SYSTEM DESIGN – SCHOOL MANAGEMENT SYSTEM

Course: Data Structures and Algorithms

BDS Y2S1

GROUP 3

|  |  |  |
| --- | --- | --- |
| Name | Reg. No | Gender |
| KIPROP GIFT TALAM | CT204/119564/24 | M |
| MELICHIZEDEK SAKWA WEVASA | CT204/119568/24 | M |
| CORNELEA ACHIENG | CT204/119589/24 | F |
| KIUNA ESTHER NJERI | CT204/121105/24 | F |

BDS Y2S1

|  |  |  |
| --- | --- | --- |
| MODULE | DATA STRUCTURE | JUSTIFICATION |
| STUDENT REGISTRY | HASH TABLE | A hash table provides O(1) time complexity for insert, delete and search operations using student ID as the key. This is crucial since student searches are the most frequent operations in the system.  *Alternative Considered*: Linked list was rejected due to O(n) search time which becomes inefficient with thousands of students. |
| COURSE SCHEDULING | QUEUE | Queue ensure fair first-come-first-served enrolment, preventing course hoarding and maintaining proper registration order. Circular array implementation prevents memory wastage.  *Alternative Considered:* Priority Queue was rejected to maintain equal treatment of all students. |
| FEE TRACKING | BINARY SEARCH TREES | BST maintains fee records in sorted order (by date or amount), enabling efficient range queries for fee reports and clearance certificates. O(log n) operations are acceptable for fee management.  *Alternative Considered:* AVL Tree was rejected due to the expected data volume. |
| LIBRARY SYSTEM | STACK | Stack’s LIFO behavior perfectly models recent book returns and quick availability checks. Simple push/pop operations suit the library’s workflow.  Alternative Considered: Hash Map was considered for ISBN lookup but Stack better handles the return sequence. |
| PERFORMANCE ANALYTICS | GRAPH | Graph structrure can model subject prerequisites, student performance correlations, and identity patterns across different courses.  Alternative Considered: Matrix was considered but Graph offers more flexible relationship modelling. |

**ARCHITECHTURE OVERVIEW**

The system follows a modular design with a central *SchoolManager* class that coordinates all operations. Each module is implemented as a separate class with well-defined interfaces. Data flows between modules through the central manger, ensuring loose coupling while maintaining data consistency. For example, when a student registers, the *StudentRegistry* updates first, then the *CourseScheduler* can access this student for enrolment.

**Module Interactions:**

StudentRegistry- feeds student data to all other modules.

CourseScheduler- uses StudentRegistry to validate students.

FeeTracker- links to StudentRegistry for payment records.

LibrarySystem- independent but shares student validation.

PerformanceAnalyzer- aggregates data from all academic modules.

**FLOW DIAGRAMS AND PSEUDOCODE**

*StudentRegistry*

1. Start
2. Display Main Menu

Options:

* Register New Student
* Search Student
* Update Student Info
* Delete Student
* Display All Students
* Exit

1. User Selects Option

*If Register New Student*

1. Input Student Details → (Name, ID, Course, etc.)
2. Check if Student ID already exists in the hash table/database

* If Yes, display “Student Already Exists”
* If No, store record in hash table

1. Display “Student Registered Successfully”
2. Return to Menu

*If Search Student*

1. Input Student ID
2. Check if ID exists

* If Found, display Student Details
* If Not Found, display “Record Not Found”

1. Return to Menu

*If Update Student Info*

1. Input Student ID
2. Check if ID exists

* If Found, prompt to edit fields
* If Not Found, display “Record Not Found”

1. Update record in hash table
2. Display “Student Info Updated”
3. Return to Menu

*If Delete Student*

1. Input Student ID
2. Check if ID exists

* If Found, delete record from hash table
* If Not Found, display “Record Not Found”

1. Display “Student Deleted”
2. Return to Menu

*If Display All Students*

1. Traverse hash table
2. Display all student records
3. Return to Menu

*If Exit*

1. End Program