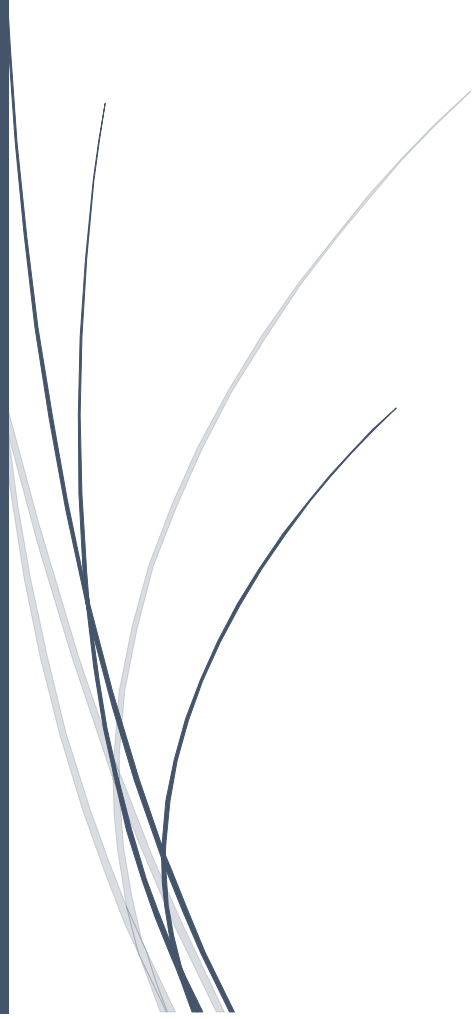
February 2025

Project Report

Course Administration System for Coatbank College (HA4G 35)



Stanislav Starishko

Contents

1. Overview

The Course Administration System for Coatbank College is designed to manage and organize data related to staff, students, courses, and enrollments. The system provides a user-friendly interface for adding, deleting, and amending records, as well as generating reports on student enrollments and course participation. The system is built using modern web technologies, including a frontend developed with **HTML, Bootstrap**, and **JavaScript**, and a backend powered by **Node.js** and **MongoDB**.

The system is structured around four main entities: **Staff**, **Student**, **Course**, and **Enrollment**. Each entity has its own set of attributes and operations, which are managed through a series of interconnected modules. The system architecture is designed to be modular, allowing for easy maintenance and future expansion.

1.1. Task Brief

You have been asked to develop a Course Administration System for Coatbank College.

The system will make use of four main files: Staff, Student, Course and Enrolment. The basic record format for each of the files is as follows:

Staff
IdNumber
Forename
Surname
DOB
Gender

Student
IdNumber
Forename
Surname
DOB
Gender

Course
CourseId
CourseName
CourseStart
CourseLength
CourseLocation

Enrolment
StudentId
CourseId

(Some modification to the record formats may be needed to deliver he required functionality.)

The files will be stored externally and read in to memory, stored as an appropriate data structure, at the start of processing. Any new or amended data will be written back to file on completion of processing.

The system should be able to carry out the following tasks:

- Add, delete or amend a Staff record
- Add, delete or amend a Student record
- Add, delete or amend a Course record
- List all the Students enrolled on a specified Course
- List all the Courses a specified Student has enrolled for

1.2. Key Features:

- **Staff Management**: Add, delete, or amend staff records.
- **Student Management**: Add, delete, or amend student records.
- **Course Management**: Add, delete, or amend course records.
- **Enrollment Management**: List all students enrolled in a specific course or all courses a specific student has enrolled in.

The system is designed to be intuitive and easy to use, with a focus on providing quick access to critical information. The frontend is built using Bootstrap for responsive design, ensuring that the system is accessible on a variety of devices.

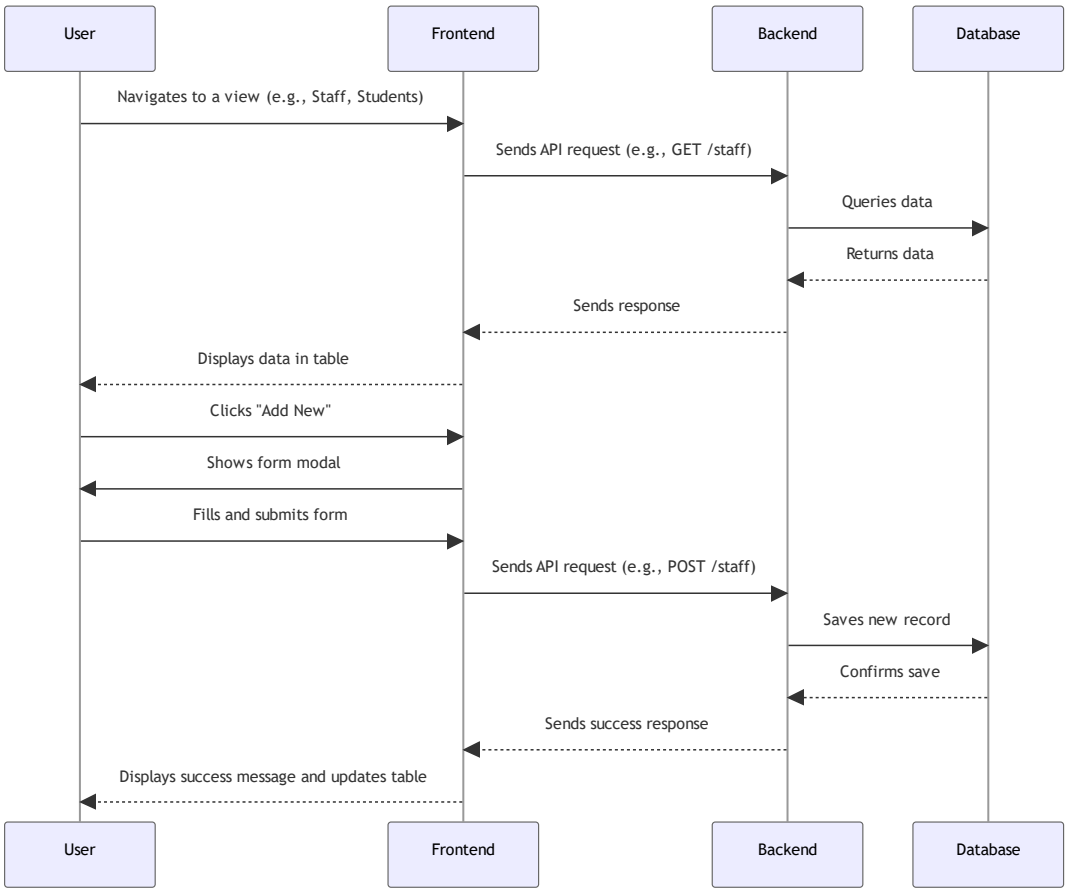
1.3. System Structure:

1.3.1. Project resources:

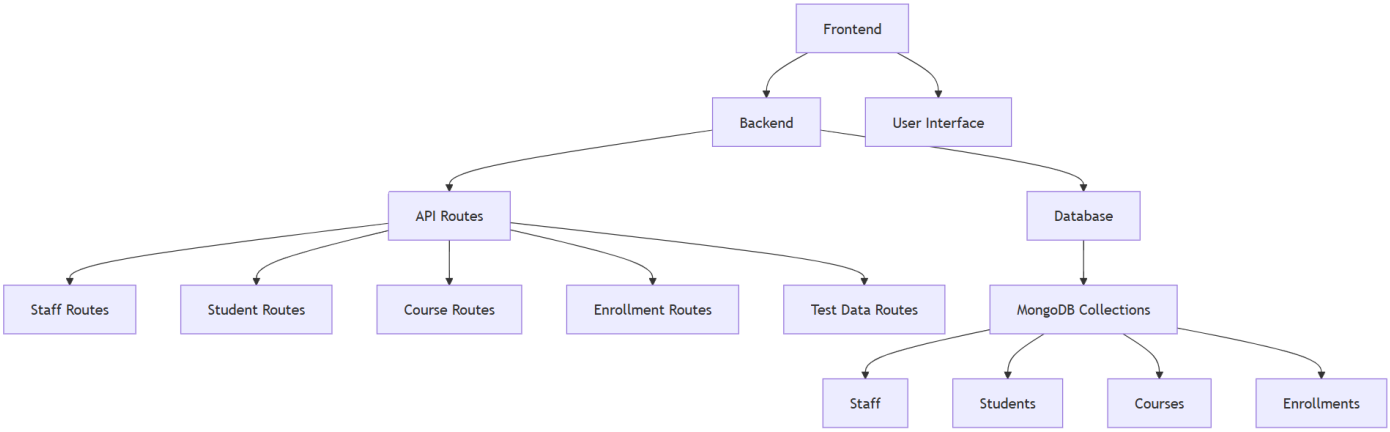
- **Source code**
- **API documentation**
- **Deployed project**

1.3.2. The system structure is reflected in the following diagrams:

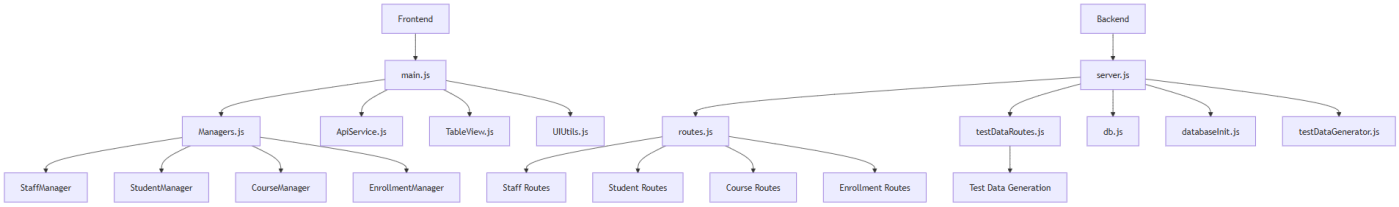
- **User Interaction Diagram**: This diagram illustrates how users interact with the system, including the flow of data between the frontend and backend.



- **System Structure Diagram**: This diagram provides an overview of the system's architecture, including the main components and their relationships.

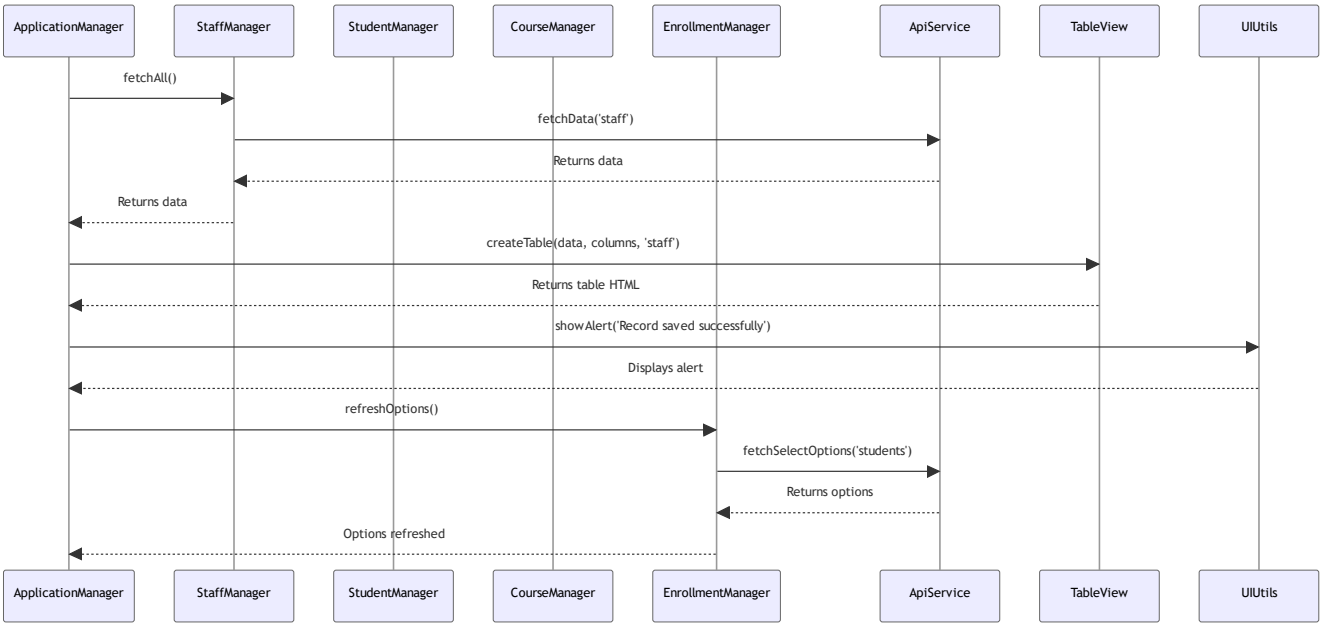


- **Module Dependency Diagram**: This diagram shows the dependencies between different modules in the system.

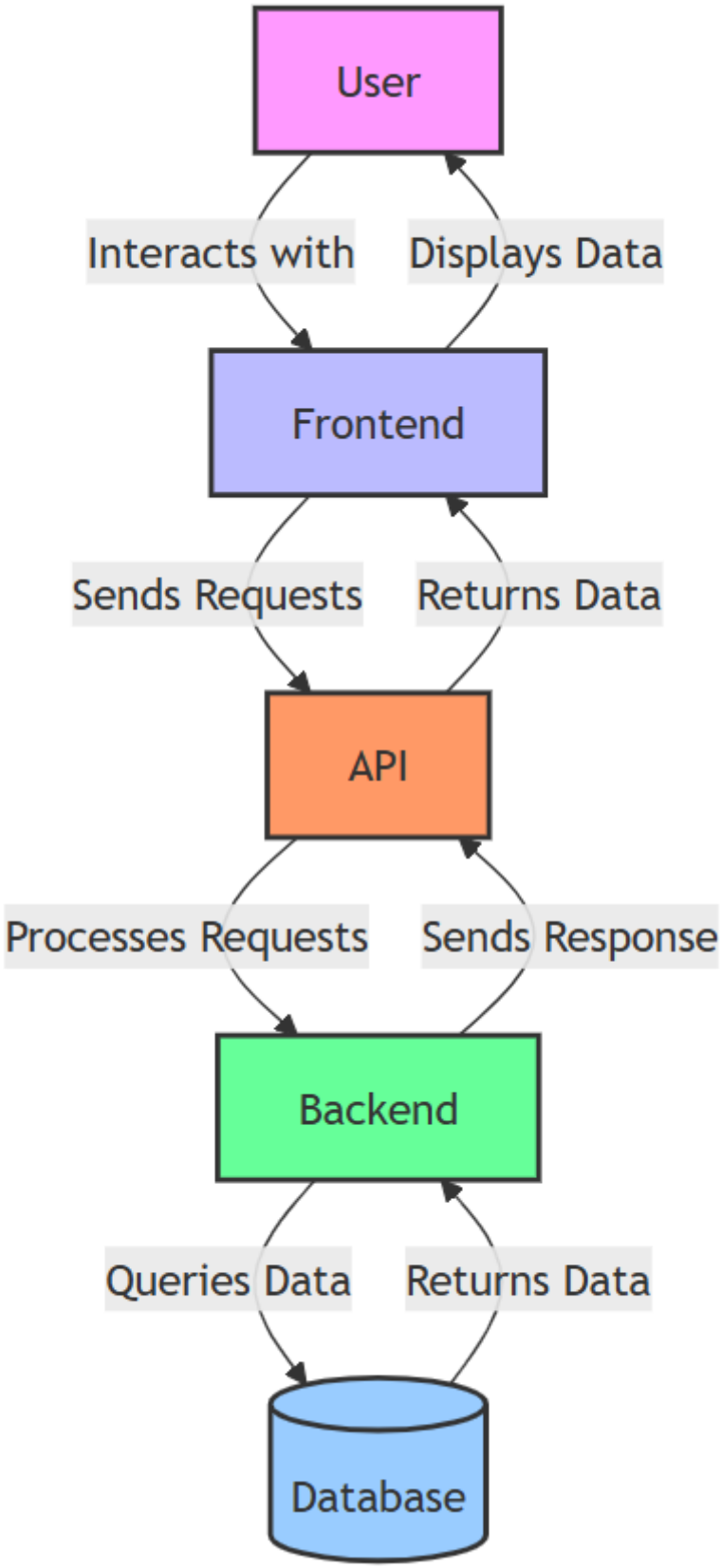


- **Class Dependency Diagram**: This diagram illustrates the relationships between classes in the system, highlighting inheritance and polymorphism.

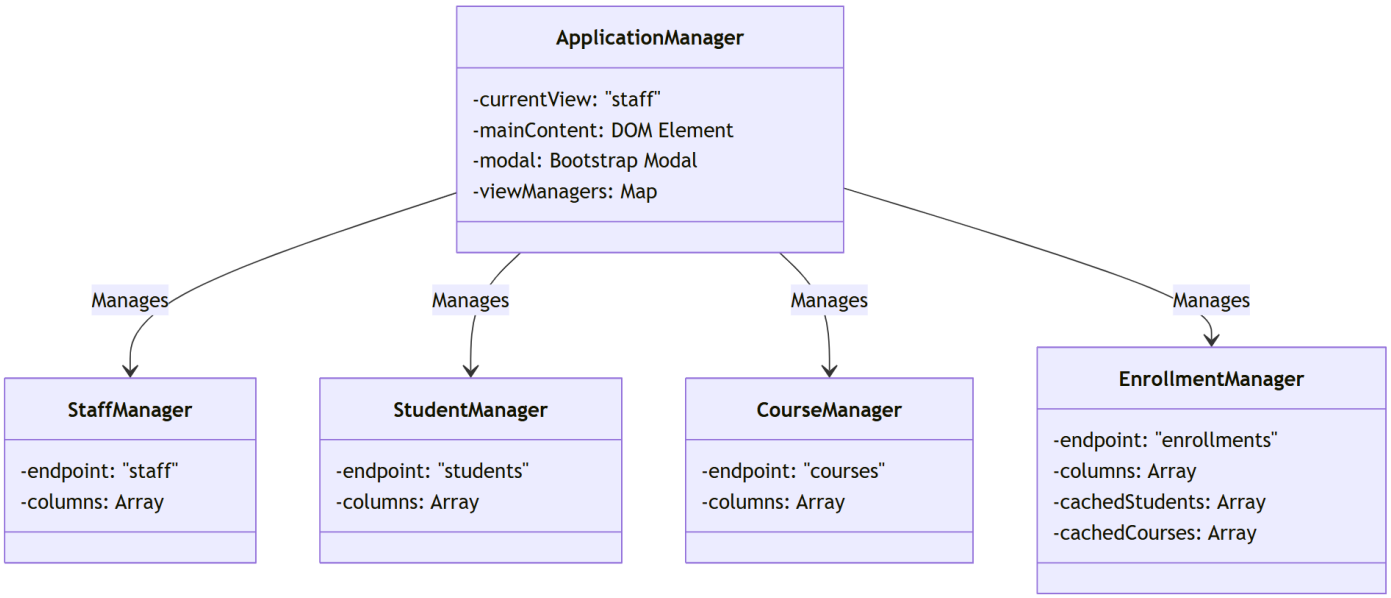
- **Class Interaction Diagram**: This diagram shows how classes interact with each other during runtime.



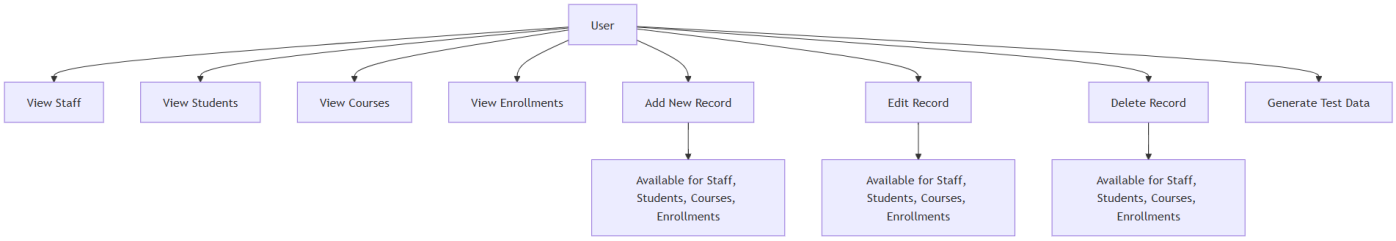
- **System Architecture Diagram**: This diagram provides a high-level view of the system's architecture, including the frontend, backend, and database.



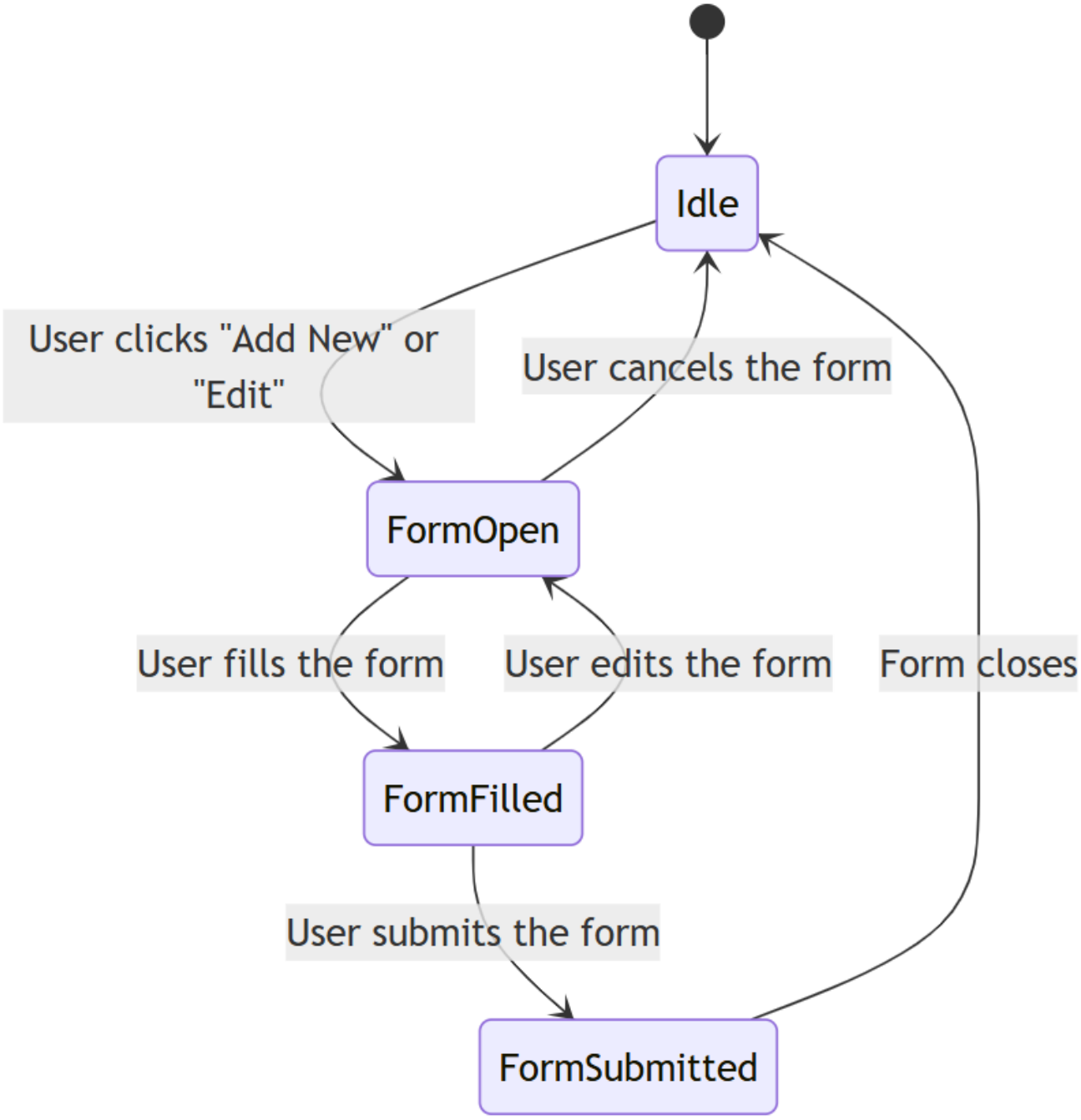
- **Object Diagram**: This diagram shows the objects instantiated from the classes and their relationships.



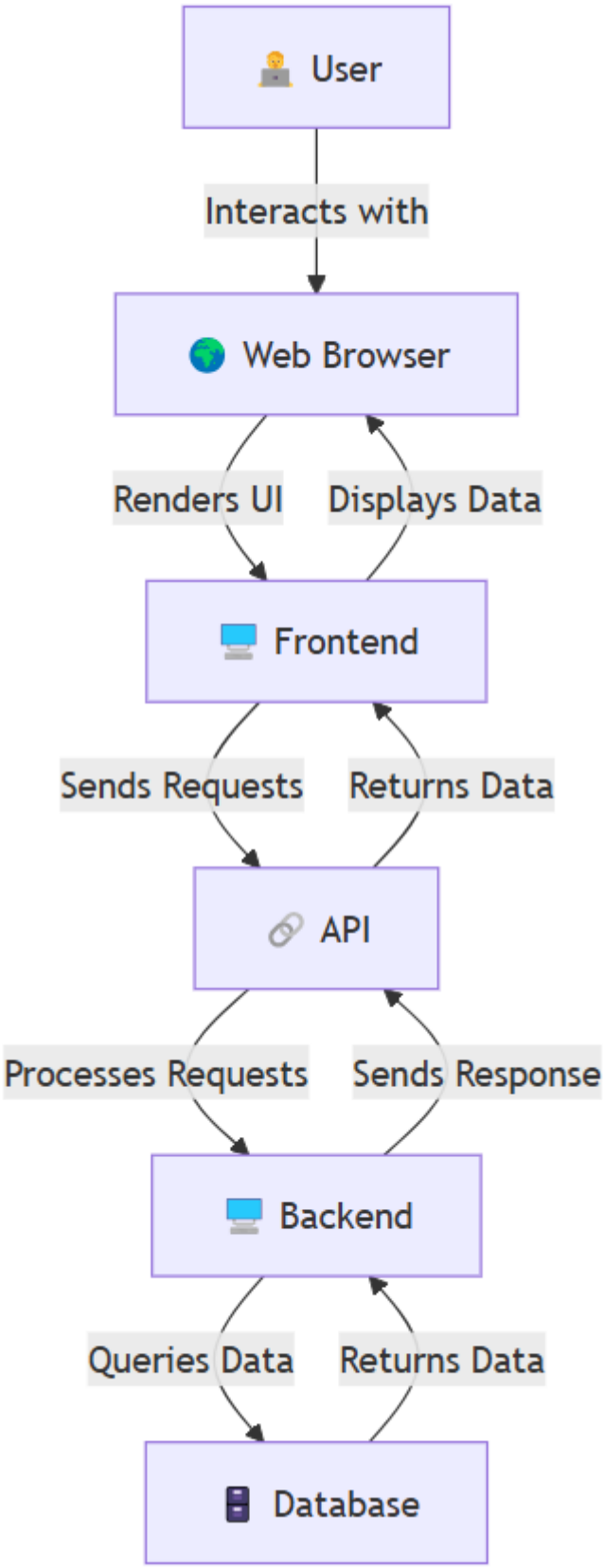
- **Use Case Diagram**: This diagram outlines the main use cases of the system, such as adding a new student or enrolling a student in a course.



- **State Diagram**: This diagram illustrates the different states of the system and how it transitions between them.



- **Deployment Diagram**: This diagram shows how the system is deployed, including the server, database, and client-side components.



2. Application of Object-Oriented Programming Constructs

2.1. Program Creation

The system is built using object-oriented programming (OOP) principles. The main entities (Staff, Student, Course, and Enrollment) are represented as classes, each with its own properties and methods. The system is designed to be modular, with each class responsible for managing its own data and operations.

2.2. Encapsulation

Encapsulation is achieved by using private fields and methods within the classes. For example, the ApplicationManager class in the frontend uses private fields (#currentView, #mainContent, etc.) to store internal state, which can only be accessed through public methods. This ensures that the internal workings of the class are hidden from the outside world, promoting data integrity and security.

2.3. Inheritance

Inheritance is used to create specialized classes that share common functionality. For example, the DataManager class serves as a base class for StaffManager, StudentManager, CourseManager, and EnrollmentManager. These child classes inherit common methods like fetchAll, create, update, and delete from the DataManager class, while also implementing their own specific logic.

2.4. Polymorphism

Polymorphism is demonstrated through the use of a single interface to handle different types of data. For example, the TableView class can create tables for different types of data (staff, students, courses, and enrollments) using the same createTable method. This allows the system to handle different types of data in a consistent manner, reducing code duplication and improving maintainability.

3. Program Construction

3.1. Use of Algorithms

The system uses various algorithms for data traversal, sorting, and searching. For example, the fetchAll method in the DataManager class retrieves all records from the database, while the createTable method in the TableView class sorts and displays the data in a tabular format. The system also uses linear search algorithms to find specific records based on user input.

3.2. Use of Data Structures

The system uses a variety of data structures, including arrays, maps, and objects. For example, the ApplicationManager class uses a Map to store different view managers, while the TableView class uses arrays to store and manipulate data before displaying it in the UI.

3.3. Operations on Data Structures

The system performs various operations on data structures, including creation, deletion, traversal, and sorting. For example, the createTable method in the TableView class creates a new table element and populates it with data, while the delete method in the DataManager class removes a record from the database.

4. Program Testing

4.1. Static Testing

Static testing was conducted by manually reviewing the code and using tools like ESLint (VS Code extension) to identify potential issues. The code was also reviewed by peers to ensure that it met the project requirements.

4.2. Dynamic Testing

Dynamic testing was conducted by running the application and providing various inputs to test its functionality. This included testing the creation, updating, and deletion of records, as well as generating reports on student enrollments and course participation.

4.3. Unit Testing

Unit testing was conducted to test individual components of the system, such as the DataManager class and its methods. The following table summarizes the unit tests conducted:

Test Case ID	Description	Input	Expected Output	Actual Output	Result
UT-001	Fetch all staff records	None	Array of staff records	Array of staff records	Pass
UT-002	Create a new staff record	Staff data	Success message	Success message	Pass
UT-003	Update a staff record	Staff ID and updated data	Success message	Success message	Pass
UT-004	Delete a staff record	Staff ID	Success message	Success message	Pass

4.4. Integration Testing

Integration testing was conducted to ensure that different components of the system work together as expected. This included testing the interaction between the frontend and backend, as well as the integration between different modules. The following table summarizes the integration tests conducted:

Test Case ID	Description	Input	Expected Output	Actual Output	Result
IT-001	Add a new student and enroll them in a course	Student and course data	Success message	Success message	Pass
IT-002	List all students enrolled in a course	Course ID	List of students	List of students	Pass
IT-003	List all courses a student has enrolled in	Student ID	List of courses	List of courses	Pass

4.5. Acceptance Testing

Acceptance testing was conducted to ensure that the system meets the requirements specified in the task brief. This included testing the system's ability to handle real-world scenarios, such as adding a large number of records. The system was also tested for usability, ensuring that it is easy to navigate and use.

5. Conclusion

The Course Administration System for Coatbank College has been successfully developed and tested. The system meets all the requirements outlined in the task brief and demonstrates the use of object-oriented programming constructs, algorithms, and data structures. The system is ready for deployment and use.