School Master
Analysis and Design Document

Student: Cosma Felicia-Iulia
Group:30431

Revision History

Date	Version	Description	Author
12/04/23	1.0		Cosma Felicia-Iulia

Table of Contents

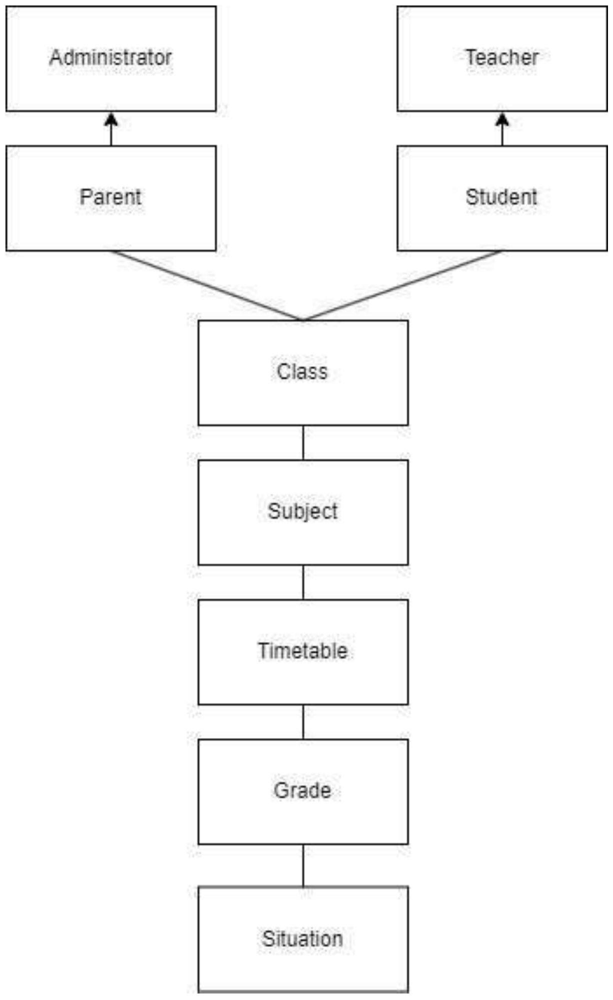
I. Project Specification

Design and implement a client-server application for school or high school administration which can be used by teachers, students, parents, and an administrator. The system should allow teachers to view their timetable, view the school situation of a student, assign grades to students, and finalize the students' situation at the subject they teach. Students should be able to view their timetable and school situation. Parents should be able to view their children's timetable and school situation. The administrator should be able to perform CRUD operations on students, teachers, and classes, assign students to classes, assign teachers to classes, and close the situation of a student at the end of a semester. The system should automatically send an email to a parent when their child receives a grade and when their situation is closed.

II. Elaboration – Iteration 1.1

1. Domain Model

The domain model consists of the following classes: Administrator, Teacher, Student, Parent, Class, Subject, Timetable, Grade, and Situation. The conceptual class diagram is shown below:



2. Architectural Design

2.1 Conceptual Architecture

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.

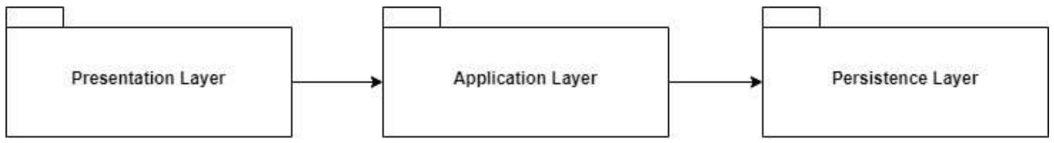
The layers of the system are as follows:

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.

2.2 Package Design

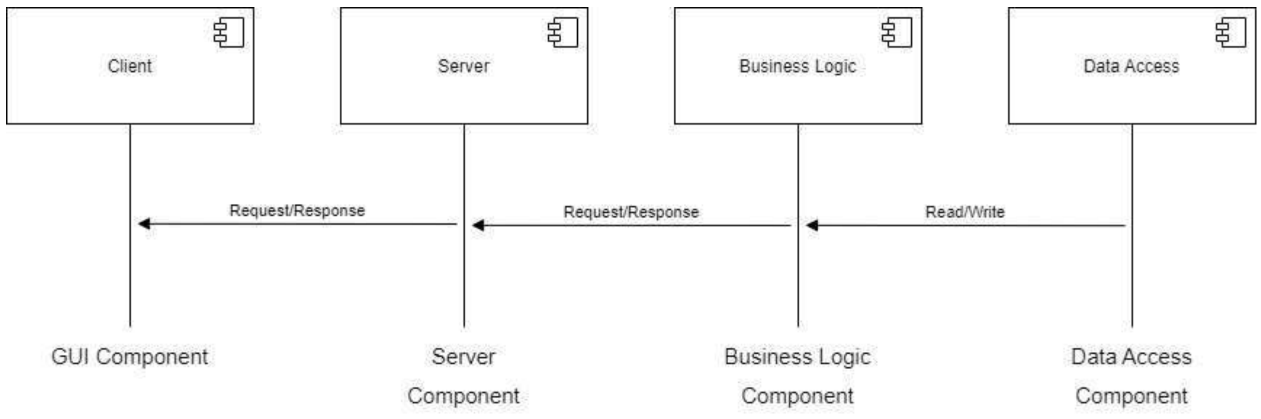
The package diagram is shown below:



2.3 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.

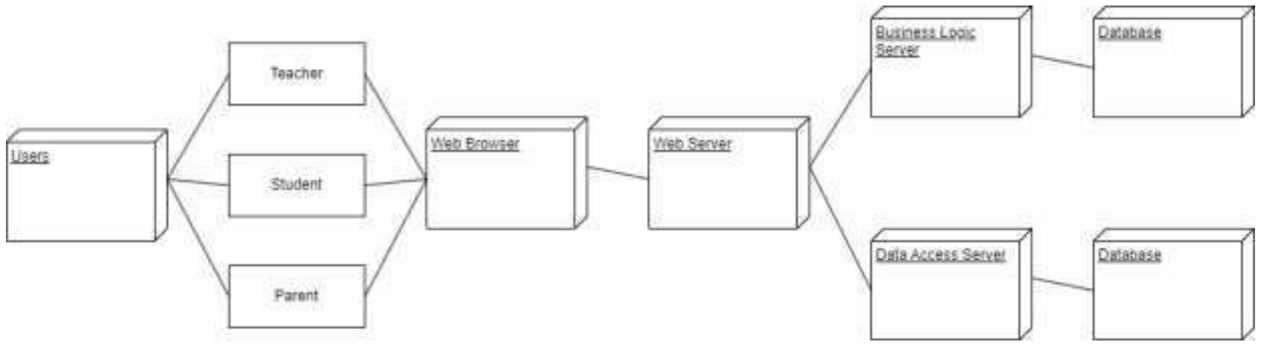


In this deployment diagram, the users access the system through a web browser. The web server component is responsible for serving the web pages to the users, and for handling their requests.

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.



III. Elaboration – Iteration 1.2

1. Design Model

1.1 Dynamic Behavior

Scenario 1: Teacher assigns grades to a student
Sequence Diagram:

```
Teacher              Server                  Student
  |                      |                        |
  |   Sends request to  |                        |
  |    assign grade      |                        |
  |---------------------->|                        |
  |                      |   Notifies teacher     |
  |                      |  of student's grades  |
  |                      |<-----------------------|
  |   Receives grades   |                        |
  |<----------------------|                        |
```

Communication Diagram:

```
    Teacher                  Server                  Student
       |                          |                          |
       |  Assign grade to         |                          |
       |  student with ID 123  |                          |
       |------------------------>|                          |
       |                          | Find student with ID  |
       |                          | 123                      |
       |                          |-------------------------->|
       |                          |   Sends student data     |
       |                          |<--------------------------|
       |                          |    Notifies student of    |
       |                          |        new grade          |
       | Sends email to parent  |                          |
       | of student with grade  |                          |
       |-------------------------->|                          |
       |                          |                          |
```

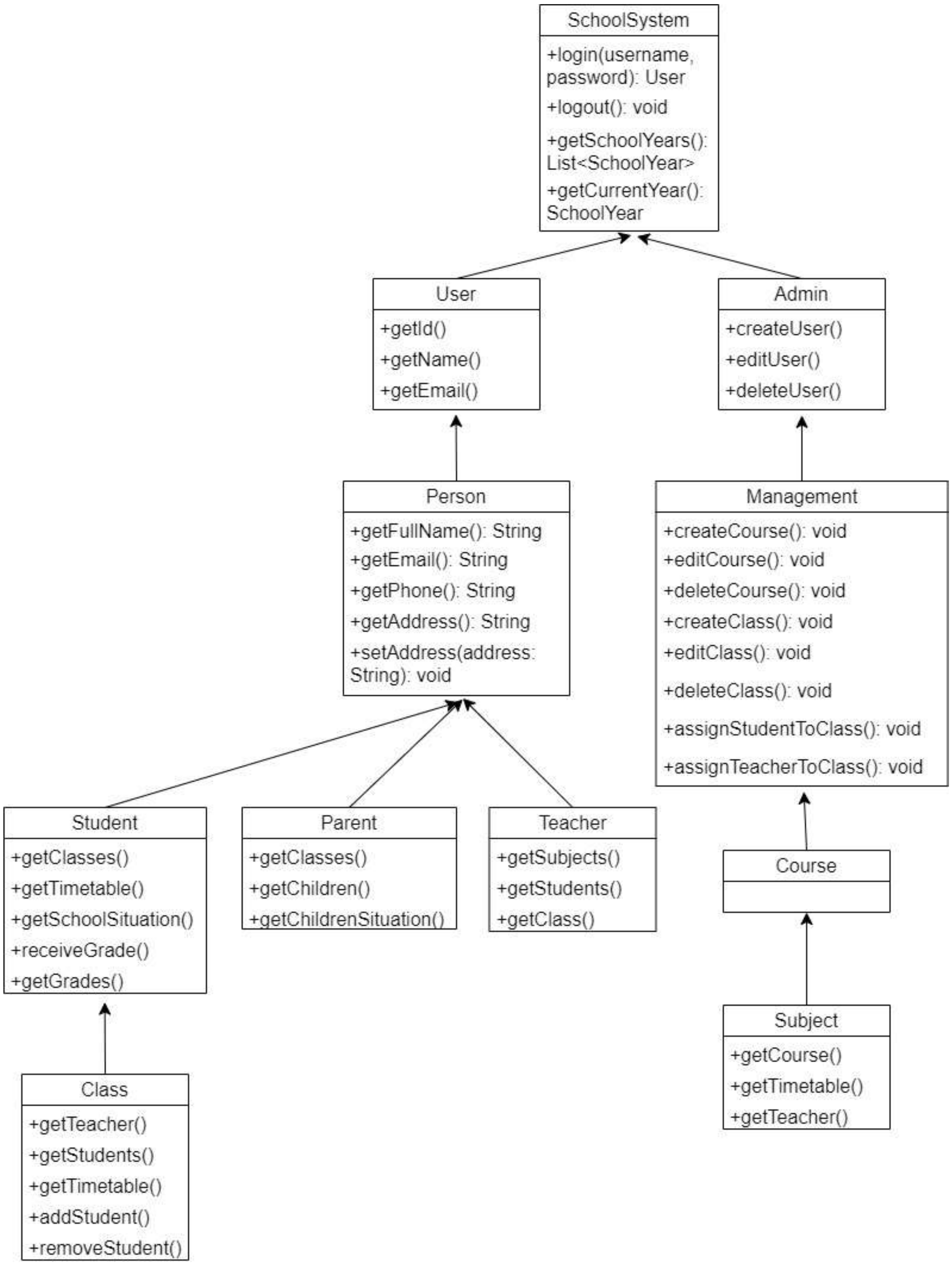
Scenario 2: Administrator adds a new student to a class
Sequence Diagram:

```
Administrator        Server                    Class
  |                     |                          |
  |   Sends request to  |                          |
  |    add new student  |                          |
  |-------------------->|                          |
  |                     |   Notifies admin         |
  |                     |      of available        |
  |                     |      classes             |
  |                     |<-------------------------|
  |  Selects class to add |                        |
  |-------------------->|                          |
  |                     |  Finds selected class    |
  |                     |  and sends class data    |
  |                     |------------------------->|
  |                     | Notifies admin of new    |
  |                     | student in the class     |
  |                     |<-------------------------|
  |  Receives response  |                          |
  |<--------------------|                          |
```

Communication Diagram:

```
Administrator              Server                    Class
      |                         |                         |
      | Add new student to   |                         |
      | class with ID 456      |                         |
      |---------------------------->|                         |
      |                          | Find available          |
      |                          | classes and sends    |
      |                          | class data               |
      |                          |<--------------------------|
      | Selects class with ID  |                          |
      | 123                     |                          |
      |---------------------------->|                          |
      |                          | Find selected class    |
      |                          | and sends class data |
      |                          |--------------------------->|
      |                          | Notifies class of new |
      |                          | student in the class  |
      |                          |<---------------------------|
      | Receives response     |                           |
      |<----------------------------|                           |
```

1.2 Class Design



SchoolSystem
+login(username, password): User
+logout(): void
+getSchoolYears(): List<SchoolYear>
+getCurrentYear(): SchoolYear
User
+getId()
+getName()
+getEmail()
Admin
+createUser()
+editUser()
+deleteUser()
Person
+getFullName(): String
+getEmail(): String
+getPhone(): String
+getAddress(): String
+setAddress(address: String): void
Management
+createCourse(): void
+editCourse(): void
+deleteCourse(): void
+createClass(): void
+editClass(): void
+deleteClass(): void
+assignStudentToClass(): void
+assignTeacherToClass(): void
Student
+getClasses()
+getTimetable()
+getSchoolSituation()
+receiveGrade()
+getGrades()
Parent
+getClasses()
+getChildren()
+getChildrenSituation()
Teacher
+getSubjects()
+getStudents()
+getClass()
Course
Class
+getTeacher()
+getStudents()
+getTimetable()
+addStudent()
+removeStudent()
Subject
+getCourse()
+getTimetable()
+getTeacher()

2. Data Model

(I didn't start my project yet so I didn't do my database, I will add it later, temporarily I wrote this:)

Entities:

User (id, name, email, password, role)
Teacher (id, user_id)
Student (id, user_id)
Parent (id, user_id)
Administrator (id, user_id)
Class (id, name)
Subject (id, name)
Grade (id, value, student_id, teacher_id, subject_id)
Semester (id, name)
Enrollment (id, student_id, class_id, semester_id)
Schedule (id, day, start_time, end_time, class_id, teacher_id, subject_id)

Relationships:

Teacher has one User (1:1)
Student has one User (1:1)
Parent has one User (1:1)
Administrator has one User (1:1)
Teacher teaches many Subjects (M:N)
Subject is taught by many Teachers (M:N)
Teacher assigns many Grades (1:N)
Student receives many Grades (1:N)
Semester has many Enrollments (1:N)
Class has many Enrollments (1:N)
Student is enrolled in many Classes (M:N)
Class has many Schedules (1:N)
Teacher has many Schedules (1:N)

Notes:

The User entity holds the common attributes for all types of users, including their role (teacher, student, parent, or administrator).
Each user has a one-to-one relationship with their corresponding entity (Teacher, Student, Parent, or Administrator).
A Teacher can teach many Subjects, and a Subject can be taught by many Teachers, hence forming a many-to-many relationship that requires a junction table.
A Teacher assigns Grades to many Students, and a Student receives Grades from many Teachers. This relationship is modeled as a one-to-many relationship with a foreign key in the Grades entity.
A Semester has many Enrollments, and an Enrollment belongs to one Semester. Similarly, a Class has many Enrollments, and an Enrollment belongs to one Class. This relationship is modeled as a one-to-many relationship with a foreign key in the Enrollment entity.
A Student can be enrolled in many Classes, and a Class can have many Students. This relationship is modeled as a many-to-many relationship that requires a junction table.
A Class has many Schedules, and a Teacher has many Schedules. This relationship is modeled as a one-to-many relationship with a foreign key in the Schedule entity.

3. Unit Testing

As with any software application, testing is an important part of the development process. The following are some testing methods that can be used to ensure the functionality and reliability of the school administration application:

1. Unit Testing: This involves testing each individual component of the system in isolation. For example, testing the functionality of a single module such as assigning grades to students or finalizing a student's situation at the subject they teach.

Test case scenario:

- Test case name: Test grade assignment functionality
- Test case description: Verify that the teacher can successfully assign a grade to a student and that the grade is accurately recorded in the system.
- Test steps:
  - Create a new student and assign them to a class.
  - Create a new subject and assign it to the teacher.
  - The teacher assigns a grade to the student for the subject.
  - Verify that the grade is recorded in the system and associated with the correct student and subject.

2. Integration Testing: This involves testing the interactions between different components of the system. For example, testing the interaction between the grade assignment functionality and the email notification system.

Test case scenario:

- Test case name: Test grade assignment and email notification functionality
- Test case description: Verify that the system sends an email to the parent when their child receives a grade.
- Test steps:
  - Create a new student and assign them to a class.
  - Create a new subject and assign it to the teacher.
  - The teacher assigns a grade to the student for the subject.
  - Verify that the email notification system sends an email to the parent associated with the student.
  - Verify that the email includes the student's name, the subject, and the assigned grade.

3. System Testing: This involves testing the system as a whole to ensure that all components work together properly. For example, testing the overall functionality of the application from the perspective of different users.

Test case scenario:

- Test case name: Test overall functionality of the application
- Test case description: Verify that all users can perform the intended operations and that the system is responsive and user-friendly.
- Test steps:
    - Login as a teacher and verify that they can view their timetable, view the school situation of a student, assign grades to students, and finalize the students' situation at the subject they teach.
    - Login as a student and verify that they can view their timetable and view their school situation.
    - Login as a parent and verify that they can view their children's timetable and view their children's school situation.
    - Login as an administrator and verify that they can perform CRUD operations on students, teachers, and classes, assign students to classes, assign teachers to classes, and close the situation of a student at the end of a semester.
    - Verify that the system is responsive and user-friendly by performing various actions and observing the system's behavior and performance.

Overall, testing should be an ongoing process throughout the development of the application to ensure that any issues or bugs are identified and addressed in a timely manner.

IV. Elaboration – Iteration 2

1. Architectural Design Refinement

[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]

2. Design Model Refinement

[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]

V. Construction and Transition

1. System Testing

[Describe how you applied integration testing and present the associated test case scenarios.]

2. Future improvements

[Present future improvements for the system]

VI. Bibliography