February 2025

Project Report

Course Administration System for Coatbank College (HA4D 35)

Stanislav Starishko

Contents

1. Definition of Software Requirements

1.1 Identifying Objects and Grouping Them into Classes

In this project, I identified several key objects that correspond to the entities outlined in the project brief. These objects were grouped into classes:

- **Staff** — responsible for storing data about college employees.
- **Student** — contains information about students.
- **Course** — describes the courses offered by the college.
- **Enrollment** — links students to the courses they are enrolled in.

On the frontend, I also created classes for managing data and displaying information:

- **DataManager** — the base class for handling data.
- **StaffManager**, **StudentManager**, **CourseManager**, **EnrollmentManager** — classes for managing specific entities.
- **TableView** — responsible for displaying data in tables.
- **DetailsView** — used to show details (e.g., a list of courses for a student or a list of students for a course).
- **ApiService** — a class for interacting with the backend via API.
- **UIUtils** — utilities for working with the user interface.

1.2 Object Interaction and Behaviour

The main interaction between objects happens through the API. For example:

- When an administrator adds a new student, **StudentManager** sends a request via **ApiService** to the backend, where the data is stored in MongoDB.
- When deleting a student or course, **ApplicationManager** first removes the record from the relevant collection and then deletes all related entries from **Enrollment**. This is crucial for maintaining data integrity.

1.3 Object Attributes

Each class has its own attributes that describe its state:

- **Staff**: idNumber, forename, surname, dob, gender.
- **Student**: idNumber, forename, surname, dob, gender.
- **Course**: courseId, courseName, courseStart, courseLength, courseLocation.
- **Enrollment**: studentId, courseId.

1.4 Defining Requirements

To describe the functional requirements, I used a use case diagram (see **Appendix 1 Diagrams, Use Case Diagram**). It shows how the administrator interacts with the system: managing staff, students, courses, and viewing details.

2. Design of Software Solutions

2.1 Functional Design Solution Using Object-Oriented Modelling Techniques

I created a class diagram (see **Appendix 1 Diagrams, Class Diagram**) that shows how the classes are interconnected. For example, **DataManager** serves as the base class for **StaffManager**, **StudentManager**, **CourseManager**, and **EnrollmentManager**. This avoids code duplication and simplifies maintenance.

2.2 Mapping Technology-Independent Concepts to Implementing Classes and Interfaces

Technology-independent concepts, such as "Student" or "Course", were implemented as classes on both the frontend and backend. For instance, the **Student** class on the backend corresponds to the **StudentManager** class on the frontend.

2.3 Implementation Constraints

During the design phase, I considered the following constraints:

- Use of MongoDB for data storage.
- Performance requirements: the system should operate without delays.
- Budget constraints: the project must be completed within the allocated budget.

3. Modelling Software Solutions

3.1 Modelling Dynamic Behaviour

To describe the dynamic behaviour, I created several sequence diagrams:

- **General Sequence Diagram** (see **Appendix 1 Diagrams, Sequence Diagram**) — shows how the administrator interacts with the system.
- **Sequence Diagram for Deleting a Student** (see **Appendix 1 Diagrams, Sequence Diagram for Deleting a Student**) — describes the process of deleting a student and related entries from **Enrollment**.
- **Sequence Diagram for Deleting a Course** (see **Appendix 1 Diagrams, Sequence Diagram for Deleting a Course**) — illustrates how a course and its related entries are deleted from **Enrollment**.

3.2 Modelling Static Structures

The static structure of the system is reflected in the class diagram (see **Appendix 1 Diagrams, Class Diagram**). It shows how the classes are interconnected and what attributes they contain.

3.3 Using Object-Oriented Modelling Language

In this project, I applied object-oriented principles such as inheritance, encapsulation, and polymorphism. For example, **DataManager** serves as the base class for all managers, enabling code reuse.

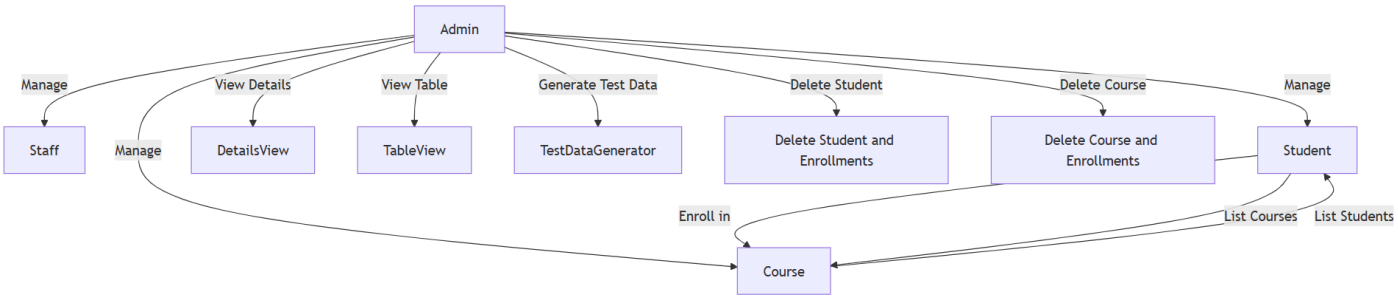
4. Conclusion

In this report, I have described how I developed the course administration system for Coatbank College. I used an object-oriented approach to create a flexible and scalable system. All requirements outlined in the project brief have been met, including the functionality for adding, deleting, and editing data, as well as displaying related records.

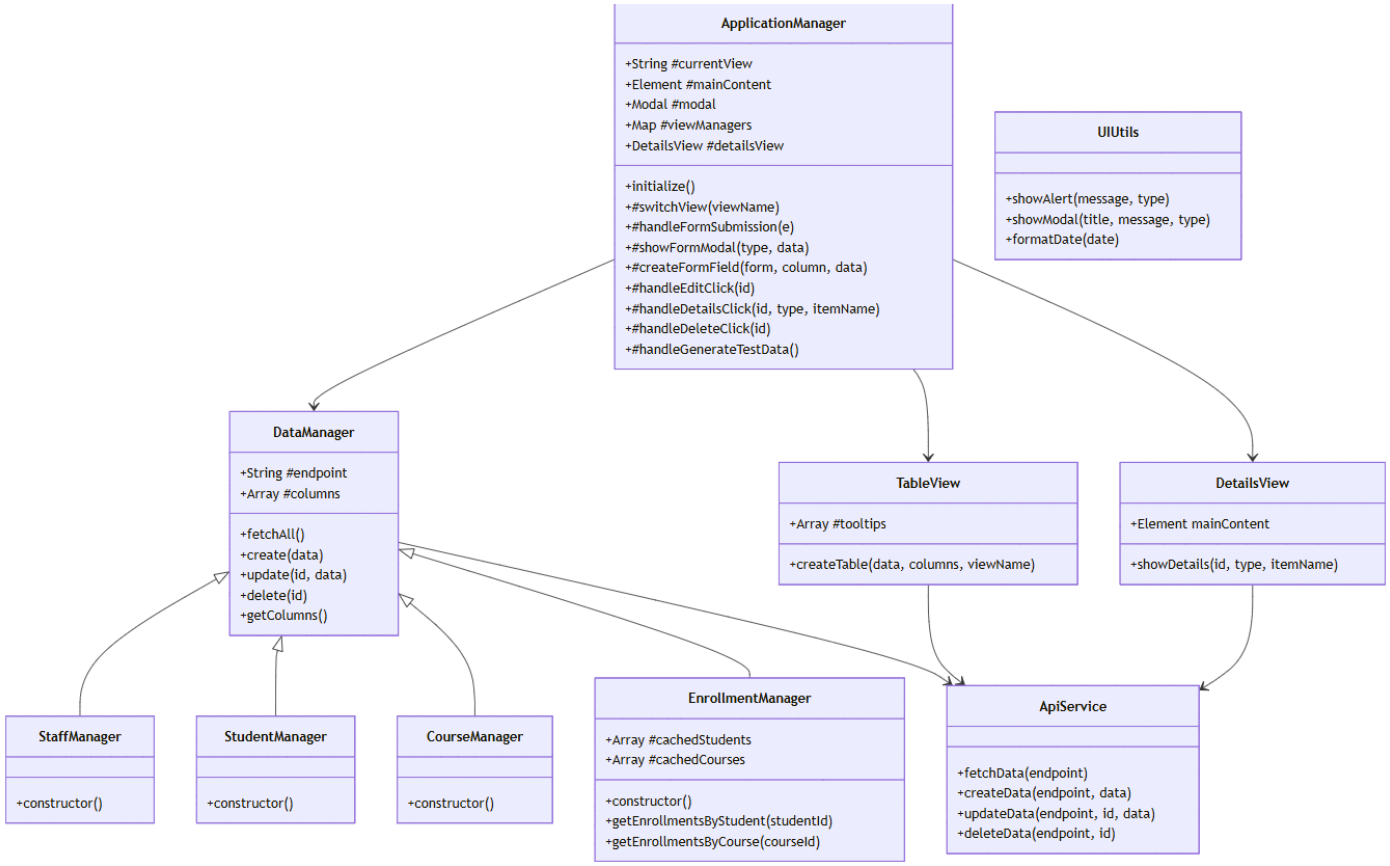
If needed, I am ready to make additional changes or provide further clarifications to the project.

5. Appendix 1 Diagrams

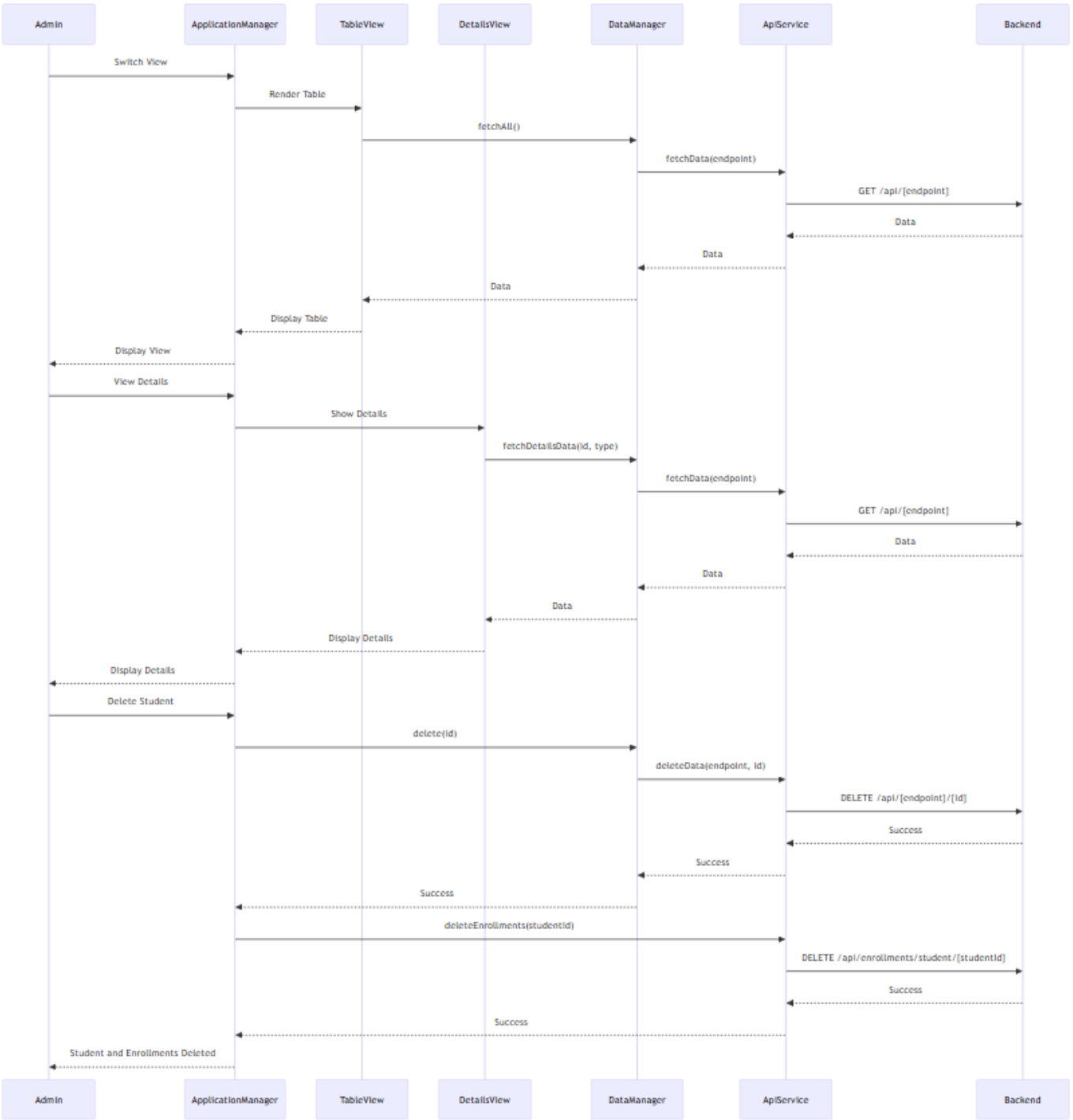
Use Case Diagram: illustrates the administrator's interaction with the system.



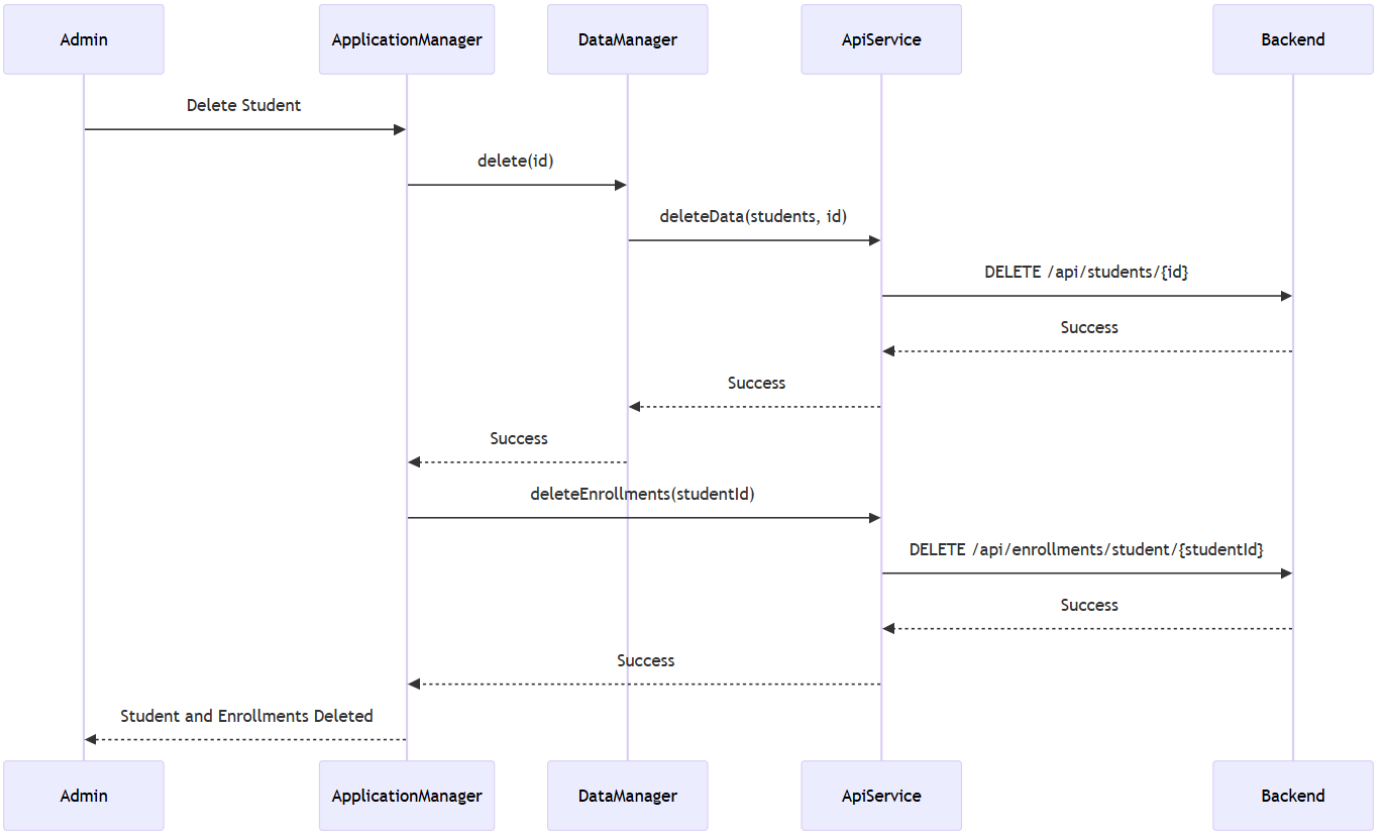
Class Diagram: displays the structure of classes and their relationships.



Sequence Diagram: a general sequence diagram for system interaction.



Sequence Diagram for Deleting a Student: the process of deleting a student and related entries.



Sequence Diagram for Deleting a Course: the process of deleting a course and related entries.