University Management System Report

Introduction

The University Management System is a comprehensive software application designed to streamline and manage administrative and academic processes within a university or educational institution. This report provides an in-depth overview of the design, architecture, and functionality of the University Management System, offering insights into the choices made during its development and how it operates.

Purpose and Objectives

The primary purpose of the University Management System is to facilitate the efficient management of student and teacher data, subject information, attendance records, and grade calculations. The system aims to automate and simplify administrative tasks while providing a user-friendly interface for students, teachers, and administrators.

Key objectives of the project include:

1. Efficiently manage student and teacher data, including personal information and course-related details.
2. Track student attendance and generate attendance reports.
3. Record and calculate subject grades and provide comprehensive academic performance insights.
4. Implement a notification service for sending alerts to students and teachers.
5. Foster adaptability by providing multiple grade calculation strategies.

Scope and Intended Audience

The University Management System is intended for use within educational institutions, particularly universities and colleges. It caters to the needs of students, teachers, and administrators by offering a range of features aimed at enhancing academic management and communication.

The report primarily targets individuals involved in the development, management, and maintenance of the University Management System. This includes software developers, project managers, system administrators, and anyone interested in understanding the architecture and functionality of the application.

Report Outline

This report is structured as follows:

1. **System Architecture:** An overview of the overall architecture of the application, including how different components interact.

2. **Design Choices and Motivations:** An exploration of the design choices made for major components of the system and the motivations behind those choices.
3. **Application Execution Flow:** A detailed description of how the application runs, from launch to the execution of various actions.
4. **Database and Data Handling:** An explanation of data storage and retrieval methods, including data structures and serialization techniques.
5. **User Interaction:** Details about how users interact with the application, input validation, and error handling.
6. **Notification Service and Grade Calculation:** Insights into the implementation of notification services and grade calculation, and how these services are integrated with the core application.
7. **Challenges and Future Enhancements:** Discussion of any challenges encountered during development and potential future enhancements.
8. **Conclusion:** A summary of key findings and the significance of the University Management System.
9. **Appendices:** Supplementary materials, including code snippets and diagrams.

Design Choices and Motivations

System Architecture

Monolithic vs. Microservices

One of the initial design considerations was choosing between a monolithic architecture and a microservices architecture. We opted for a monolithic architecture due to its simplicity, as our application's initial scope did not require the complexity of microservices. A monolithic architecture allowed us to maintain a single codebase, simplifying development and deployment.

Object-Oriented Programming (OOP)

We embraced object-oriented programming (OOP) principles to achieve modularity and code reusability. This approach allowed us to model the system with classes, creating a clear structure for students, teachers, subjects, and other entities. The use of inheritance facilitated the creation of specialized student classes like `ComputerScienceStudent` and `AccountingStudent`, which inherit attributes and methods from the base `Student` class.

Java as the Programming Language

Java was chosen as the primary programming language for its portability, robustness, and extensive standard libraries. Leveraging Java allowed us to create a cross-platform application that can be executed on various operating systems without modification.

Database Management

Serialization for Data Storage

We decided to use Java's serialization mechanism for data storage. This decision simplifies the process of saving and loading student, teacher, and subject data. Serialization also ensures data consistency and integrity, allowing us to store and retrieve data seamlessly.

User Interface

Command-Line Interface (CLI)

To keep the application lightweight and user-friendly, we designed a command-line interface (CLI). While it may lack the graphical appeal of a graphical user interface (GUI), the CLI offers a more efficient and responsive interface for administrative tasks. This choice aligns with our objective of keeping the system simple and accessible.

Grade Calculation

Extensible Grade Calculation

We implemented an abstract class, `GradeCalculator`, to provide a structured approach to grade calculation. By doing so, we allow for extensibility, enabling different strategies for calculating grades. The `WeightedGradeCalculator` class is an example of a specific implementation that calculates weighted averages based on subject coefficients. This extensible design ensures adaptability to diverse grading systems.

Notification Service

Notification Service Interface

To foster adaptability, we introduced a `NotificationService` interface that defines methods for sending notifications to students and teachers. This abstraction allows us to implement different notification methods such as email, SMS, or other communication channels in the future.

Application Execution Flow

System Initialization

Upon launching the University Management System, the application performs a series of initialization steps to prepare for user interactions. These steps include loading data from serialized files, setting up data structures, and establishing the core components.

Data Loading

The application begins by loading pre-existing student, teacher, and subject data from serialized files. This data includes student profiles, teacher profiles, and subject details. Loading data at the start ensures that the system operates with up-to-date information.

User Authentication

Before granting access to the system, users must authenticate themselves. We maintain separate authentication mechanisms for administrators, students, and teachers. After successful authentication, users are granted the corresponding privileges and access rights.

Main Menu

The core interaction with the University Management System is through a text-based menu-driven interface. The main menu provides users with a list of available actions and options, including the following:

1. Student Management: This option allows administrators to manage student profiles, including the addition of new students and modification of existing profiles.
2. Teacher Management: Administrators can also manage teacher profiles, providing the ability to add, edit, or remove teachers.
3. Subject Management: Subject details, including course codes, coefficients, and assigned teachers, can be managed through this option.
4. Mark Attendance: Faculty members can use this option to mark student attendance, recording the date and whether the student was present.
5. Add Marks: Teachers input students' marks for specific subjects, including the grading coefficient.
6. Student Reports: Students can access their attendance records and grades for each subject.
7. Teacher Reports: Teachers can generate reports containing a list of their assigned subjects and related students.
8. Exit: Users can exit the application through this option.

Modular Workflow

Student Management

- **Add Student**: Administrators are prompted to input essential details of the student, including their name, student ID, email, password, age, date of birth, and specialization (if applicable).
- **Edit Student**: The system allows administrators to modify student profiles, updating information such as name, email, or specialization.
- **Remove Student**: If necessary, administrators can remove students from the system, which will result in the permanent deletion of student records.

Teacher Management

- **Add Teacher**: Administrators add teacher profiles, including personal information, age, email, password, phone number, and the subject they will teach.
- **Edit Teacher**: Any changes to teacher profiles can be made via this option.
- **Remove Teacher**: If a teacher leaves or if changes are required, administrators can remove teacher profiles.

Subject Management

- **Add Subject**: Course codes, coefficients, and assigned teachers are input to create subject profiles.
- **Edit Subject**: Any adjustments to subject information are performed here.
- **Remove Subject**: When subjects are discontinued or require removal, this option facilitates the process.

Attendance and Mark Entry

- **Mark Attendance**: Teachers can record attendance, specifying whether a student was present or absent for a particular day.
- **Add Marks**: Teachers input students' marks for each subject, including the grading coefficient.

Reports

- **Student Reports**: Students can generate and view their attendance records and grades for each subject.
- **Teacher Reports**: Teachers can access lists of their assigned subjects and related student profiles.

Notifications

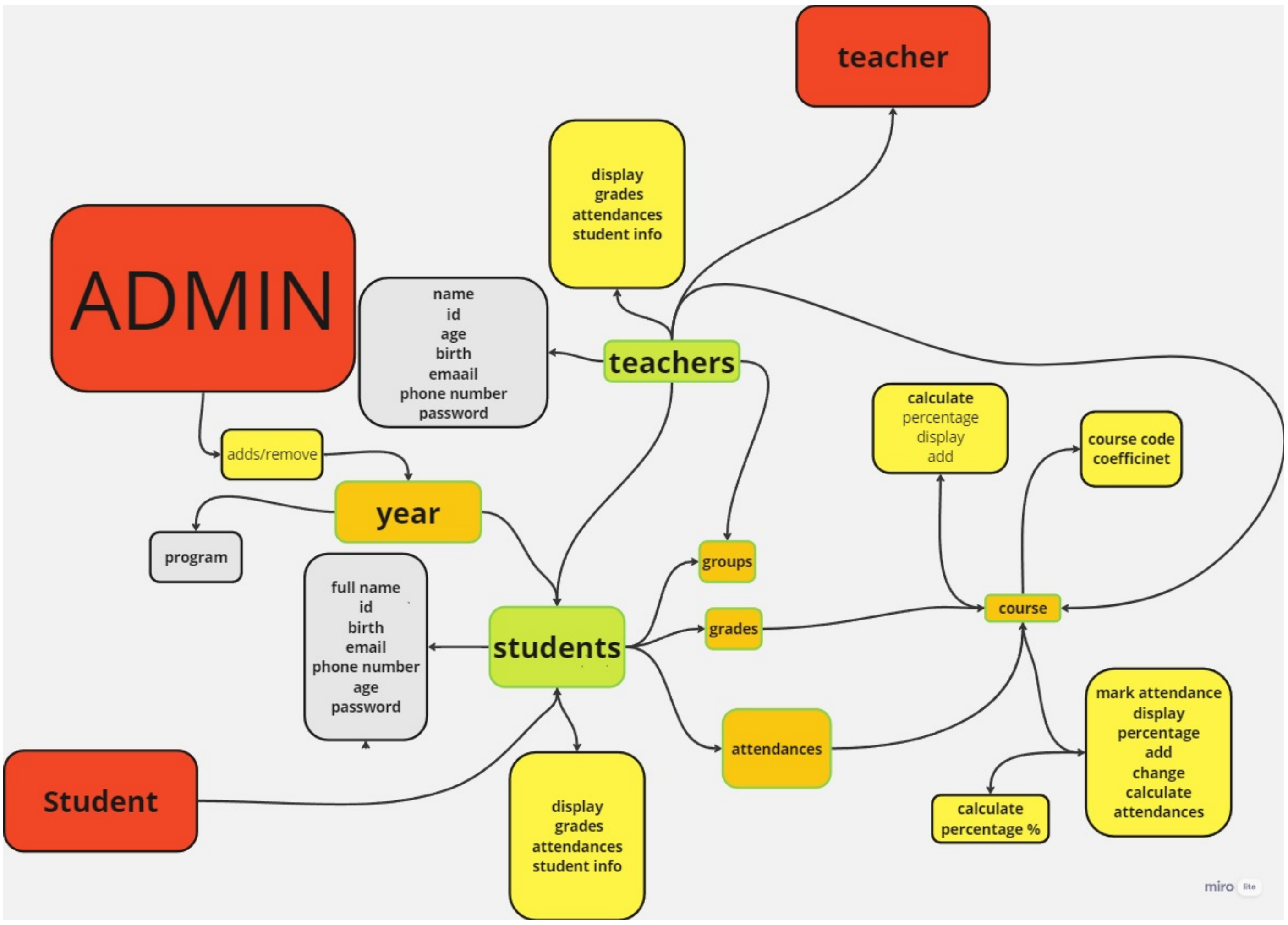
The system includes a `NotificationService` interface, allowing for future implementation of notification methods. Although not yet implemented, this feature can be expanded to include email, SMS, or other forms of communication for sending notifications to students and teachers.

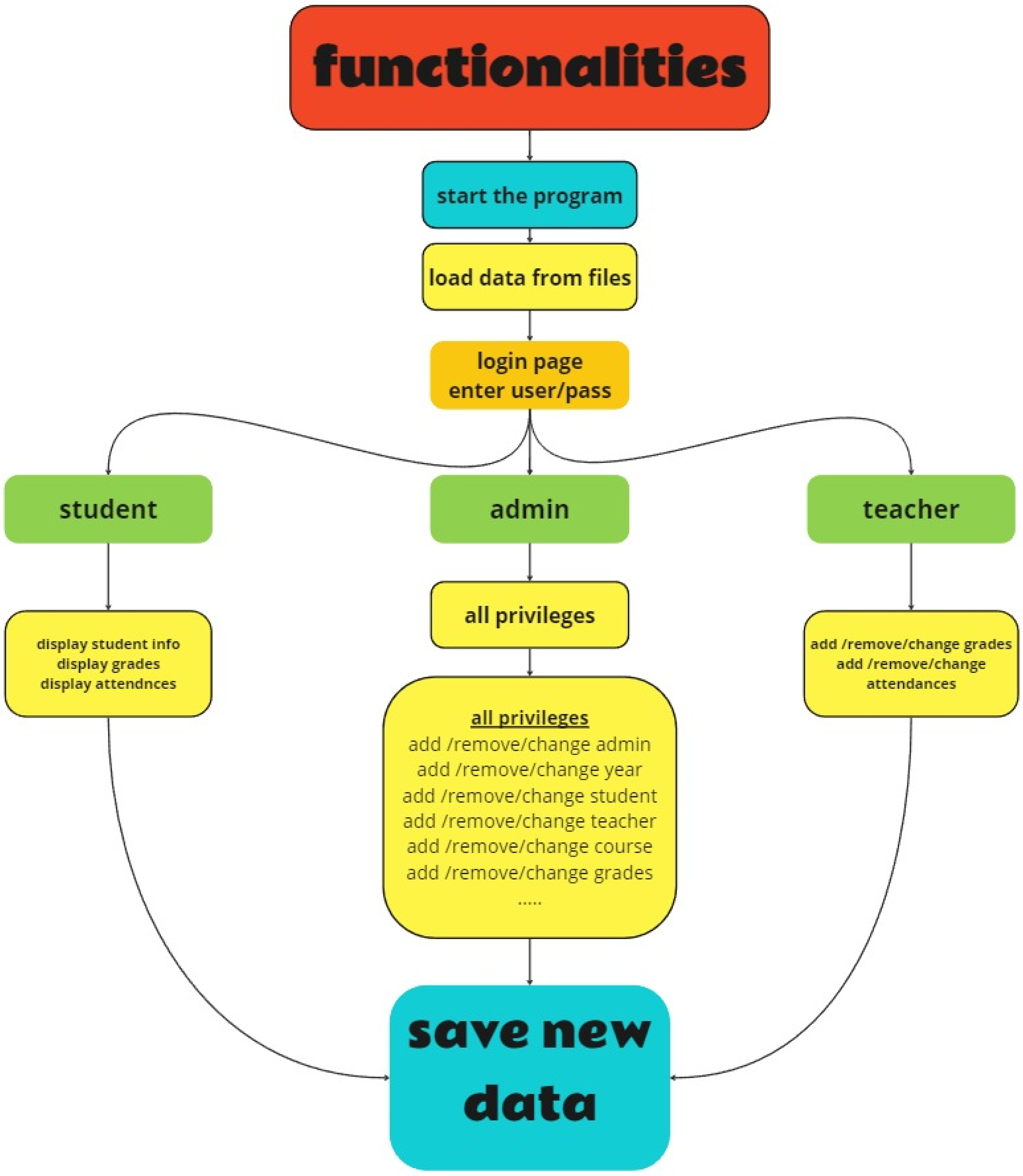
Grade Calculation

The `GradeCalculator` abstract class provides a structured approach to grade calculation. A specific implementation, such as the `WeightedGradeCalculator`, calculates weighted averages based on subject coefficients and student marks.

System Exit

Users can exit the application at any time, which prompts the system to save the latest data to serialized files, ensuring that changes and updates are preserved for the next use of the system.


teacher
display
grades
attendances
student info
ADMIN
name
id
age
birth
emaail
phone number
password
teachers
calculate
percentage
display
add
course code
coefficinet
adds/remove
year
program
groups
full name
id
birth
email
phone number
age
password
students
grades
course
attendances
mark attendance
display
percentage
add
change
calculate
attendances
calculate
percentage %
Student
display
grades
attendances
student info
miro lite


functionalities
start the program
load data from files
login page
enter user/pass
student
admin
teacher
display student info
display grades
display attendnces
all privileges
all privileges
add /remove/change admin
add /remove/change year
add /remove/change student
add /remove/change teacher
add /remove/change course
add /remove/change grades
.....
add /remove/change grades
add /remove/change
attendances
save new data