Assignment 3
Analysis and Design Document

Student: Cosma Felicia - Iulia
Group: 30431

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The client-server application for school or high school administration will be implemented using a combination of Observer and Strategy design patterns. Additionally, unit tests will be created to cover both the happy flow and edge scenarios.

The application will consist of a server component and multiple client components for teachers and students. The server will handle all the data storage and processing, while the clients will interact with the server to perform various operations.

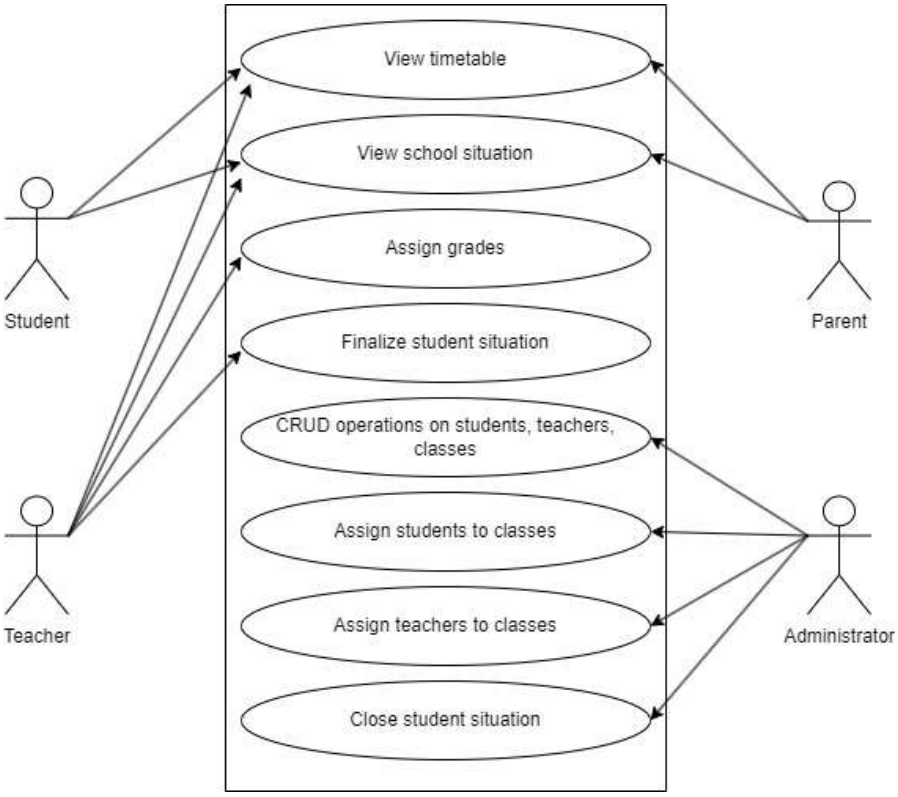
1.2 Functional Requirements

- User Registration and Login
- User Roles and Permissions
- Each user should have a specific role with corresponding permissions
- CRUD operations on students
- Email notification

1.3 Non-functional Requirements

- Security:
- The system ensures secure authentication and data access
- User passwords is securely stored using encryption techniques

2. Use-Case Model



Use case diagram for the whole project (for the assignment there are only Student and Admin)

Use Case: Manage Student Information
Level: User-goal level
Primary Actor: Administrator
Main Success Scenario:

- The system presents a list of existing students or an empty form to add a new student.
- If the administrator wants to add a new student:
  - The administrator fills in the required information such as student name, ID, and other relevant details.
  - The system validates the input and creates a new student record.
  - The system notifies the administrator about the successful addition of the student.
- If the administrator wants to view or update an existing student:
  - The administrator selects a student from the list.
  - The system retrieves and displays the student's information.
  - If the administrator wants to update the student's information:
    - The administrator modifies the relevant fields.
    - The system validates the changes and updates the student's record.
    - The system notifies the administrator about the successful update.
- If the administrator wants to delete a student:
  - The administrator selects a student from the list.
  - The system prompts for confirmation.
  - If confirmed, the system removes the student's record from the system.
  - The system notifies the administrator about the successful deletion.

If the administrator enters invalid or incomplete information when adding or updating a student: The system displays an error message and prompts the administrator to provide valid information.

If the system encounters an error while performing CRUD operations on student information: The system displays an error message and prompts the administrator to try again or contact technical support.

The "Manage Student Information" use case allows the administrator to perform CRUD operations (Create, Read, Update, Delete) on student records. It enables the administrator to add new students, view and modify existing student information, and delete students when necessary. This use case provides the necessary functionality for the administrator to maintain accurate and up-to-date student data within the system.

3. System Architectural Design

3.1 Architectural Pattern Description

- Layered Architecture:

The layered architecture is a pattern that organizes the application into logical layers, each responsible for a specific set of tasks. This pattern helps in achieving separation of concerns and modularity.

- Presentation Layer: This layer handles the user interface and user interaction. It includes

the client applications used by teachers, students, parents, and administrators.

- Business Logic Layer: This layer contains the core business logic of the application. It handles operations such as authentication, grade assignment, timetable management, and school situation calculations. It interacts with the data layer for accessing and manipulating data.
- Data Layer: This layer is responsible for data storage and retrieval. It can include a database management system or other forms of data storage. The data layer provides the necessary APIs and services to interact with the underlying data storage.

The layered architecture promotes separation of concerns, making the system more maintainable and testable. It allows for flexibility in making changes or additions to specific layers without affecting the other layers.

3.2 Diagrams

The system will be designed using a layered architecture, which is a common architectural style for client-server applications. This architecture separates the system into layers, with each layer having a specific responsibility and interacting only with the layer directly below or above it.

The layers of the system are as follows:

1. Presentation Layer: This layer is responsible for presenting information to the users and receiving input from them. It will be implemented using a web-based user interface, which will allow users to access the system from any device with an internet connection.
2. Application Layer: This layer will contain the business logic of the system, including the processing of requests from the users, the validation of input data, and the execution of database queries. It will be implemented using a RESTful API, which will allow the presentation layer to communicate with the application layer using HTTP requests and responses.
3. Persistence Layer: This layer will be responsible for storing and retrieving data from the database. It will be implemented using a relational database management system (RDBMS), which will provide a structured and scalable storage solution for the system.

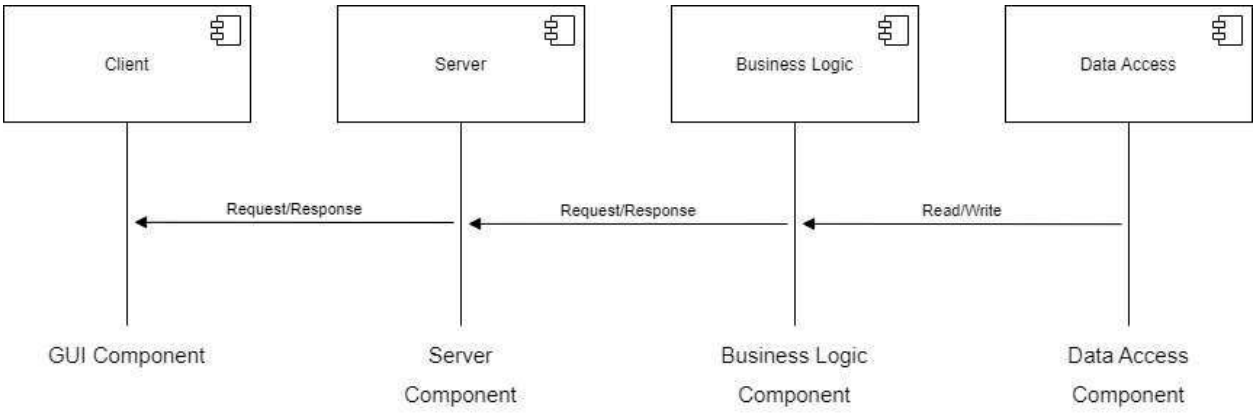
By using a layered architecture, we can ensure that the system is modular, scalable, and maintainable. Each layer can be developed and tested independently, allowing for easier debugging and refactoring. Additionally, the separation of concerns makes it easier to implement changes to one layer without affecting the others.

The package diagram is shown below:

The component and deployment diagrams:

The diagram shows the four components of the system: the Client, Server, Business Logic, and Data Access components. The Client component represents the graphical user interface used by teachers, students, and parents, which can also be accessed through a command-line interface by students. The Server component is responsible for handling requests from the clients and coordinating the business logic and data access components. The Business Logic component is responsible for implementing the rules and logic of the system, such as verifying user credentials and managing the assignment of grades to students. The Data Access component is responsible for managing the communication with the database, retrieving and storing data.

The GUI and Business Logic components communicate with each other through well-defined interfaces, as do the Business Logic and Data Access components. The communication between components is shown by the Request/Response and Read/Write arrows in the diagram. Finally, the deployment diagram will show how these components will be deployed on different servers or machines.

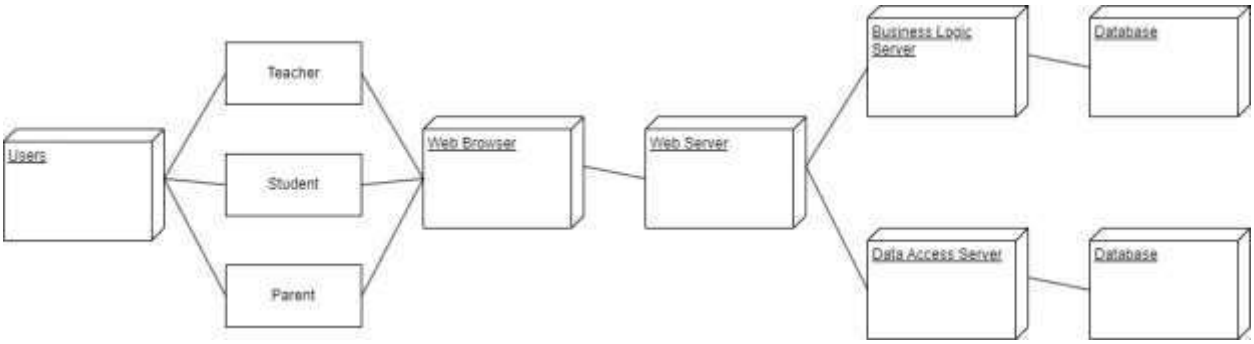


In this deployment diagram, the users access the system through a web browser. The web server component is responsible for serving the web pages to the users, and for handling their requests.

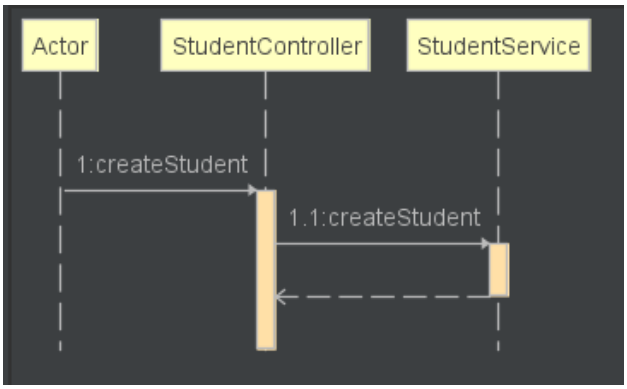
The business logic server component is responsible for implementing the system's business logic, and communicates with the data access server component to retrieve and store data in the databases.

The databases are represented as separate components, with one database for storing information related to users, classes, and timetables, and another database for storing grades and other academic records.

Overall, this deployment diagram shows how the various components of the system will be deployed on different servers or machines, and how they will communicate with each other to provide the necessary functionality to users.



4. UML Sequence Diagrams



Sequence diagram for createStudent method

5. Class Design

5.1 Design Patterns Description

Observer Pattern:

The Observer pattern is used to implement the notification system in the application. When important events occur, such as generating a billing report, the Observer pattern allows for automatic notification of subscribers (observers) who are interested in receiving updates about these events.

In the application, two observers are implemented as separate components: one that writes the event data to a log file and another that sends the event via email. The notification itself acts as the subject, which maintains a list of observers and notifies them when an event occurs. The Observer pattern provides a loosely coupled and extensible way to handle notifications, allowing for easy addition or removal of observers without affecting the subject or other observers.

Behavioral Pattern (e.g., Chain of Responsibility, Strategy, State, Command, Visitor, etc.):

A specific behavioral pattern from the list (Chain of Responsibility, Strategy, State, Command, Visitor, etc.) is chosen and implemented to handle complex business logic flows in the application.

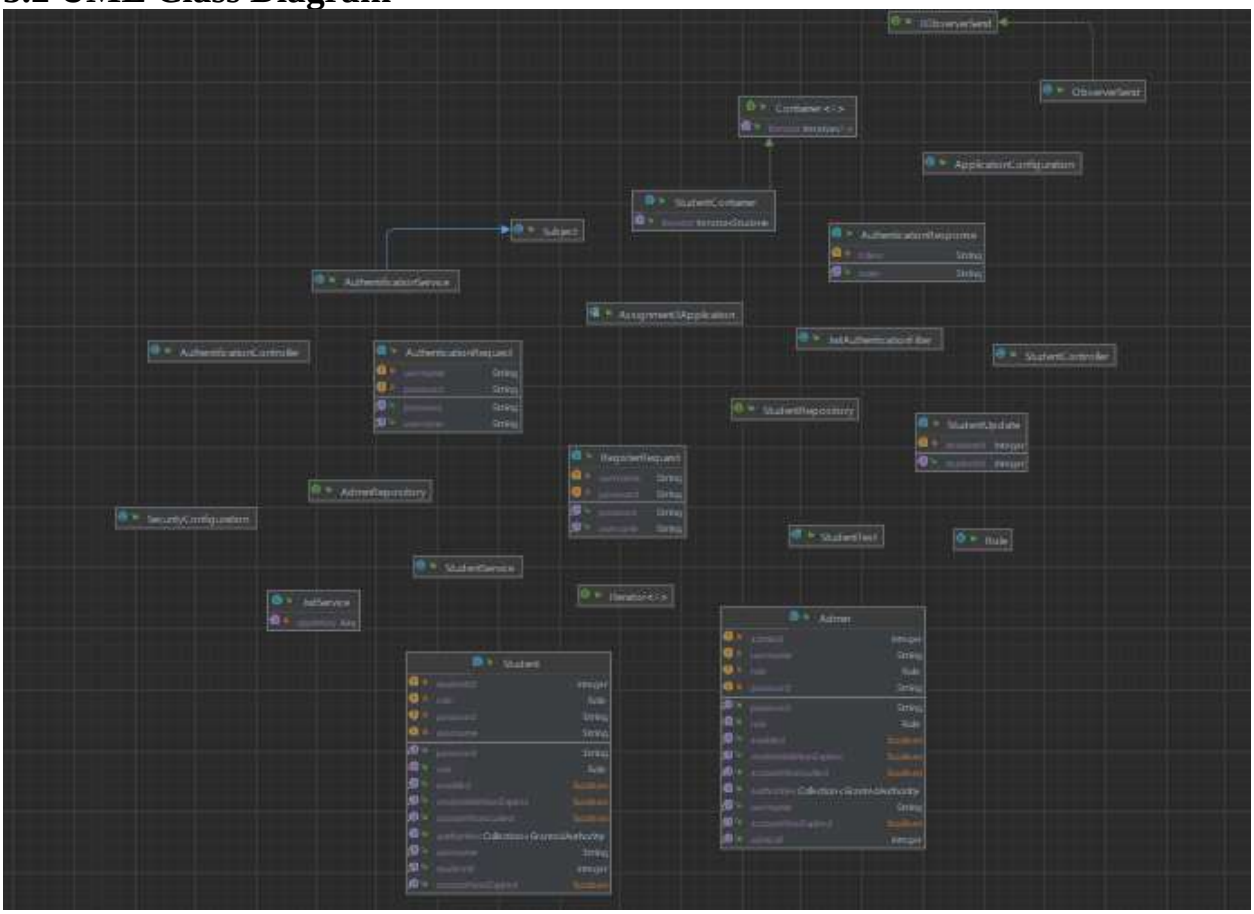
The choice of the specific behavioral pattern depends on the complexity of the business logic flow. For example, if there is a need to handle a sequence of operations or apply different strategies based on certain conditions, the Strategy pattern can be used. It allows for

encapsulating various algorithms or approaches as separate strategies, which can be dynamically selected and applied at runtime.

The chosen behavioral pattern enhances the flexibility, modularity, and maintainability of the application by separating the implementation of individual strategies from the overall logic flow. It allows for easy modification and extension of the business logic without impacting other parts of the system.

These design patterns, the Observer pattern, and a specific behavioral pattern, are utilized in the application to facilitate efficient notification handling and manage complex business logic flows in a scalable and maintainable manner.

5.2 UML Class Diagram



6. Data Model

- User:
    - Represents a user of the system.
    - Attributes:
    - User ID: Unique identifier for the user.
    - Username: The username used for login.
    - Password: The password for authentication.
    - Role: The role or type of user (e.g., teacher, student, parent, administrator).

7. System Testing

The testing is done by using JUnit and Mockito. The testing class focuses on testing the createStudent method of the StudentService class. Mockito is used to create a mock object for the StudentRepository dependency, allowing control over its behavior. The test verifies that when createStudent is called, the student object is saved in the repository and returned correctly. It employs the assertEquals method to validate the expected and actual results. The test follows a data-flow testing approach, ensuring the proper flow of data within the method under test.

8. Bibliography

- https://www.youtube.com/watch?v=KxqlJblhzfI&ab_channel=Amigoscode
- https://www.youtube.com/watch?v=-oLDJ2dbadA&ab_channel=Geekific
- https://www.youtube.com/watch?v=G3uNYHtn83c&ab_channel=AlexLee
- https://www.youtube.com/watch?v=8TEAycT5HfY&ab_channel=JRACADEMY
- https://www.youtube.com/watch?v=QwQuro7ekvc&ab_channel=Amigoscode