* **DB Structure :**

****

* **Entity :**

****

* **SpringBoot application**

****

* **Controller**

****

* **ResponseDto for student details w/ courses**

****

* **Application architecture :**

*Client -> hit endpoint url -> Server with 3-layer architecture*

**3-layer architecture Server**

Repository

Service layer

Rest Controller

(**StudentRegistrationController**

)

Rest Controller to map endpoint urls to respective service 🡪 service layer to talk to repository layer 🡪 Repository layer to fetch and return from DB 🡪 response from DB flows back through same layers via repository to service to controller and response is sent to client

* **Skeleton code for required scenarios**

1. **Add a new student along w/ its course registrations**

**Rest APIs Endpoints and backend server architecture :**

1. Add Student with all course registration details

* <http://localhost:8080/addStudent>
* Request Method : POST
* Body :

{

“studentId” : “1”,

“courses” :

[

{

“courseName“ : “Java”,

},

{

“courseName“ : “Spring Boot”,

}

]

}

***Implementation*** :

|  |
| --- |
| @PostMapping(value = **"/addStudent"**) **public** Student saveStudent(@RequestBody Student student){  **try**{  System.*out*.println(**"Saving Student details for id "**+student.getStudentId());  **return** studentRepository.save(student);   }**catch** (Exception e){  System.*out*.println(**"Exception while deleting id :"**+student.getStudentId()+e.getMessage());   }  **return null**; } |

1. Delete Student by studentId

* [http://localhost:8080/ deleteStudent/{studentId}](http://localhost:8080/%20deleteStudent/%7bstudentId%7d)

***Implementation*** :

|  |
| --- |
| @DeleteMapping(value = **"/deleteStudent"**) **public void** deleteStudent(@RequestParam Long studentId){  System.*out*.println(**"Deleting Student id for "**+studentId);  **try**{  studentRepository.deleteById(studentId);  System.*out*.println(**"Student Id "**+studentId+**" deleted successfully!"**);  }**catch** (Exception e){  System.*out*.println(**"Exception while deleting id :"**+studentId+e.getMessage());  }  } |

1. Get all students by courseName as i/p, sorted by studentName

***Please note, this solution includes fetching students’ score also for the courses registerd.***

* [http://localhost:8080/ getStudentDetailsByCourseName/{courseName}](http://localhost:8080/%20getStudentDetailsByCourseName/%7bcourseName%7d)

***Implementation*** :

|  |
| --- |
| @GetMapping(value = **"/getStudentDetailsByCourseName/{courseName}"**) **public** List<RegistrationDetailsResponse> getStudentsByCourseName(@PathVariable String courseName){  System.*out*.println(**"Student and course details fetched for "**+courseName);  **return** studentCourseRegistrationService.findAllByCoursesOrderByStudentName(courseName); } |

 

1. Find Students who didn’t register for a given course

 

* **Best Practices :**

1. Use proper annotations to identify different classes as per their purpose :

For e.g : @RestController for the controller exposing rest endpoints

@Service, @Entity, @Component etc

1. Use latest annotations :

e.g @RestController instead of @Controller + @ResponseMapping

or @GetMapping instead of @RequestMapping + method = RequestMethod.GET

1. Use proper exception handling, at proper places
2. HIBERNATE being performance intensive for batch processes, not allowing multiple inserts, complex – hence should be used selectively, for its simplified DB access, productivity and protability.

Application files for reference :

