REST API in Spring Boot

REST API :

REST stands for Representational State Transfer.
A REST API allows different systems to communicate with each other using HTTP protocol.

In Spring Boot, we can build REST APIs to handle operations like:

- Get data (GET)
- Save data (POST)
- Update data (PUT)
- Delete data (DELETE)

Key Annotations Used

Annotation	Purpose
`@RestController`	Marks a class as a controller where every method returns a JSON or plain response
`@RequestMapping`	Maps HTTP requests to handler methods
`@PostMapping`	Handles HTTP POST requests
`@GetMapping`	Handles HTTP GET requests
`@RequestBody`	Binds JSON body of request to a Java object
`@Autowired`	Enables dependency injection

Please consider below code

1. BatchEntry.java – Data Model

This class defines the structure of data we are working with.

```
public class BatchEntry
{
private  long id;

private String name;

private int fees;


   // Getters and Setters
}
```

This is called a POJO (Plain Old Java Object). It holds the data passed via API.

2. BatchEntryController.java – REST Controller

This class handles API endpoints using Spring Boot annotations.

```
@RestController
@RequestMapping("/batches")
public class BatchEntryController
{
    private Map<Long, BatchEntry> batchentries = new HashMap<>();
    // HTTP : GET
    // R : Read
    // select * from batches;
    @GetMapping
    public List<BatchEntry> getAll()
    {
        return new ArrayList<>(batchentries.values());
    }
    // HTTP : POST
    // C : Create
    // insert into batches values(1,'PPA',25000);
    @PostMapping
    public boolean createEntry(@RequestBody BatchEntry myentry)
    {
        batchentries.put(myentry.getId(), myentry);
        return  true;
    }
    // HTTP : GET
    // R : Read
    // select * from batches where id = 1;
    @GetMapping("/id/{myid}")
    public  BatchEntry getBatchEntryById(@PathVariable Long myid)
```

```
    {
        return batchentries.get(myid);
    }
    // HTTP : DELETE
    // D : Delete
    // delete from batches where id = 1;
    @DeleteMapping("/id/{myid}")
    public BatchEntry deleteEntryById(@PathVariable Long myid)
    {
        return batchentries.remove(myid);
    }

    //  HTTP : PUT
    // update batches set fees = 30000 where id = 1;
    @PutMapping("/id/{myid}")
    public BatchEntry updateEntryById(@PathVariable Long myid, BatchEntry myentry)
    {
            return batchentries.put(myid,myentry);
    }
}
```

1. POST – Create/Insert Data

```
@PostMapping
    public boolean createEntry(@RequestBody BatchEntry myentry)
```

Accepts JSON data in the request body.

- Converts it to BatchEntry object.
- Adds it to the list (like saving in memory).
- Use tool like Postman or frontend form to send data.

Example Request:

```
{
 "id" : 2
 "name": "Python ML",
 "fees": "28000"
}
```

2. GET – Read Data

```
@GetMapping
    public List<BatchEntry> getAll()
```

- Returns the list of all batch entries.
- Called from browser or Postman using GET request.

URL: http://localhost:8080/batches

3. PUT – Update Existing Data

```
@PutMapping("/id/{myid}")
    public BatchEntry updateEntryById(@PathVariable Long myid, BatchEntry myentry)
```

- Accepts index in the URL (/batches/1) and JSON body.
- Replaces the batch data at the given index.

Request Example:

```
PUT http://localhost:8080/batches/1
Body:
{
 "id" : 1
 "name": "PPA",
  "fees": "30000"
}
```

4. DELETE – Remove Data

```
@DeleteMapping("/id/{myid}")
   public BatchEntry deleteEntryById(@PathVariable Long myid)
```

- Deletes a batch at the specified index in the list.
- Can be tested using Postman or curl.

URL: http://localhost:8080/batches/1

Key Concepts

Term	Meaning
`@RestController`	Tells Spring Boot to return data as JSON (RESTful)
`@RequestBody`	Binds incoming JSON to a Java object
`@PathVariable`	Gets value from URL (like index or ID)
`List<BatchEntry>`	Temporary in-memory storage (ArrayList used as fake database)
`POST`	To insert new data
`GET`	To retrieve data
`PUT`	To update data
`DELETE`	To delete data

3. HealthCheck.java – Test Endpoint

```
@RestController
public class HealthCheck
{
   @GetMapping("Healthcheck")
   public String check()
      {
      return "Everything is OK";
   }
}
```

Useful to verify that your application is running correctly.

Please consider below steps to test the application

1. Test POST – Add New Batch

➤ Purpose: Insert a new batch into the system

- **Method: POST**
- URL: http://localhost:8080/batches
- **Headers:**
  - Content-Type: application/json
- **Body (JSON):**

```
{
      "id" : 3
      "name": "Python Machine Learning",
  "fees": "28000"
}
```

Click "Send"

2. Test GET – Display All Batches

➤ Purpose: View all stored batches

- **Method: GET**
- URL: http://localhost:8080/batches

Click "Send"

3. Test PUT – Update Batch by Index

➤ Purpose: Update batch details at a specific position

- **Method: PUT**
- URL: http://localhost:8080/bacthes/1
- **Headers:**
  - Content-Type: application/json
- **Body (JSON):**

```
{
"id" : 3
```

```
  "batchName": "Python + GenAI",
 "fees": "40000"
}
```

Click "Send"

4. Test DELETE – Remove Batch by Index

➤ Purpose: Delete a batch at a given position

- **Method: DELETE**
- URL: http://localhost:8080/batches/1
  (Deletes the batch at index 1, if it exists)

Click "Send"

Health Check (Optional)

➤ Check if server is running

- **Method: GET**
- URL: http://localhost:8080/Healthcheck

Output:

Everything is OK...