School Management System

Team no: 65
Team members:
CS23I1014 Bhadresh L
CS23I1065 Santhana Srinivasan R
CS23B1074 JP Akshaya

1 System Overview

The School Management System is a C++ application designed to streamline and enhance the administrative tasks of educational institutions. The system provides a user-friendly interface for three primary user roles:

- **Principal:** Responsible for overseeing school operations and managing staff and student records.
- **Teachers:** Manage classroom activities, student assessments, and attendance tracking.
- **Students:** Access their academic records, attendance, and course materials.

The system employs JSON files for data persistence, allowing for efficient storage and retrieval of information while maintaining a role-based access control system to ensure data security and privacy.

2 Data Storage

The system utilizes separate JSON files for different types of data, facilitating easy management and access:

- `principals.json`: Stores principal information, including credentials and profiles.
- `teachers.json`: Stores teacher information, including subjects taught and schedules.
- `students.json`: Stores student information, including grades, attendance records, and personal details.
- `courses.json`: Stores course details, including course codes, names, and enrolled students.

This structured data storage approach not only enhances the efficiency of data retrieval but also simplifies the process of data manipulation, making it easier to implement CRUD (Create, Read, Update, Delete) operations.

3 Class Structure

3.1 Base Class: User

An abstract base class that provides common functionality for all user types, ensuring a consistent interface and promoting code reuse.

Protected Members:

- `id: string` - Unique identifier for the user, essential for distinguishing between different users within the system.
- `name: string` - User's full name, used for personalizing interactions within the system.

Public Methods:

- `User (const string &userId, const string &userName):` Constructor that initializes user ID and name.
- `virtual void viewProfile() const = 0:` Pure virtual method for viewing user profile, which must be implemented by derived classes.
- `string getId() const:` Returns user ID, allowing for easy access to user identification.
- `string getName() const:` Returns user name, facilitating personalized user experiences.
- `User ():` Virtual destructor to ensure proper cleanup of derived class resources.

3.2 Principal Class

Inherits from User and manages administrative functions, providing a comprehensive set of tools for school management.

Public Methods:

- `Principal(string &userId, const string &userName):` Constructor that initializes principal with ID and name.
- `void viewProfile() const:` Displays principal's profile information, including contact details and administrative responsibilities.

Student Management:

- `void createStudent(const string &id, const string &name)`: Creates a new student record, ensuring that all necessary information is collected.
- `void retrieveStudent(const string &id)`: Retrieves student information based on ID, allowing for quick access to individual records.
- `void updateStudent(const string &id)`: Updates existing student record, enabling changes to personal details or academic performance.
- `void deleteStudent(const string &id)`: Removes student record from the system, ensuring data integrity and compliance with privacy regulations.
- `void viewAllStudents() const`: Displays all student records, providing a comprehensive overview of the student body.

Teacher Management:

- `void createTeacher(const string &id, const string &name)`: Creates a new teacher record, ensuring that all relevant information is captured.
- `void retrieveTeacher(const string &id)`: Retrieves teacher information based on ID, facilitating easy access to staff records.
- `void updateTeacher(const string &id)`: Updates existing teacher record, allowing for modifications to personal details or teaching assignments.
- `void deleteTeacher(const string &id)`: Removes teacher record from the system, maintaining data accuracy and compliance with regulations.
- `void viewAllTeachers() const`: Displays all teacher records, providing an overview of the teaching staff.

Course Management:

- `void createCourse(const string &id, const string &name)`: Creates a new course, ensuring that all necessary details are recorded.
- `void retrieveCourse(const string &id)`: Retrieves course information based on ID, allowing for quick access to course details.
- `void updateCourse(const string &id)`: Updates existing course information, enabling changes to course content or structure.
- `void deleteCourse(const string &id)`: Removes course from the system, ensuring that outdated or irrelevant courses are not displayed.
- `void viewAllCourses() const`: Displays all courses offered by the institution, providing a comprehensive overview of the curriculum.

3.3 Teacher Class

Inherits from User and manages teaching-related functions, providing tools for classroom management and student assessment.

Private Members:

- `Course* courses:` Pointer to an array of courses taught by the teacher, allowing for dynamic management of course assignments.

Public Methods:

- `Teacher(const string &userId, const string &userName, Course* c):` Constructor that initializes teacher with ID, name, and assigned courses.

- `Teacher():` Destructor to clean up resources associated with the teacher object.

- `void viewProfile() const:` Displays teacher's profile, including subjects taught and contact information.

- `void updateAttendance(const string &studentId, const string &courseCode):` Updates attendance records for students, ensuring accurate tracking of student participation.

- `void updateGrade(const string &studentId, const string &courseCode):` Updates student grades, facilitating timely feedback on academic performance.

3.4 Student Class

Inherits from User and provides student-specific functionality, enabling students to manage their academic records.

Private Members:

- `StudentStats* stats:` Pointer to an array of student's academic statistics, allowing for dynamic management of performance data.

Public Methods:

- `Student(const string &userId, const string &userName, StudentStats*):` Constructor that initializes student with ID, name, and academic statistics.

- `Student():` Destructor to clean up resources associated with the student object.

- `void viewProfile() const:` Displays student's profile, including personal details and academic performance.

- `void viewAttendances() const:` Displays student's attendance records, providing insights into participation.

- `void viewGrades() const`: Displays student's grades, allowing for self-assessment and tracking of academic progress.

4 Supporting Structures

4.1 Course Structure

```
struct Course {
    string code;
    string name;
};
```

4.2 StudentStats Structure

```
struct StudentStats {
    string coursecode;
    int grade;
    float attendance;
};
```

5 Memory Management

Dynamic memory is utilized for Course and StudentStats arrays, ensuring efficient use of resources. Proper cleanup is handled in respective destructors to prevent memory leaks and ensure system stability.

6 Menu System

6.1 Principal Menu

1. View Profile
2. Manage Students
3. Manage Teachers
4. Manage Courses
5. Logout

6.2 Teacher Menu

1. View Profile
2. Update Student Grade
3. Update Student Attendance

4. Logout

6.3 Student Menu

1. View Profile
2. View Grades
3. View Attendance
4. Logout

7 Concepts Used

The system is built upon several key object-oriented programming concepts, which enhance its functionality and maintainability:

- **Abstraction:** Hides complex implementation details and exposes only the necessary parts of the system to the user.
- **Encapsulation:** Bundles data and methods that operate on the data within a single unit, restricting direct access to some of the object's components.
- **Operator Overloading:** Allows custom definitions for operators, enhancing the readability and usability of the code.
- **Inheritance:** Facilitates code reuse by allowing new classes to inherit properties and methods from existing classes.
- **Polymorphism:** Enables methods to do different things based on the object it is acting upon, allowing for flexible and dynamic code.
- **Exception Handling:** Provides a mechanism to handle runtime errors gracefully, ensuring the system remains robust and user-friendly.

8 Ideas to be Implemented Further

To enhance the functionality and user experience of the School Management System, the following features are proposed for future implementation:

- **Automated ID Generation:** Implement a system to automatically generate unique IDs for Students, Teachers, and Courses to minimize human error during data entry.
- **Dynamic Options Display:** Provide users with options for any choice, such as selecting which course, student, or teacher to manage during CRUD operations.

- **Real-time Database Updates:** Show updated database information immediately after any update is performed, ensuring users have the latest data at their fingertips.
- **Enhanced Student Model:** Add fields for overallGrade and overallAttendance in the Student model to provide a comprehensive view of student performance.

9 Conclusion

In conclusion, the School Management System offers a structured and efficient solution for managing various administrative tasks within an educational environment. By integrating distinct roles for principals, teachers, and students, the system provides tailored functionalities that enhance organization and accountability. The use of JSON files for data persistence ensures data integrity and easy access to information, while the robust class structure promotes modularity and scalability. The implementation of object-oriented programming concepts such as inheritance, encapsulation, and polymorphism has contributed to a well-designed system that is both maintainable and extensible. Future enhancements, including automated ID generation and improved CRUD options, will further optimize usability and reduce manual errors. This system lays a solid foundation for school management and holds significant potential for growth in response to evolving educational needs.