University of Sri Jayewardenepura

Faculty of Technology

ITS 4243 - Microservices and Cloud Computing

Assignment 01

Name - Piyumika W.M.D.

Index Number - ICT/21/898

Registration Number - TE104804

Part 1: Theory

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

Spring Boot is an open-source Java-based framework that simplifies the development of production-ready applications built on the Spring Framework. It was created by Pivotal Team to address the complexity and extensive configuration requirements of traditional Spring applications.

Spring Boot is used because,

- It significantly reduces the time and effort required to set up and configure Spring applications.
- It provides auto-configuration capabilities that automatically configure your application based on the dependencies you have added to the project.
- Additionally, it comes with embedded servers like Tomcat, Jetty, or Undertow, eliminating the need to deploy WAR files to external servers.

This makes it particularly valuable for building microservices, RESTful APIs, and standalone applications quickly and efficiently.

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

The Spring Framework is a comprehensive framework for enterprise Java development that provides extensive infrastructure support for developing applications. However, it requires significant manual configuration, including XML configurations or extensive Java-based configurations, dependency management, and server setup.

Spring Boot, on the other hand, is built on top of the Spring Framework and aims to simplify the entire development process. The key differences are that Spring Boot offers auto-configuration, which automatically sets up beans based on classpath dependencies, while Spring Framework requires manual setup. Spring Boot includes embedded servers, removing deployment complexities, whereas Spring Framework requires external server deployment. Spring Boot also provides opinionated defaults and starter dependencies that bundle commonly used libraries, making dependency management much easier. Essentially, Spring Boot leverages the powerful features of Spring Framework and makes them quicker and simpler to implement.

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

Inversion of Control is a design principle where the control of object creation and management is transferred from the application code to a container or framework. Instead of objects creating their own dependencies, the framework creates and manages these objects and their lifecycles.

Dependency Injection is a specific implementation of IoC where dependencies are "injected" into a class rather than the class creating them itself. In Spring, this means that when a class needs another class to function, Spring automatically provides (injects) that dependency. There are three types of dependency injection: constructor injection, where dependencies are provided through the class constructor; setter injection, where dependencies are provided through setter methods; and field injection, where dependencies are injected directly into fields using annotations. This approach promotes loose coupling, makes code more testable, and improves maintainability since classes don't need to know how to create their dependencies.

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

The application.properties or application.yml files serve as the central configuration files for Spring Boot applications. These files allow developers to externalize configuration, meaning you can modify application behavior without changing the code itself.

These configuration files are used to define various settings such as server port numbers, database connection details including URL, username, and password, logging levels and patterns, and application-specific custom properties. They can also configure profile-specific settings for different environments like development, testing, and production. The application.yml format uses YAML syntax, which is more readable and supports hierarchical data better than the properties format, though both serve the same purpose. Spring Boot automatically reads these files at startup and applies the configurations throughout the application.

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

REST (Representational State Transfer) API is an architectural style for designing networked applications. It uses HTTP protocols to enable communication between client and server, where resources are identified by URIs and manipulated using standard HTTP methods. REST APIs are stateless, meaning each request contains all the information needed to process it, and they typically exchange data in JSON or XML format.

The primary HTTP methods used in REST APIs are:

- **GET**: Used to retrieve or read resources from the server. It should not modify any data and is considered safe and idempotent.
- **POST**: Used to create new resources on the server. It submits data to be processed and typically results in the creation of a new resource.
- **PUT**: Used to update or replace an existing resource completely. It is idempotent, meaning multiple identical requests should have the same effect as a single request.
- **PATCH**: Used to partially update an existing resource, modifying only the specified fields rather than replacing the entire resource.
- **DELETE**: Used to remove or delete a resource from the server.

Additional methods like HEAD, OPTIONS, and TRACE exist but are less commonly used in typical REST API development.

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

Spring Data JPA is a part of the Spring Data project that makes it easy to implement JPA (Java Persistence API) based repositories. It significantly reduces the amount of boilerplate code required to implement data access layers by providing automatic implementation of repository interfaces.

An **Entity** is a Java class that represents a table in a relational database. Each instance of an entity corresponds to a row in that table. Entities are annotated with @Entity and typically include an identifier field annotated with @Id. The entity's fields map to table columns, and relationships between entities can be defined using annotations like @OneToMany, @ManyToOne, and @ManyToMany.

A **Repository** is an interface that provides methods to perform database operations on entities. By extending Spring Data JPA interfaces like JpaRepository or CrudRepository, you automatically get implementations for common operations such as save, find, delete, and more. You can also define custom query methods by following naming conventions or using the @Query annotation. This abstraction layer eliminates the need to write repetitive data access code.

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

These annotations are all stereotypes that mark classes as Spring-managed components, but they serve different purposes and convey different semantic meanings in the application architecture.

@Component is the most generic stereotype annotation. It indicates that a class is a Spring-managed component and can be used for any class that should be automatically detected through classpath scanning.

@Service is a specialization of @Component used for classes that contain business logic. It marks the service layer of your application and indicates that the class performs service tasks or business operations. While functionally similar to @Component, it provides better semantic clarity.

@Repository is used for classes that access the database and perform data access operations. It's a specialization of @Component that not only marks the class as a Spring bean but also enables automatic exception translation, converting database-related exceptions into Spring's DataAccessException hierarchy.

@Controller is used for classes that handle web requests in a Spring MVC application. These classes return view names that are resolved to actual view templates. The controller processes requests and prepares model data for the view layer.

@RestController is a combination of @Controller and @ResponseBody. It's specifically designed for RESTful web services where every method returns domain objects instead of views. The response is automatically serialized into JSON or XML format, making it ideal for building REST APIs.

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

@Autowired is an annotation used to enable automatic dependency injection in Spring. When you mark a field, constructor, or setter method with @Autowired, Spring automatically resolves and injects the appropriate bean from the application context. This eliminates the need to manually instantiate dependencies.

However, there are situations where @Autowired should be avoided or used carefully. Field injection using @Autowired directly on fields should generally be avoided because it makes testing difficult since you cannot easily inject mock dependencies, creates tight coupling to the Spring framework, and makes dependencies less explicit. Additionally, it can lead to circular dependencies that are harder to detect.

The recommended alternative is constructor injection, which makes dependencies explicit and required, facilitates testing by allowing dependencies to be provided without Spring, enables immutability by using final fields, and makes circular dependencies immediately obvious at startup. Constructor injection should be the preferred approach, and when there are too many dependencies being autowired into a single class, it might indicate a design problem where the class has too many responsibilities and should be refactored.

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

Exception handling in Spring Boot can be implemented globally using the @ControllerAdvice annotation. This provides a centralized way to handle exceptions across the entire application rather than handling them individually in each controller.

A class annotated with @ControllerAdvice acts as a global exception handler. Within this class, you define methods annotated with @ExceptionHandler that specify which exception types they handle. When an exception occurs anywhere in your application's controllers, Spring automatically routes it to the appropriate handler method in the @ControllerAdvice class.

These exception handler methods can return customized error responses, including appropriate HTTP status codes, error messages, and additional details in a structured format like JSON. You can create different handler methods for different exception types, allowing you to provide specific error responses for validation errors, database errors, authentication failures, and custom business exceptions. You can also use @ResponseStatus to automatically set HTTP status codes or return ResponseEntity objects for complete control over the response. This approach promotes clean code by separating error handling logic from business logic, ensures consistent error responses across the application, and makes it easier to maintain and modify error handling behavior.

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

Maven and Gradle are build automation and dependency management tools that play crucial roles in Spring Boot projects. They manage the entire build lifecycle, from compiling code to packaging the final application.

Their primary role is dependency management, where they automatically download and manage all the libraries and frameworks your project needs. In Spring Boot projects, you can use starter dependencies like spring-boot-starter-web or spring-boot-starter-data-jpa, which are parent POMs or dependency bundles that include all necessary related libraries. This eliminates the need to manually specify each individual dependency and ensures version compatibility.

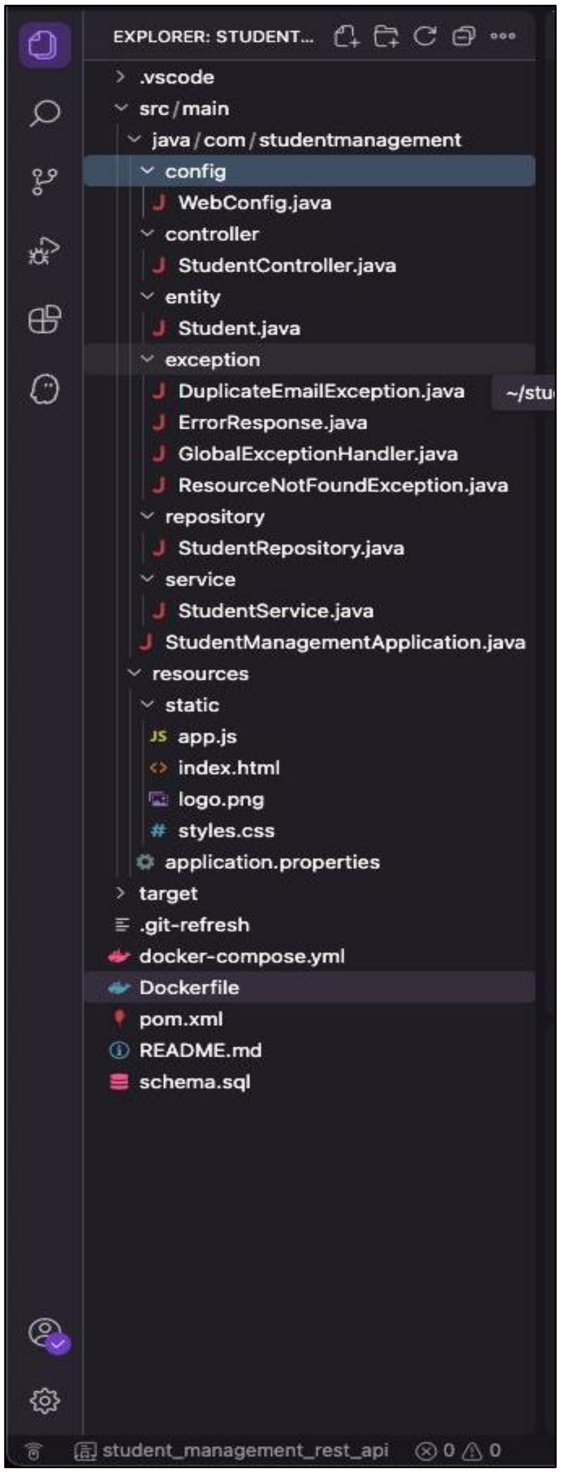
These tools also handle project building by compiling source code, running tests, packaging the application into JAR or WAR files, and managing different build profiles for various environments. They provide plugin support that extends functionality, such as the Spring Boot Maven or Gradle plugin that creates executable JAR files with embedded servers. Additionally, they manage project structure and configuration through files like pom.xml for Maven or build.gradle for Gradle, where you define project metadata, dependencies, plugins, and build instructions. Both tools integrate well with IDEs and CI/CD pipelines, though Gradle is generally faster and uses a more flexible Groovy or Kotlin DSL syntax, while Maven uses XML and follows a more convention-based approach.

Part 2 :

GitHub Repo Link : Student Management REST API

The GitHub repository includes the **source code, README.md with setup instructions and API endpoints, SQL file, and Postman testing screenshots.**

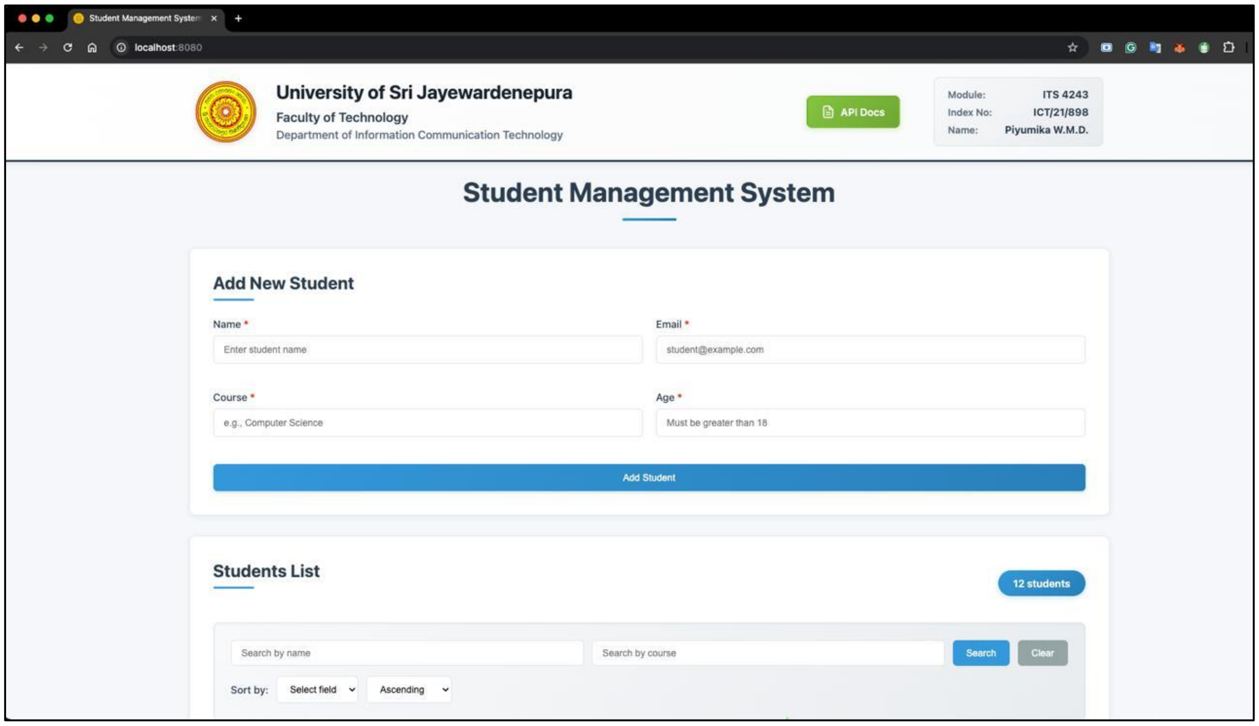
Here are some screenshots of the frontend interface**.**



```
EXPLORER: STUDENT...
> .vscode
v src / main
  v java / com / studentmanagement
    v config
      WebConfig.java
    v controller
      StudentController.java
    v entity
      Student.java
    v exception
      DuplicateEmailException.java        ~/stu
      ErrorResponse.java
      GlobalExceptionHandler.java
      ResourceNotFoundException.java
    v repository
      StudentRepository.java
    v service
      StudentService.java
    StudentManagementApplication.java
  v resources
    v static
      app.js
      index.html
      logo.png
      styles.css
    application.properties
> target
.git-refresh
docker-compose.yml
Dockerfile
pom.xml
README.md
schema.sql

student_management_rest_api   ⊗ 0 ⚠ 0
```

Figure 1 - Folder structure



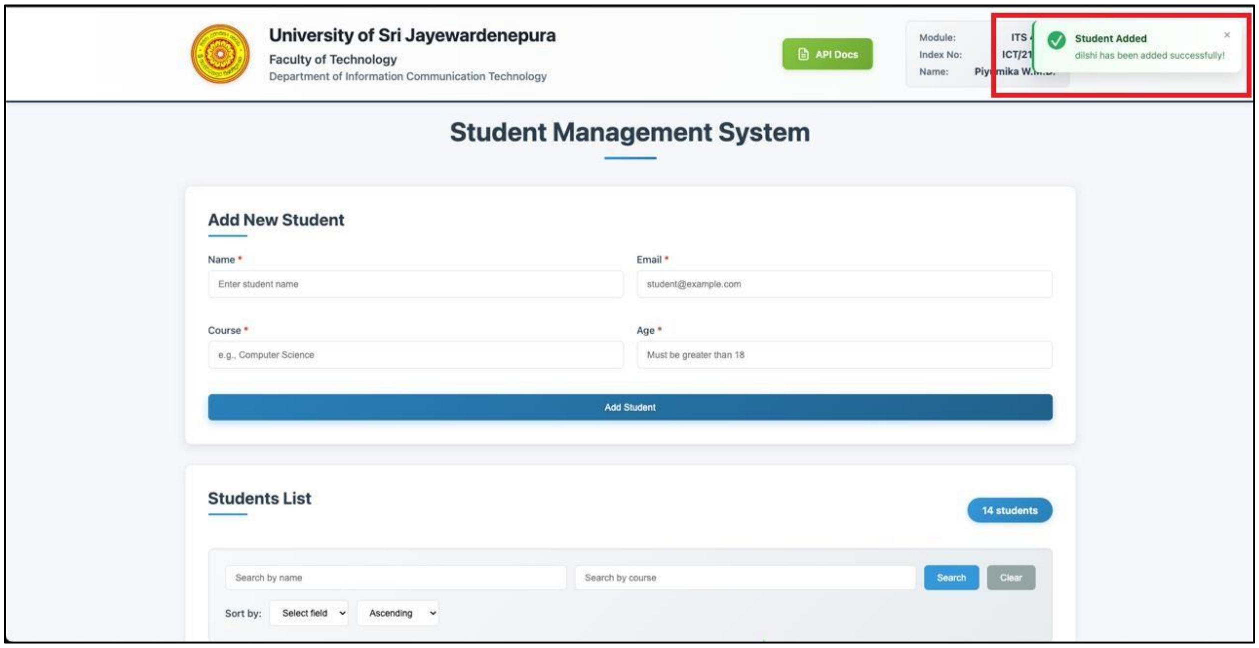
University of Sri Jayewardenepura
Faculty of Technology
Department of Information Communication Technology

Module: ITS 4243
Index No: ICT/21/898
Name: Piyumika W.M.D.

Student Management System

12 students

Figure 2- Data Entry Fields



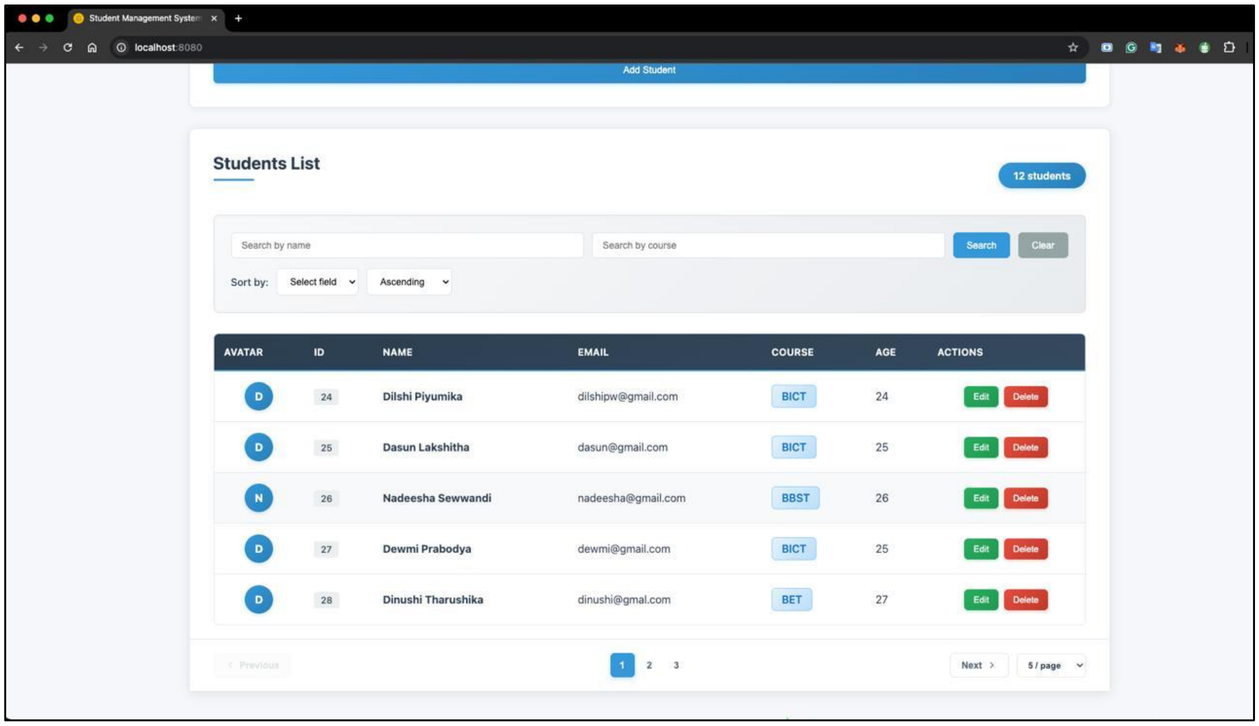
University of Sri Jayewardenepura
Faculty of Technology
Department of Information Communication Technology

Student Added
dilshi has been added successfully!

Student Management System

14 students

Figure 3- Successfully added a student



Student Management System

localhost:8080

Add Student

Students List

12 students

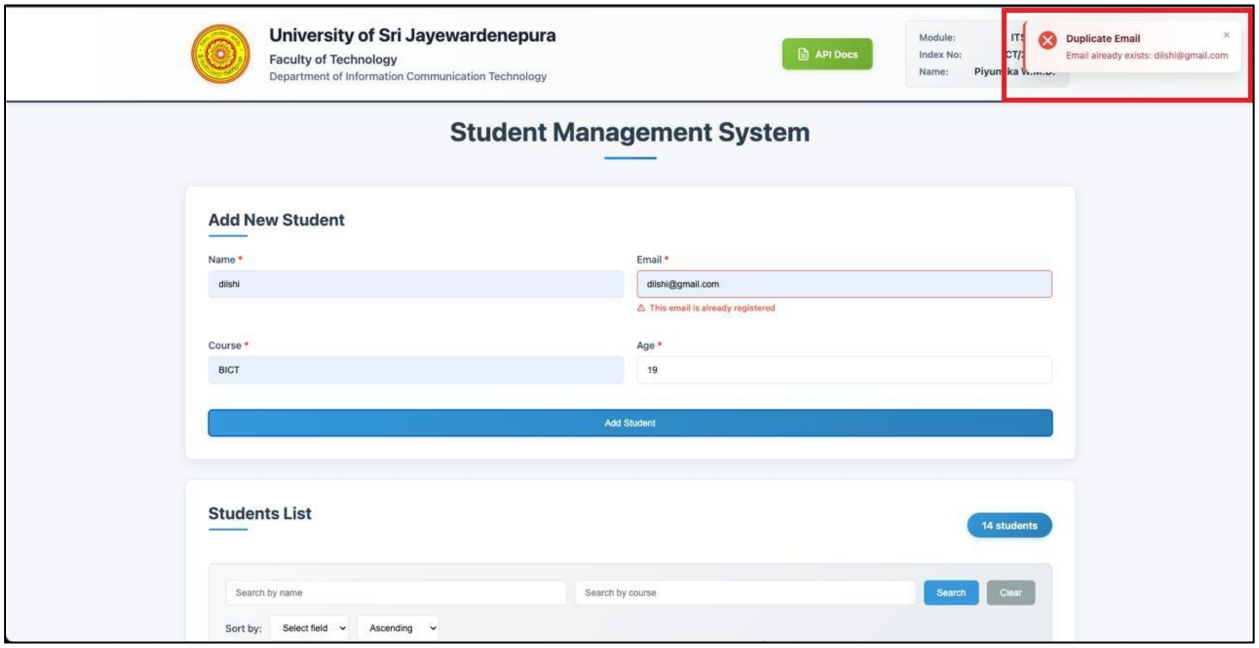
Search by name | Search by course | Search | Clear

Sort by: Select field | Ascending

AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
D	24	Dilshi Piyumika	dilshipw@gmail.com	BICT	24	Edit Delete
D	25	Dasun Lakshitha	dasun@gmail.com	BICT	25	Edit Delete
N	26	Nadeesha Sewwandi	nadeesha@gmail.com	BBST	26	Edit Delete
D	27	Dewmi Prabodya	dewmi@gmail.com	BICT	25	Edit Delete
D	28	Dinushi Tharushika	dinushi@gmal.com	BET	27	Edit Delete

< Previous | 1 2 3 | Next > | 5 / page

Figure 4- Added students list



University of Sri Jayewardenepura
Faculty of Technology
Department of Information Communication Technology

API Docs

Module: ITS
Index No: CT/
Name: Piyum ka W.M.D.

Duplicate Email
Email already exists: dilshi@gmail.com

Student Management System

Add New Student

Name *	dilshi
Email *	dilshi@gmail.com
	⚠ This email is already registered
Course *	BICT
Age *	19

Add Student

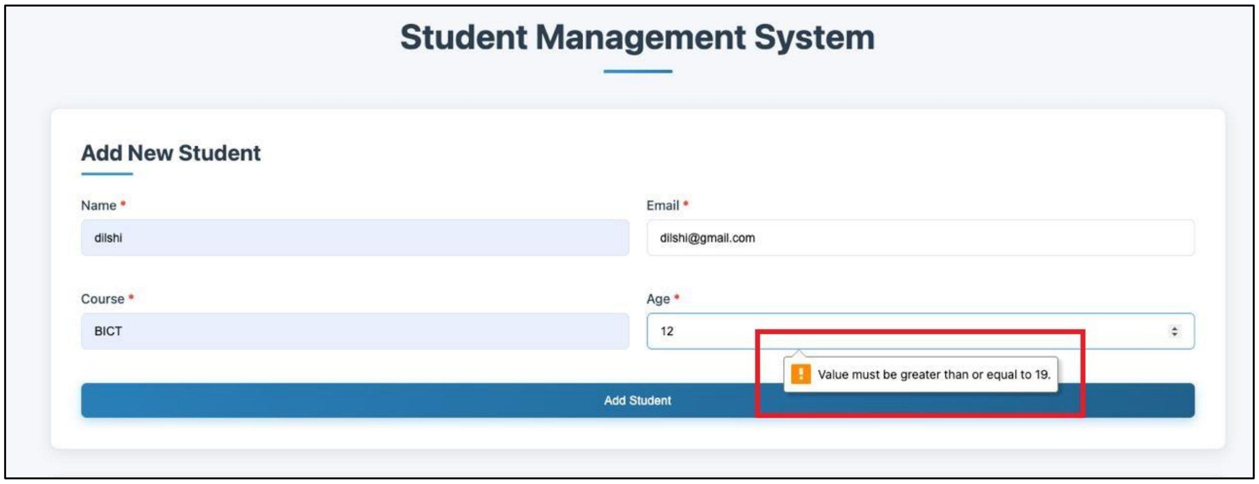
Students List

14 students

Search by name | Search by course | Search | Clear

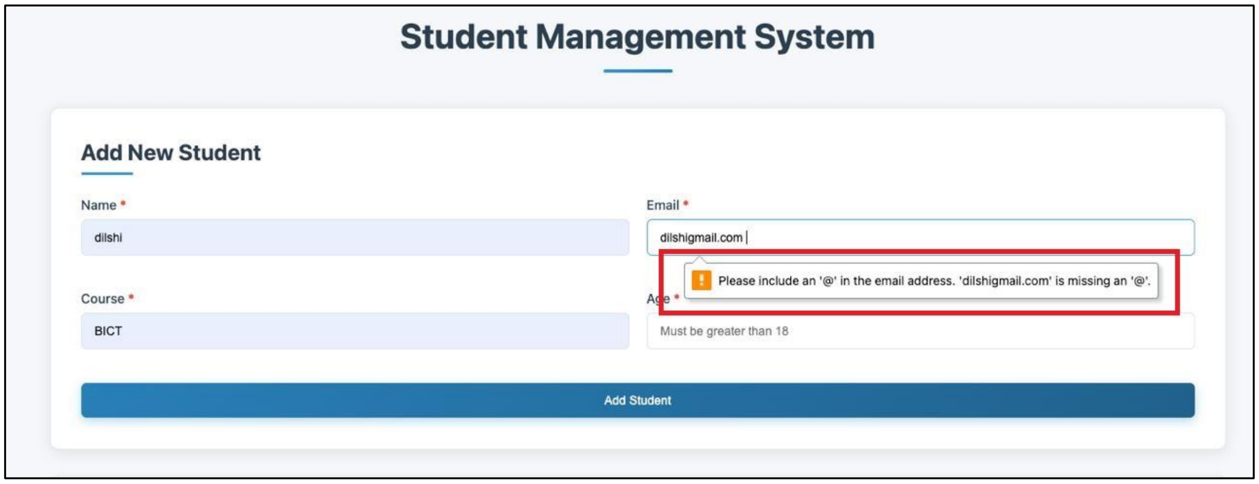
Sort by: Select field | Ascending

Figure 5 - Error message for duplicate emails



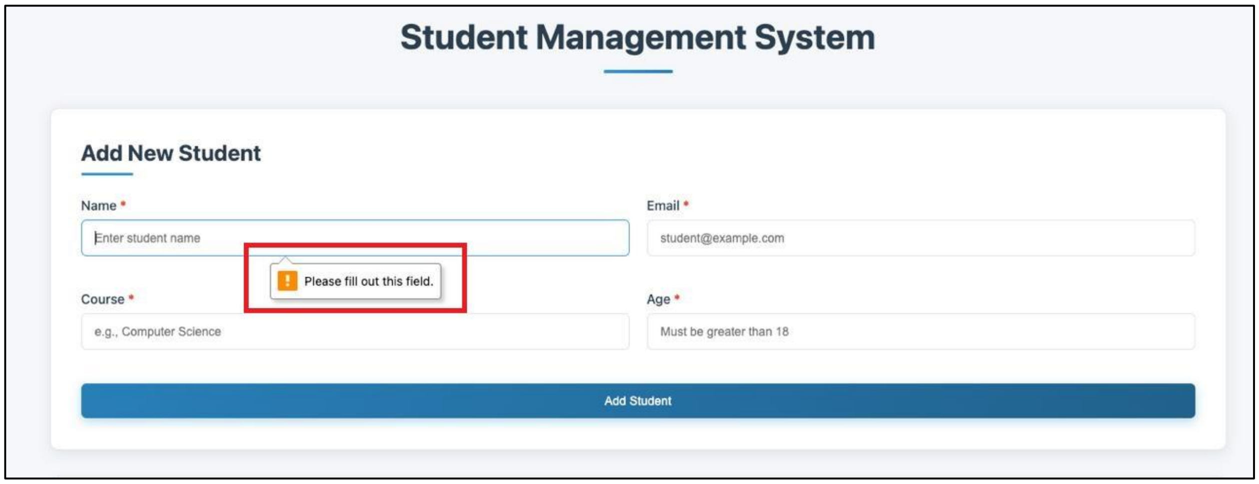
Value must be greater than or equal to 19.

Figure 6 - Age validation



Please include an '@' in the email address. 'dilshigmail.com' is missing an '@'.

Figure 7- Email format validation



Student Management System

Add New Student

Name *

Enter student name

Please fill out this field.

Email *

student@example.com

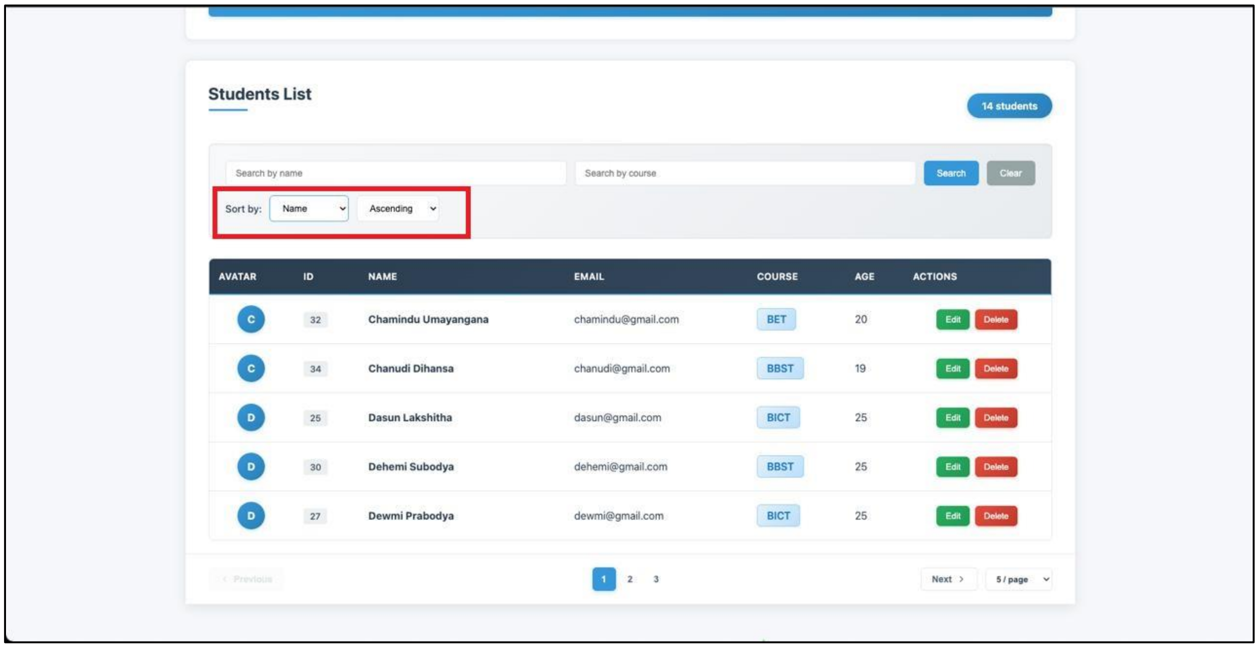
Course *

e.g., Computer Science

Age *

Must be greater than 18

Figure 8- Validate empty fields



Students List

14 students

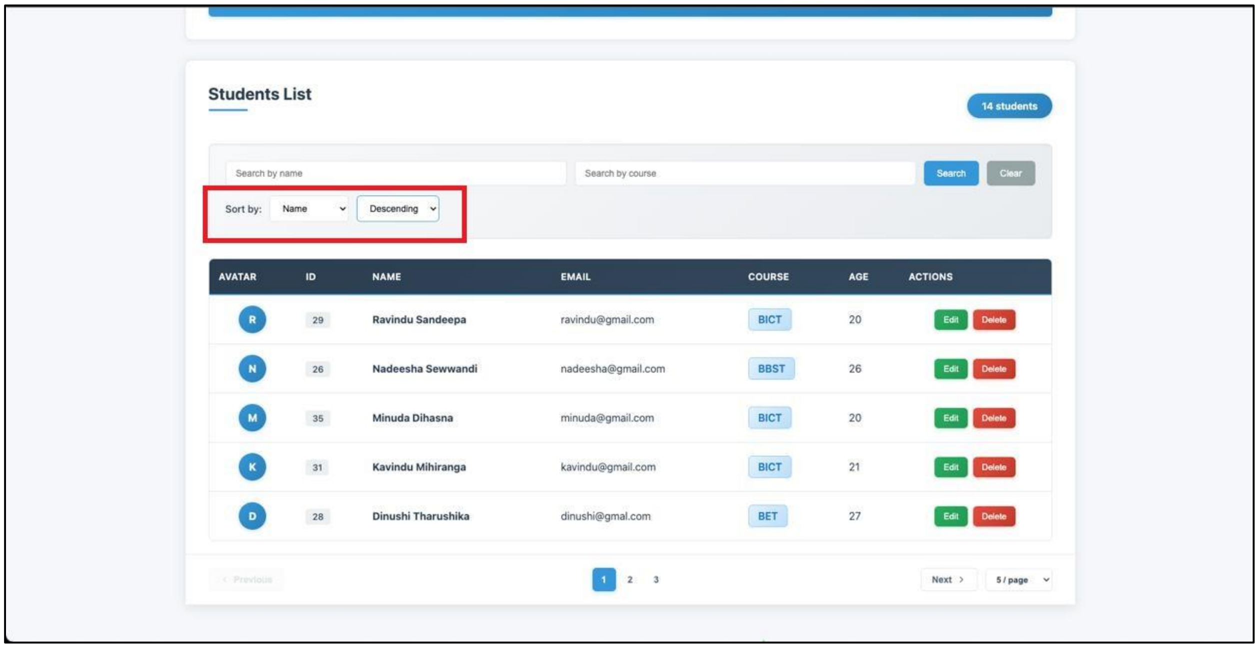
Search by name

Search by course

Sort by: Name Ascending

AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
C	32	Chamindu Umayangana	chamindu@gmail.com	BET	20	Edit Delete
C	34	Chanudi Dihansa	chanudi@gmail.com	BBST	19	Edit Delete
D	25	Dasun Lakshitha	dasun@gmail.com	BICT	25	Edit Delete
D	30	Dehemi Subodya	dehemi@gmail.com	BBST	25	Edit Delete
D	27	Dewmi Prabodya	dewmi@gmail.com	BICT	25	Edit Delete

Figure 9- Sort by name (Ascending)



Students List

14 students

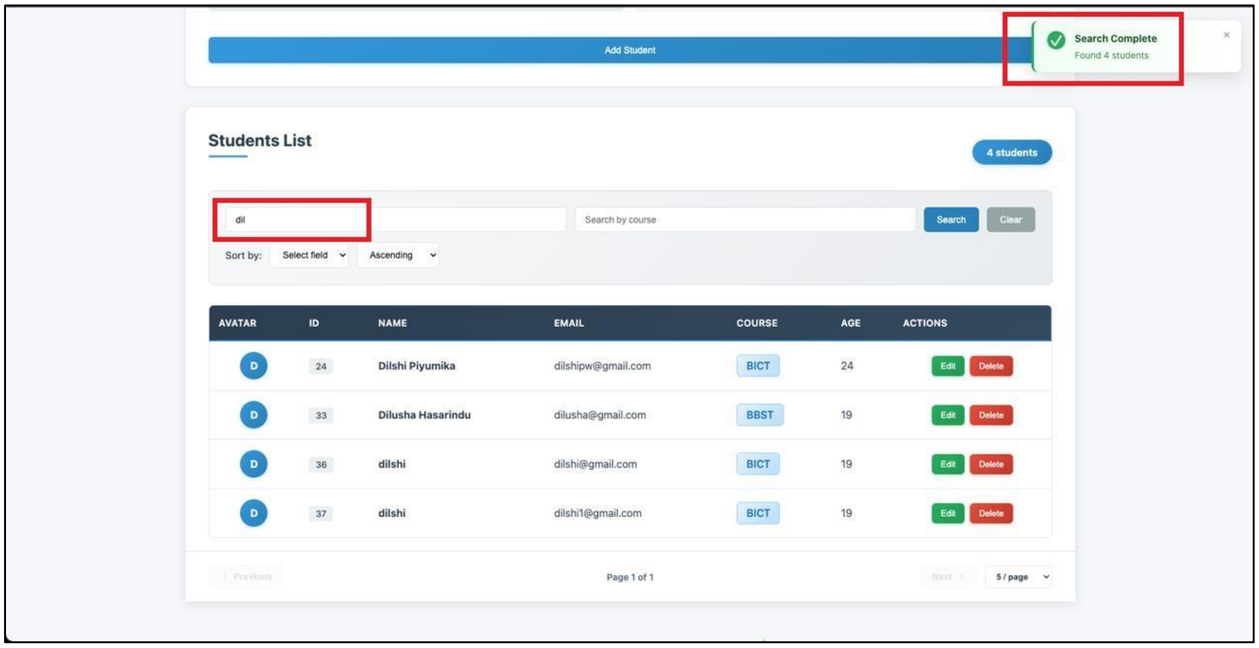
Search by name | Search by course | Search | Clear

Sort by: Name | Descending

AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
R	29	Ravindu Sandeepa	ravindu@gmail.com	BICT	20	Edit Delete
N	26	Nadeesha Sewwandi	nadeesha@gmail.com	BBST	26	Edit Delete
M	35	Minuda Dihasna	minuda@gmail.com	BICT	20	Edit Delete
K	31	Kavindu Mihiranga	kavindu@gmail.com	BICT	21	Edit Delete
D	28	Dinushi Tharushika	dinushi@gmal.com	BET	27	Edit Delete

< Previous 1 2 3 Next > 5 / page

Figure 10- Sort by Name (Descending)



Add Student

Search Complete
Found 4 students

Students List

4 students

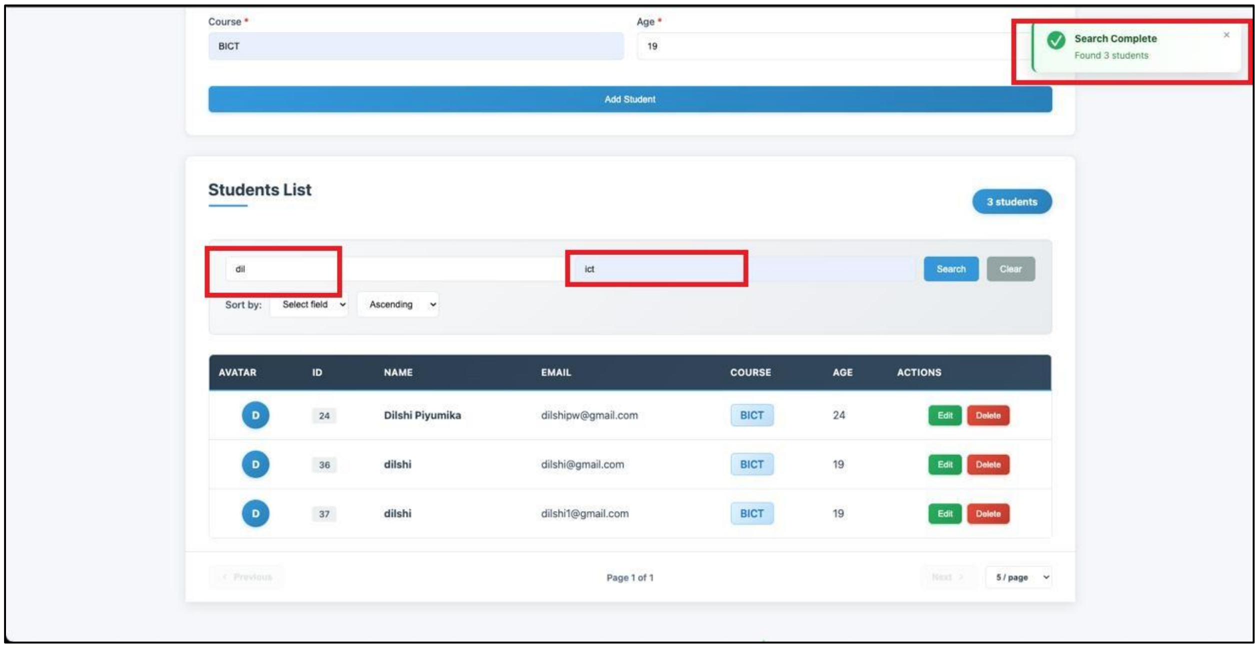
dil | Search by course | Search | Clear

Sort by: Select field | Ascending

AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
D	24	Dilshi Piyumika	dilshipw@gmail.com	BICT	24	Edit Delete
D	33	Dilusha Hasarindu	dilusha@gmail.com	BBST	19	Edit Delete
D	36	dilshi	dilshi@gmail.com	BICT	19	Edit Delete
D	37	dilshi	dilshi1@gmail.com	BICT	19	Edit Delete

< Previous Page 1 of 1 Next > 5 / page

Figure 11 - Search by name


Course *

BICT

Age *

19

Search Complete
Found 3 students

Add Student

Students List

3 students

dil

ict

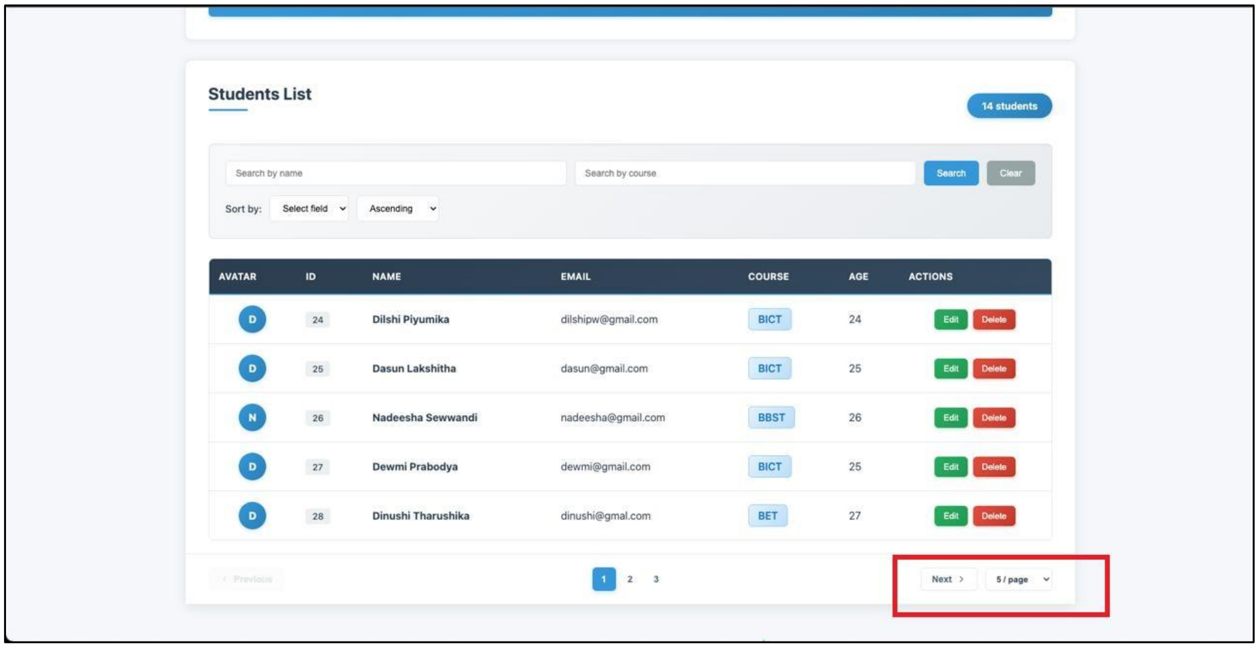
Search Clear

Sort by: Select field Ascending

AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
D	24	Dilshi Piyumika	dilshipw@gmail.com	BICT	24	Edit Delete
D	36	dilshi	dilshi@gmail.com	BICT	19	Edit Delete
D	37	dilshi	dilshi1@gmail.com	BICT	19	Edit Delete

‹ Previous Page 1 of 1 Next › 5 / page

Figure 12- Search by course


Students List

14 students

Search by name

Search by course

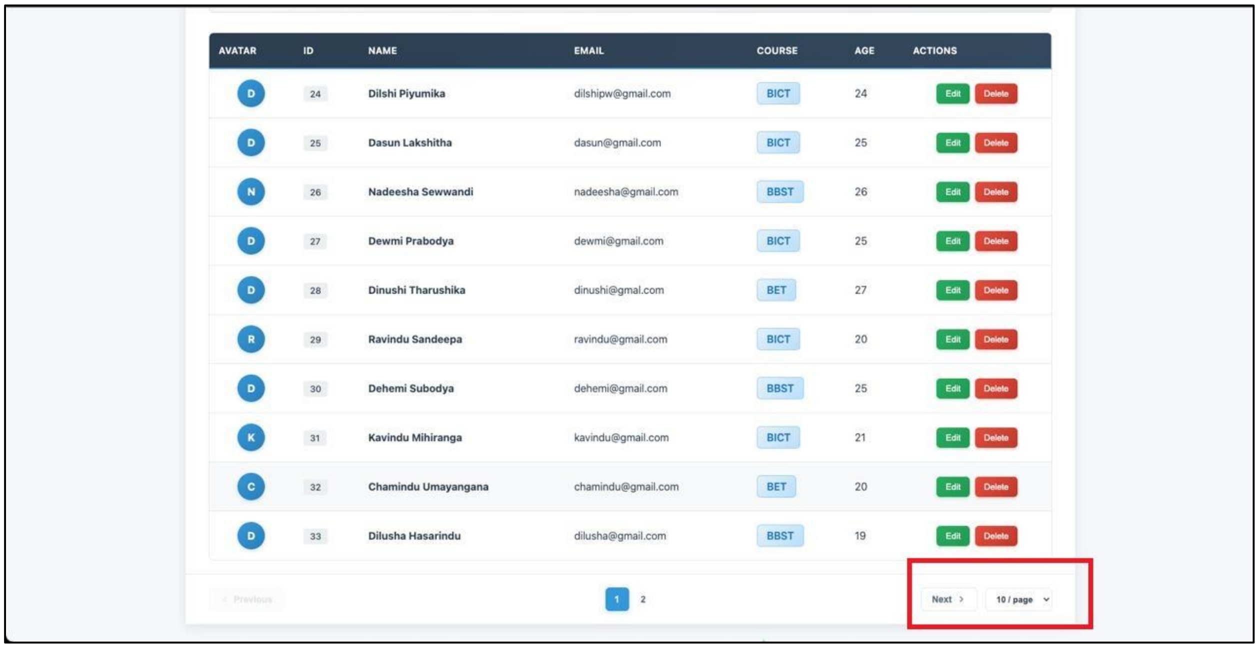
Search Clear

Sort by: Select field Ascending

AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
D	24	Dilshi Piyumika	dilshipw@gmail.com	BICT	24	Edit Delete
D	25	Dasun Lakshitha	dasun@gmail.com	BICT	25	Edit Delete
N	26	Nadeesha Sewwandi	nadeesha@gmail.com	BBST	26	Edit Delete
D	27	Dewmi Prabodya	dewmi@gmail.com	BICT	25	Edit Delete
D	28	Dinushi Tharushika	dinushi@gmal.com	BET	27	Edit Delete

‹ Previous 1 2 3 Next › 5 / page

Figure 13- Pagination (5/page)



AVATAR	ID	NAME	EMAIL	COURSE	AGE	ACTIONS
D	24	Dilshi Piyumika	dilshipw@gmail.com	BICT	24	Edit Delete
D	25	Dasun Lakshitha	dasun@gmail.com	BICT	25	Edit Delete
N	26	Nadeesha Sewwandi	nadeesha@gmail.com	BBST	26	Edit Delete
D	27	Dewmi Prabodya	dewmi@gmail.com	BICT	25	Edit Delete
D	28	Dinushi Tharushika	dinushi@gmal.com	BET	27	Edit Delete
R	29	Ravindu Sandeepa	ravindu@gmail.com	BICT	20	Edit Delete
D	30	Dehemi Subodya	dehemi@gmail.com	BBST	25	Edit Delete
K	31	Kavindu Mihiranga	kavindu@gmail.com	BICT	21	Edit Delete
C	32	Chamindu Umayangana	chamindu@gmail.com	BET	20	Edit Delete
D	33	Dilusha Hasarindu	dilusha@gmail.com	BBST	19	Edit Delete

Figure 14- Pagination (10/page)