University of Sri Jayewardenepura

Faculty of Technology

Department of Information Communication Technology

ITS 4243 - Microservices and Cloud Computing

Assignment 01

Name: Athapattu D.S.

Index No: ICT/21/809

Part 1 - Theory

1. **What is Spring Boot and why is it used?**

- **What is Spring Boot?**

Spring Boot is a framework that is based on Java that ease the work of building production ready spring applications. It removes most of the manual configuration that is used in the traditional Spring framework by using Auto-Configuration, Embedded Servers (Tomcat, Jetty), Opinionated Defaults, Starter dependencies.

- **Why is it used?**

Spring Boot is used widely because it provides many valuable features which are:

  - Reduced complex configuration
  - Provides starter dependencies
  - Faster Development
  - Provides Embedded Servers
  - Great for microservices
  - Production ready tools

Spring Boot is popular because it greatly simplifies and speeds up the development of applications. Spring Boot allows developers to concentrate on writing actual business logic by providing sensible defaults and automatic configuration, eliminating the need to manually configure XML files or set up external servers. Additionally, it has an embedded server, so with just a command like java -jar, apps can run on their own, making deployment incredibly easy. Building microservices, where rapid startup and simple scalability are crucial, is made possible by its lightweight design. Additionally, Spring Boot includes a number of production-ready features that aid developers in better monitoring and maintaining applications, including integrated logging, health checks, and performance metrics.

2. **Explain the difference between Spring Framework and Spring Boot.**

Spring is a complete, modular framework that offers many tools for creating scalable enterprise applications, including Dependency Injection, AOP, and POJO-based development. However, its different sub-frameworks require more manual setup and configuration. The development process is made simpler by Spring Boot, which is based on Spring and offers all of its essential features along with auto-configuration, starter dependencies, and an embedded server. REST-based microservices in particular benefit greatly from Spring Boot's emphasis on speed and user-friendliness, which contrasts with Spring's emphasis on providing developers complete control over the development of an application's various layers.

Feature	Spring Framework	Spring Boot
Purpose	A comprehensive framework for Java application development.	A streamlined, subjective method for building Spring applications quickly.
Configuration	Requires a lot of manual configurations (XML or Java-based).	Provides auto-configuration because of that minimal manual setup needed.
Setup Complexity	More complex and time-consuming to initially start a project.	Very simple, easy and fast to start a project due to in built starter dependencies.
Server Setup	Needs external server (Tomcat/Jetty) to be installed and configured separately.	Has built-in embedded servers like (Tomcat/Jetty/Undertow).
Project Structure	Flexible but everything requires to be defined.	Comes with predefined structure and conventions.
Dependencies	Developer must add and manage individual dependencies manually.	Provides starter POMs that bundle dependencies cleanly.

Focus	Gives full control but requires more effort.	Speeds up development through defaults and automation.
Microservices Support	Can build them, but requires more configuration.	Designed with microservices in mind structure, faster and lighter.
Learning Curve	Higher due to manual setup and configuration.	Lower because its simplicity.
Production Features	Must be configured manually using additional tools.	Comes with in-built Actuator for health checks, metrics, logging, etc.

3. What is Inversion of Control (IoC) and Dependency Injection (DI)?

- **Inversion of Control (IoC)**

The idea behind Inversion of Control (IoC) is to transfer the burden of object creation and management from the developer to the framework. Using the new keyword, developers manually create objects in a typical Java application and manage their interactions. It's the other way around with Spring. Object creation, configuration, management, and even destruction are all done automatically by the IoC container. Because the framework, not the developer, manages the application's overall object lifecycle and flow, this method eliminates tight coupling between classes and facilitates testing, maintenance, and scaling.

- **Dependency Injection (DI)**

Spring uses Dependency Injection (DI) to implement IoC. Springs provides (or "injects") dependencies from the outside rather than having a class create them internally. Injecting constructors, setters, or fields can accomplish this. DI makes code cleaner, more reusable, and more testable by taking the burden of creating dependencies off the class itself. While the framework manages how all necessary components are supplied, DI makes sure that each class only concentrates on its primary goal.

4. **What is the purpose of application.properties / application.yml?**

A Spring Boot project's application.properties or application.yml file serves as a central location for managing the application's configuration settings. Database URLs, port numbers, logging levels, and security credentials can be defined externally by developers using Spring Boot, allowing them to configure the application without altering the source code. By enabling profile-specific configurations, these files regulate how the application operates in various environments (development, testing, and production). Through its auto-configuration mechanism, Spring Boot reads these files automatically at startup and applies the specified settings, increasing the project's flexibility, maintainability, and ease of deployment across various environments.

5. **Explain what a REST API is and list HTTP methods used.**

An approach to web service design known as a REST API (Representational State Transfer Application Programming Interface) allows various systems to interact with one another via the internet by utilizing common HTTP protocols. Everything is viewed as a resource in REST, and every resource can be accessed via a different URL. Because REST APIs are stateless, the server does not store client data in between requests, which enhances scalability and performance. REST APIs are the most popular method for creating contemporary web and mobile applications due to their ease of use, adaptability, and efficiency. They usually send and receive data using JSON or XML.

HTTP methods used in REST APIs:

- **GET** – Retrieve or read data from the server.
- **POST** – Create new data or resources on the server.
- **PUT** – Update or completely replace an existing resource.
- **PATCH** – Partially update an existing resource.
- **DELETE** – Remove or delete a resource from the server.

6. **What is Spring Data JPA? What is an Entity and a Repository?**

Spring Data JPA:

The Spring framework includes a module called Spring Data JPA that greatly simplifies working with relational databases. Spring Data JPA offers a straightforward abstraction layer on top of the Java Persistence API (JPA) to eliminate the need to write complex SQL queries or a lot of boilerplate code. Through the use of clear method names and established conventions, it enables developers to communicate with the database. This keeps the code neat and effective while significantly accelerating development and lowering errors.

Entity

A Java class that represents a database table is called an entity. Every object (instance) of the class represents a row, and every variable (field) within the class corresponds to a column in the table. The @Entity annotation is used to identify and manage entities in JPA and Spring Data. They serve as the foundation for the data model of the application.

Repository

An interface for managing and accessing the data kept in entity classes is called a repository. Developers may simplify database operations by using built-in methods like save(), findAll(), delete(), and findById() rather than writing SQL queries. Repositories in Spring Data JPA provide robust database functionality with minimal coding by extending interfaces such as JpaRepository or CrudRepository.

7. **What is the difference between @Component, @Service, @Repository, @Controller,@RestController**?

@Component:

A generic Spring annotation called @Component can be used to identify any Java class as a bean that is managed by Spring. When component scanning is underway, it instructs the Spring

container to automatically identify and generate an instance of the class. This stereotype, which is the most basic, is applied when a class does not fall into a particular layer category.

```
@Component
public class EmailValidator {
   public boolean isValid(String email) {
      return email.contains("@");
   }
}
```

@Service:

Classes that contain business logic are marked with @Service, a specialization of @Component. It is utilized in an application's service layer. In addition to making the code easier to read and understand, using @Service enables Spring to implement extra features like transaction management.

```
@Service
public class OrderService {

   public String placeOrder(String item) {
      return "Order placed for: " + item;
   }
}
```

@Repository:

Another specific type of @Component utilized in the data access layer is @Repository. It shows that dealing with the database is the responsibility of the class. Additionally, it has an integrated exception translation feature that transforms low-level database exceptions into the DataAccessException hierarchy of Spring.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
   User findByEmail(String email);
}
```

@Controller:

In Spring MVC, classes that handle web requests and return views (like HTML pages using Thymeleaf or JSP) are marked with @Controller. It is mostly utilized when creating conventional web apps that return templates rather than JSON.

```
@Controller
public class HomeController {

   @GetMapping("/home")
   public String homePage() {
      return "home"; // returns home.html template
   }
}
```

@RestController:

@Controller and @ResponseBody are combined to form @RestController. It is employed when creating RESTful APIs, in which the methods directly return data in the form of XML or JSON. Without requiring the @ResponseBody annotation, each method inside a @RestController automatically returns the response body.

```
@RestController
public class UserController {

   @GetMapping("/users")
   public List<String> getUsers() {
      return List.of("John", "Alice", "Dini");
   }
}
```

8. What is @Autowired? When should we avoid it?

The Spring framework can automatically add dependencies to a class by using the @Autowired annotation. Spring takes care of the wiring and object creation for you, saving you the trouble of manually creating objects with the new keyword. Spring will automatically search the application context for a matching bean and inject it when you add @Autowired to a constructor, field, or setter method. In addition to supporting loose coupling, this cleans up, maintains, and facilitates testing the code.

```
@Service
public class OrderService {

   private final PaymentService paymentService;

   @Autowired
   public OrderService(PaymentService paymentService) {
      this.paymentService = paymentService;
   }
}
```

When should we avoid @Autowired?

- **Avoid Field Injection**

Avoid field injection since it hides the dependencies of the class and makes testing the code more difficult. There is less flexibility when dependencies are injected straight into fields because it is difficult to swap them out for mocks or stubs during unit testing. Additionally, it results in bad design because, despite depending on Spring to inject dependencies, the class appears to have none. For instance, putting @Autowired directly on a field, like @Autowired private UserRepository userRepository; is something to avoid.

- **Avoid @Autowired When There Is Only One Constructor**

Spring automatically injects all necessary dependencies without requiring the @Autowired annotation when a class has a single constructor. In this instance, using @Autowired is redundant and adds unnecessary verbosity to the code. In addition to improving readability, using implicit constructor injection adheres to current best practices for Spring. The annotation is useless since Spring already knows that the only constructor that is available must be used.

- **Avoid Circular Dependencies**

When two classes are dependent on one another, as in Class A needing Class B and Class B needing Class A simultaneously, this is known as a circular dependency. Because @Autowired is unable to determine which component should be created first in this scenario, it may cause Spring to fail during application startup. A circular dependency error is the outcome of this. Redesigning the architecture to break the cycle is preferable to attempting force injection; this is frequently accomplished by adding an interface, a new service layer, or reorganizing the logic.

9. Explain how Exception Handling works in Spring Boot (@ControllerAdvice).

Using @ControllerAdvice, Spring Boot manages exceptions in a centralized and clean manner, enabling developers to control errors across all controllers from a single location. A class annotated with @ControllerAdvice can intercept any exception thrown in the application, eliminating the need to write repetitive try-catch blocks in each controller. The @ExceptionHandler-marked methods in this class define which exceptions should be handled and what response such as particular HTTP status codes or custom error messages should be returned. When handling common errors like missing resources, invalid input, or unexpected server issues, this method makes the application easier to maintain, improves code organization, and guarantees consistent error responses.

10. What is the role of Maven/Gradle in a Spring Boot project?

Spring Boot projects use Maven and Gradle, build automation and dependency management tools, to manage all the libraries, plugins, and configurations required for the application to function. These tools automatically retrieve the appropriate dependencies from online repositories and

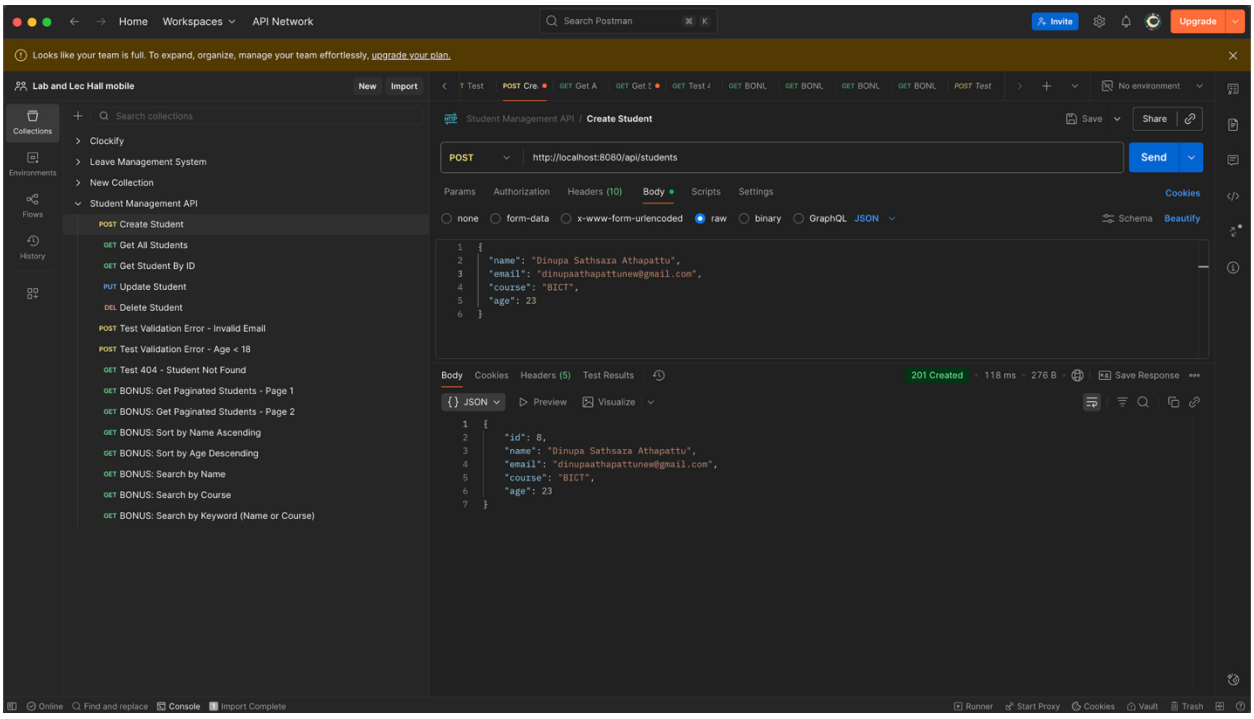
maintain them in order, eliminating the need to manually download jar files. Additionally, they oversee the project's build lifecycle, which includes versioning, code compilation, test execution, and application packaging into an executable JAR. Whereas Gradle uses a more adaptable and quicker Groovy or Kotlin-based script (build.gradle), Maven uses an XML-based configuration (pom.xml). Both tools simplify project setup, guarantee consistent builds, and facilitate the scaling and maintenance of Spring Boot applications.

Part 2 - Practical

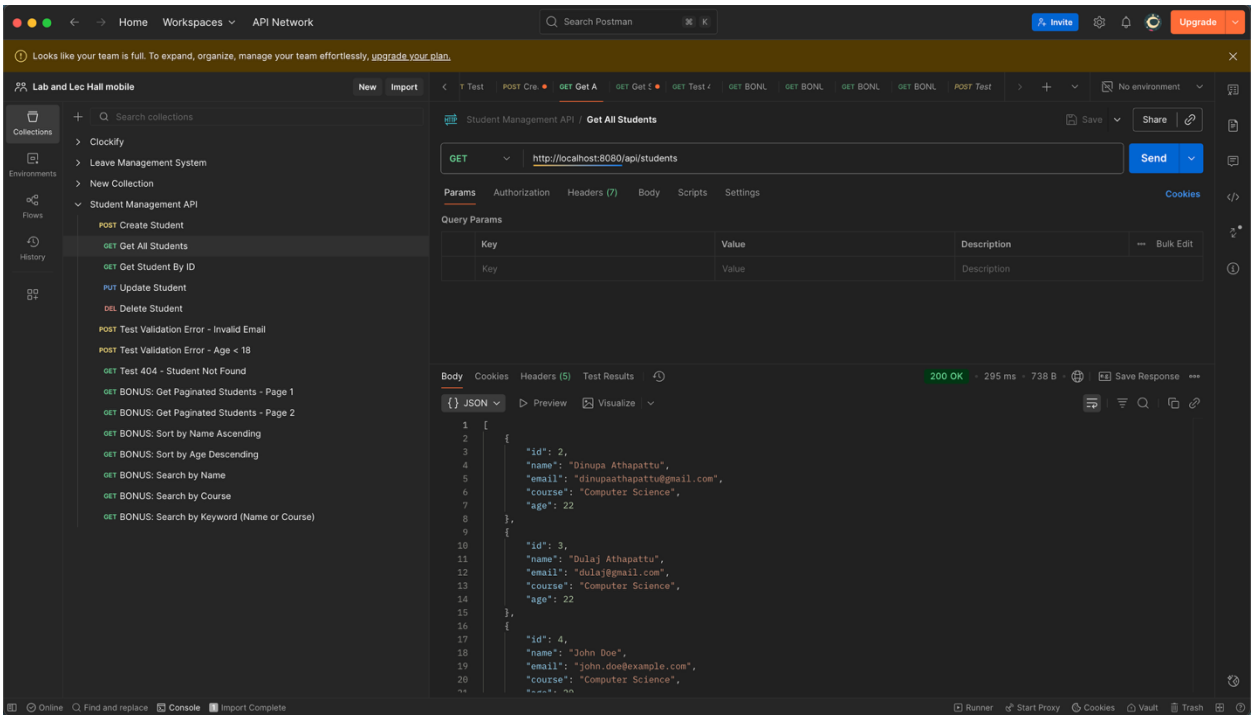
Github repository link: https://github.com/DinupaAthapattu/studentapi-assignment

Postman Testing Screenshots (Also included in the github repo)

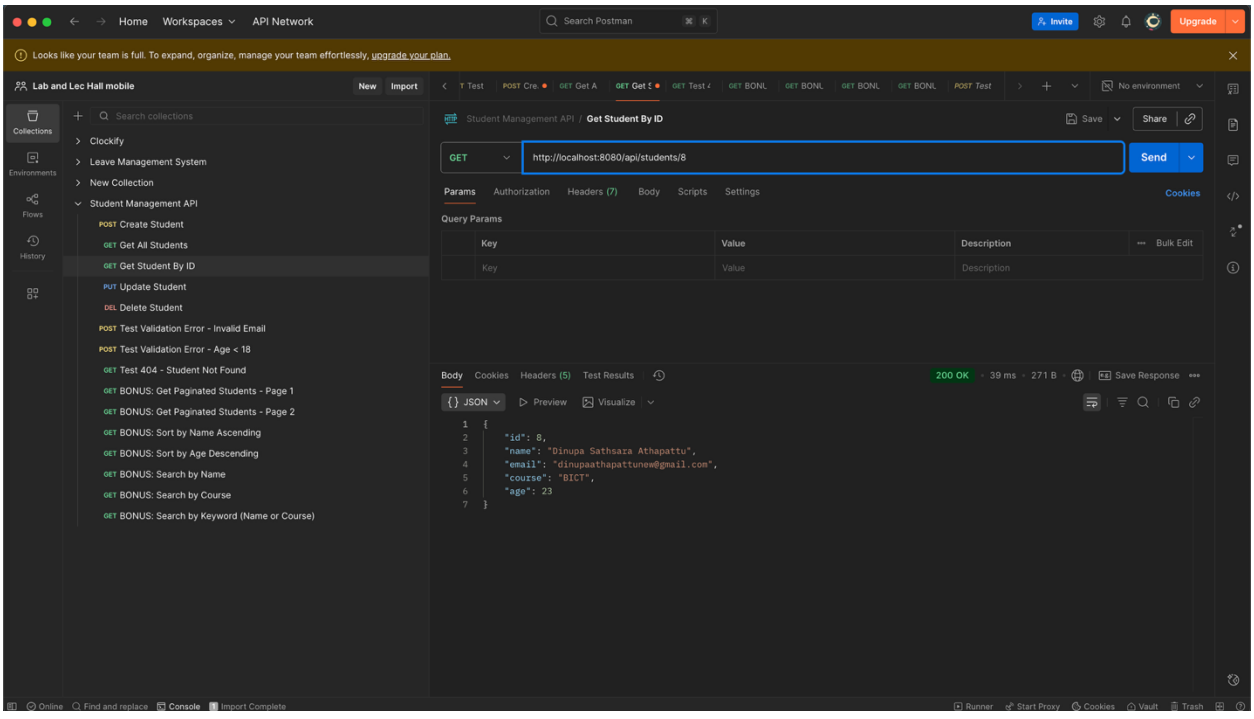
- **Create Student**



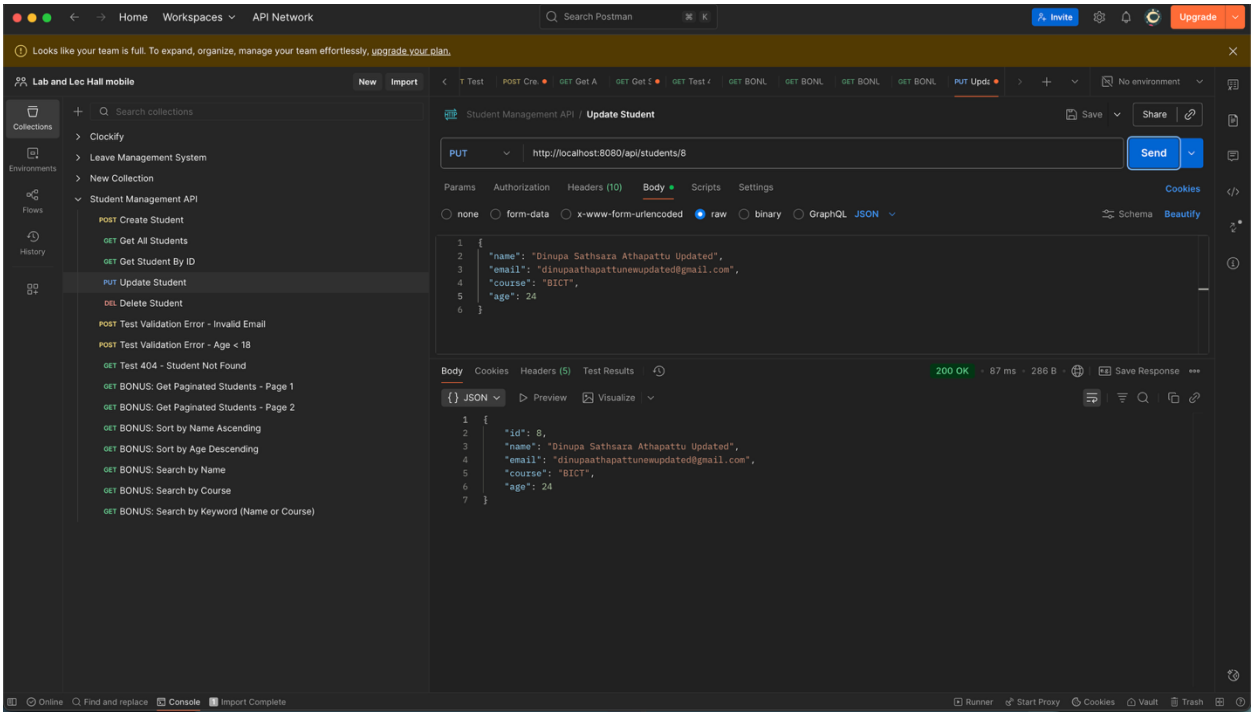
- **Get All Students**



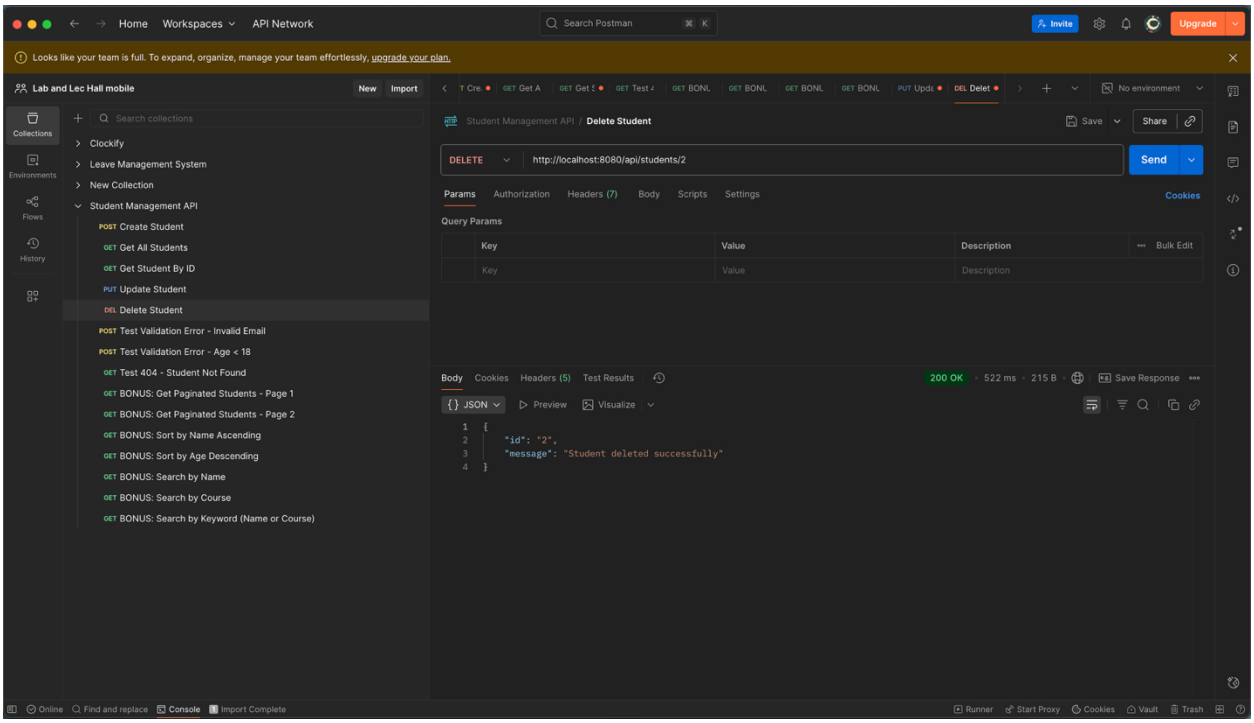
- **Get Student By ID**



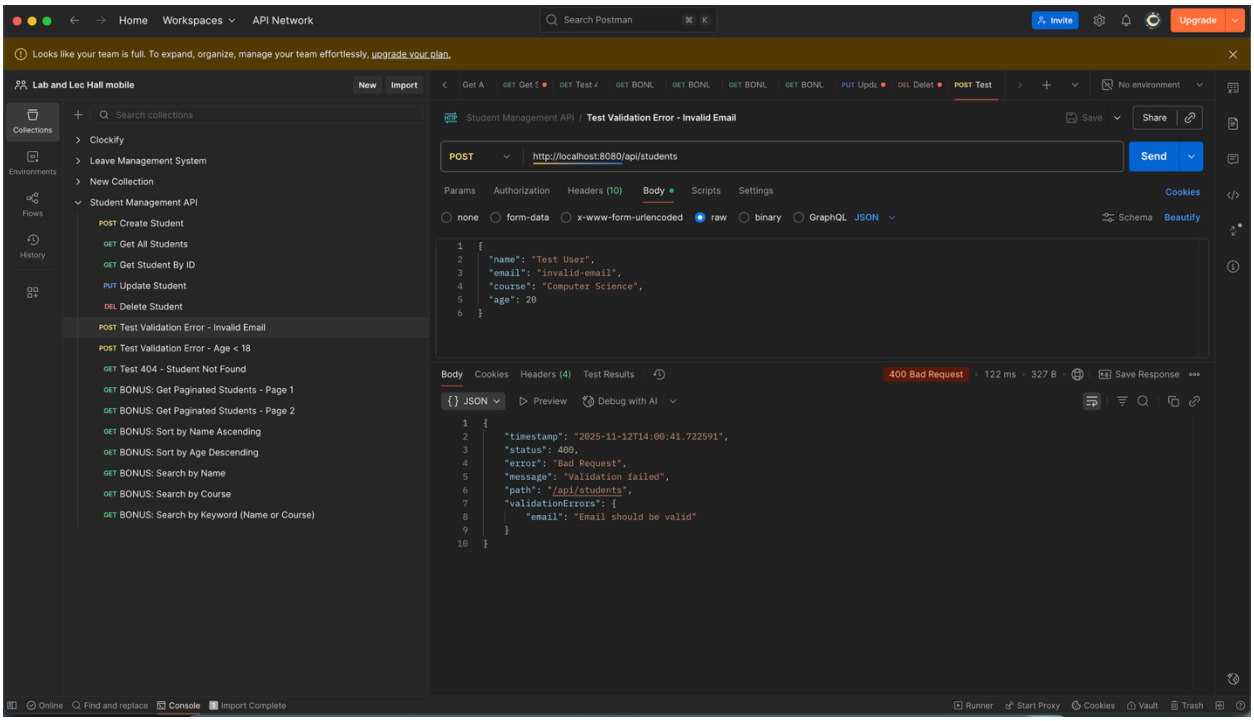
- **Update Student**



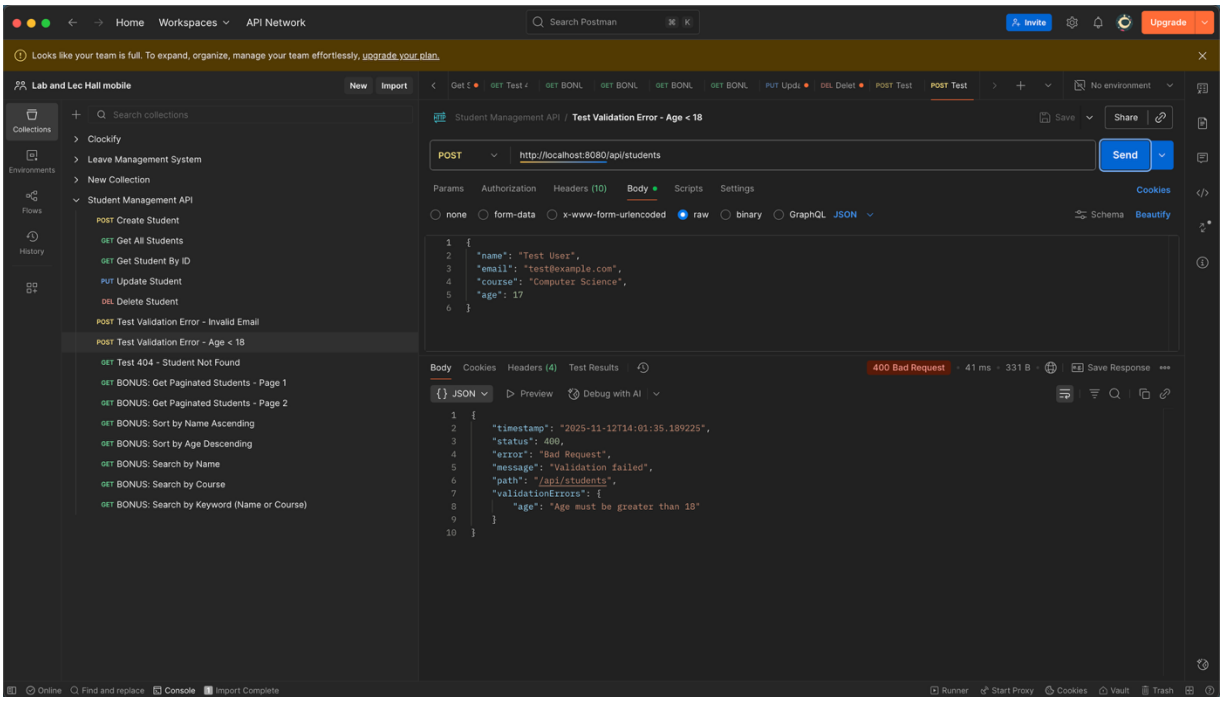
- **Delete Student**



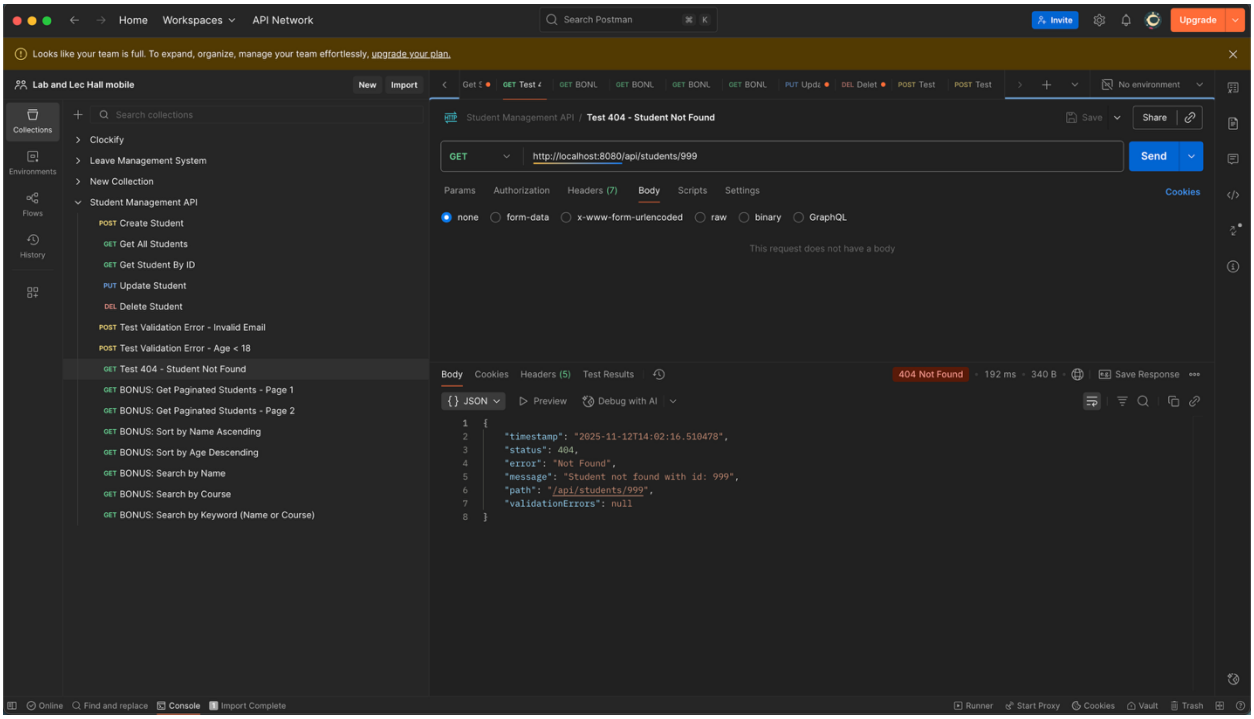
- **Test Validation Error – Invalid Email**



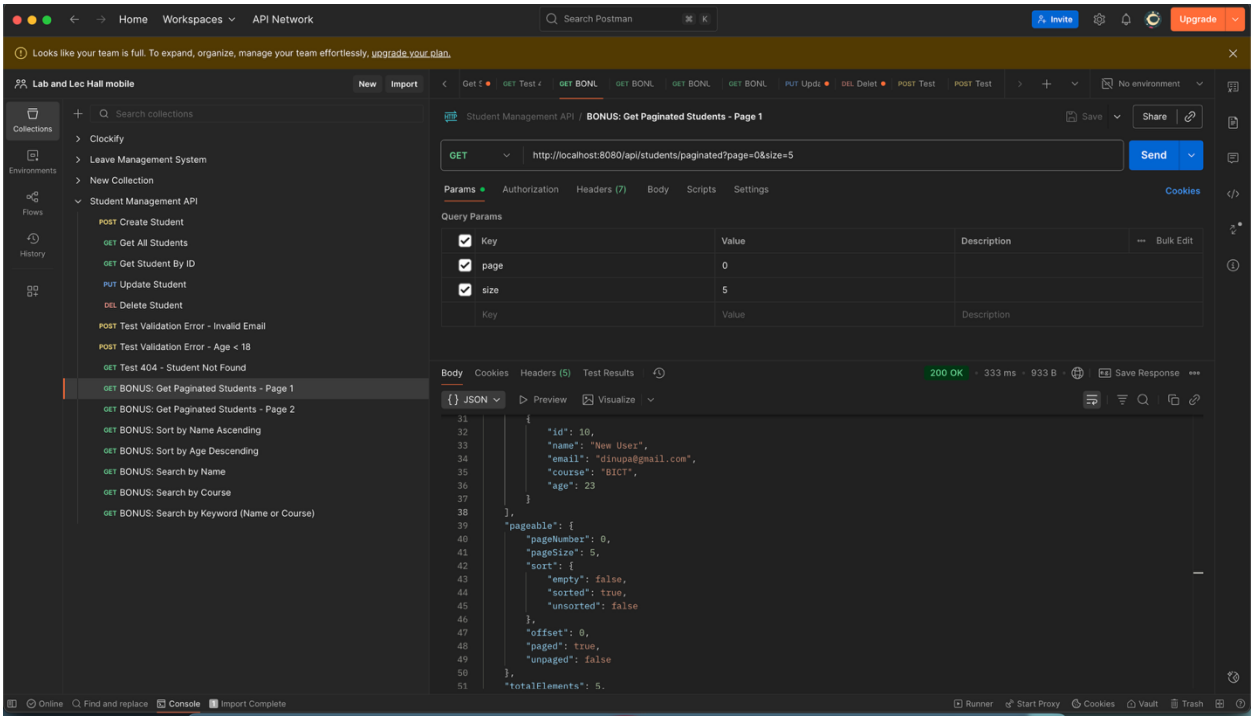
- **Test Validation Error - Age < 18**



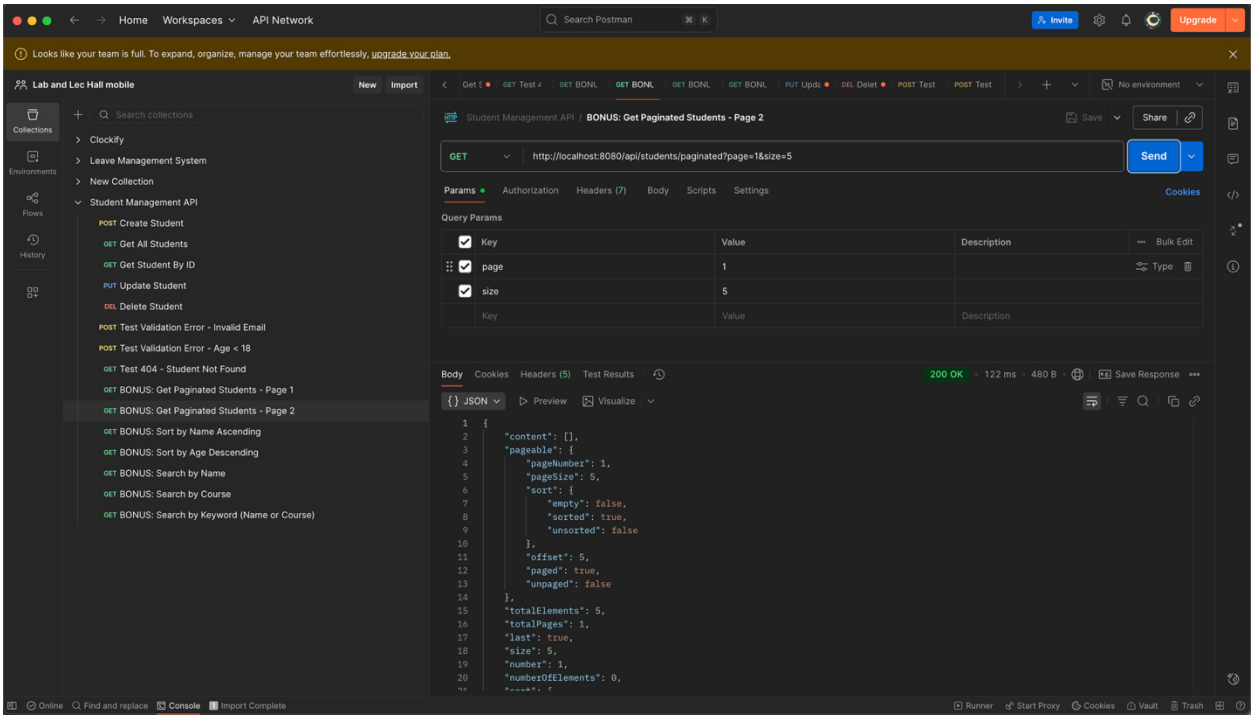
- **Test 404 - Student Not Found**



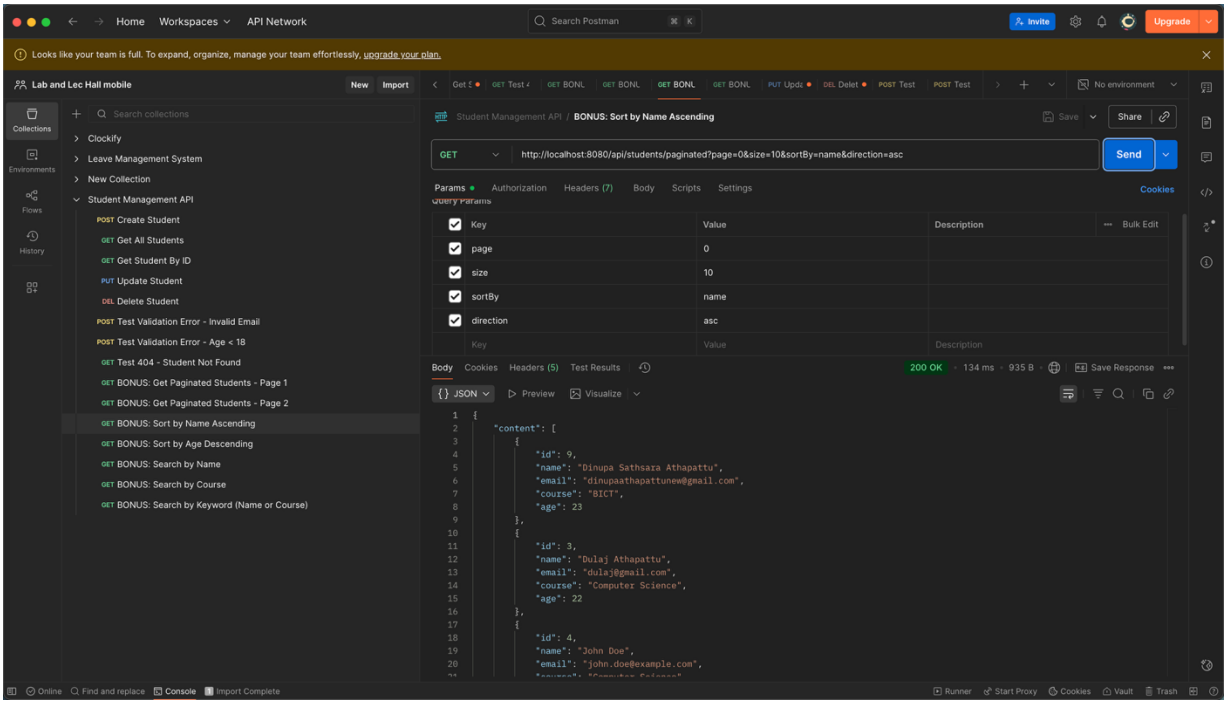
- **Optional: Get Paginated Students - Page 1**



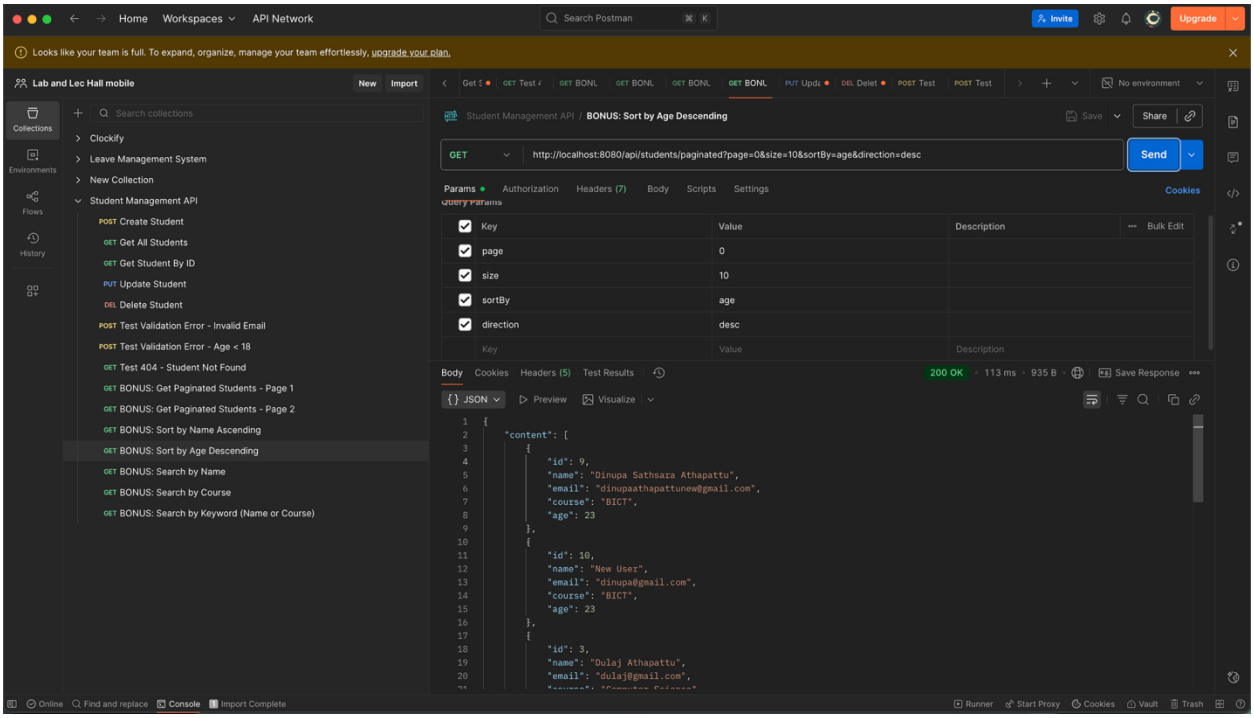
- **Optional: Get Paginated Students - Page 2**



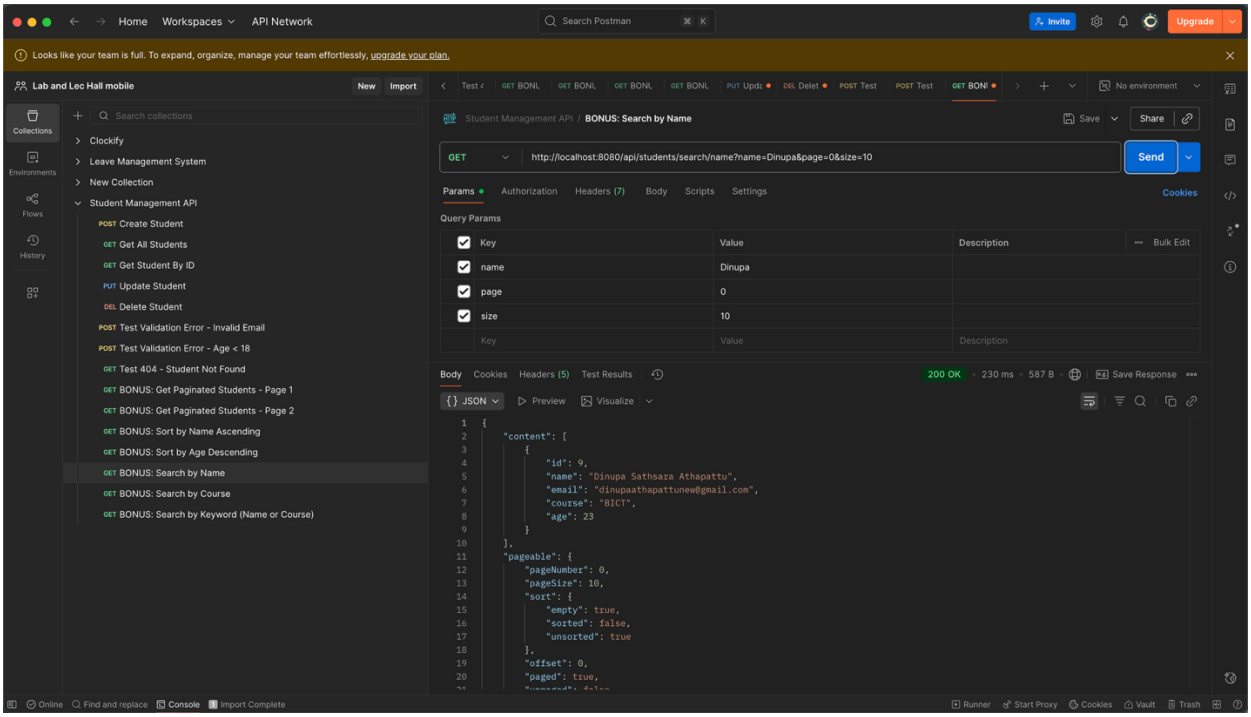
- **Optional: Sort by Name Ascending**



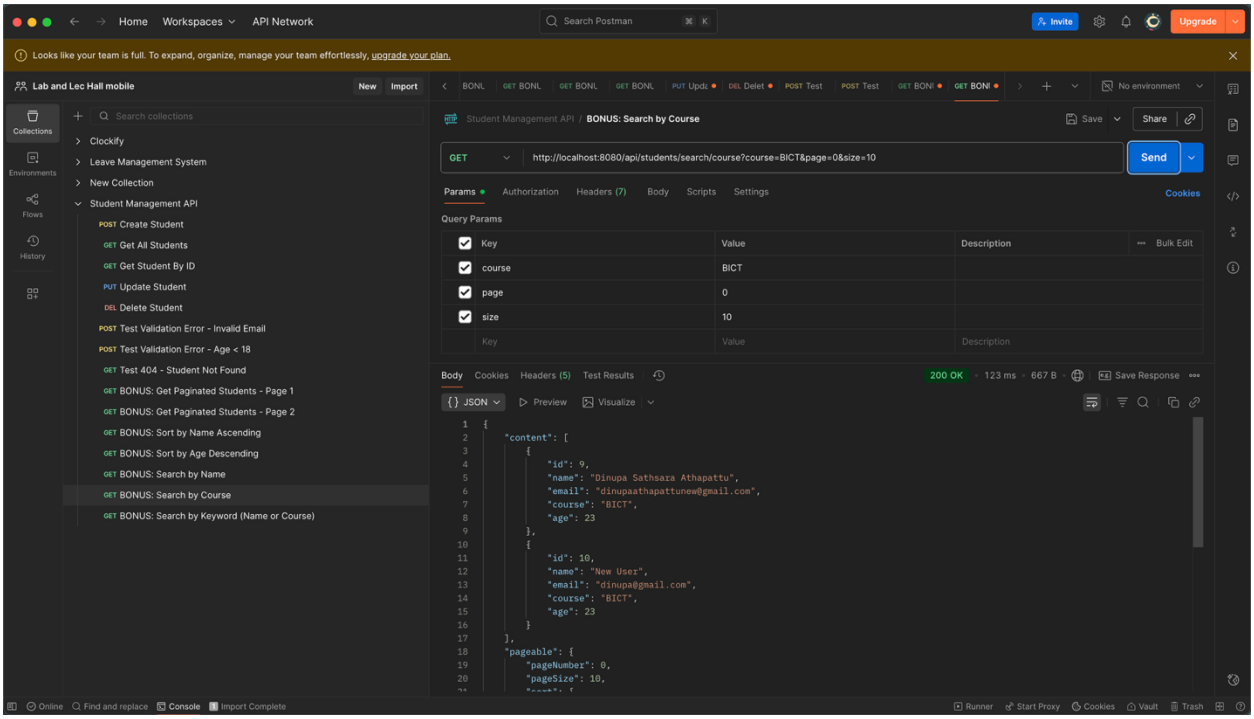
- **Optional: Sort by Age Descending**



- **Optional: Search by Name**



- **Optional: Search by Course**



- **Optional: Search by Keyword (Name or Course)**