:8080/api/training-centers**
- ***GET API*: http://localhost:8080/api/training-centers**

Step 6: Verify the PostgreSQL Database

After running the application and using the APIs, you can verify that the data is being stored in your PostgreSQL database by checking the traini8db database using your PostgreSQL client.

Further Enhancements:

1. *Additional Filters*:
    - You can easily extend the GET API by adding more filters such as filtering by pincode or a range for studentCapacity.
2. *Pagination and Sorting*:
    - Add pagination support to the GET API to improve performance when retrieving large datasets. Spring Data JPA supports pagination through the Pageable interface.
    - Sorting can be added using query parameters to allow sorting by fields such as centerName, createdOn, etc.
3. *Security*:
    - Implement authentication and authorization using Spring Security to secure the API endpoints. You can choose to secure endpoints with roles such as admin and user.
4. *Dockerization*:
    - You can create a Dockerfile and docker-compose configuration to containerize the application along with the database (e.g., MySQL or PostgreSQL).

Conclusion:

This project provides a basic yet functional Minimum Viable Product (MVP) for a Government-funded training center registry. The two core functionalities include:

- The ability to create a training center with field validation.
- The ability to retrieve a list of training centers with filtering capabilities.

The system is designed to be extendable with more features like pagination, security, and database enhancements. The use of standard practices like Spring Data JPA, Exception Handling, and validation annotations ensures the codebase is maintainable and scalable.