

**MAKERERE UNIVERSITY BUSINESS SCHOOL**

**FACULTY OF COMPUTING AND INFORMATICS**

**DEPARTMENT OF APPLIED COMPUTING AND IT**

**BACHELOR OF BUSINESS COMPUTING YEAR 3**

**ADVANCED WEB APPLICATION DEVELOPMENT**

**COURSEWORK TWO SEMESTER ONE ASSIGNMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **REG\_NUMBER** | **STUDENT\_NUMBER** | **SIGNATURE** |
| **MIRUNDI ARAFAT** | **22/U/6305** | **2200706305** |  |
| **LUKWAGO HERBERT** | **22/u/4478/EVE** | **2200704478** |  |
| **KANSIIME WINNIE** | **23/U/26658/EVE** | **2300726658** |  |

**Student Management System Documentation**

**1. Introduction**

The **Student Management System** is a web-based application designed to streamline the management of students, courses, enrolments, payments, timetables, and lecturers. The system is developed using the **Laravel framework** ensuring a secure and scalable solution for educational institutions. This system provides an efficient interface for administrators to manage student data, course details, financial records, and academic schedules.

The goal of the system is to automate key administrative tasks, improve data accuracy, and enhance user experience for students, lecturers, and administrators.

**2. System Overview**

The **Student Management System** consists of multiple entities, each representing a key component of academic and administrative processes:

**Entities:**

1. **Student**: Represents individual students with personal details, grades, and enrolled courses.
2. **Course**: Represents courses available for students, with details such as course name, description, and associated lecturer.
3. **Payment**: Tracks payments made by students for various services like tuition, course fees, etc.
4. **Enrolment**: Represents the relationship between students and courses, storing enrolment details such as grades.
5. **Timetable**: Stores the schedule for courses, specifying when and where each course takes place.
6. **Lecturer**: Represents academic staff who teach courses, with details like department, contact, and assigned courses.

**System Components:**

* **Frontend**: Built with Laravel's Blade templating engine and HTML/CSS for user interaction.
* **Backend**: Powered by Laravel controllers, models, and database management.
* **Database**: MySQL (or another relational database) to store all data related to students, courses, enrolments, payments, and more.

**Entities and Attributes**

**Student Entity**

The student entity represents individual students in the system. Students can be enrolled in one or more courses, and their payment details can be tracked.

**Attributes:**

* **id**: Integer, Auto-increment, Primary Key
  + A unique identifier for each student.
* **name**: String
  + The name of the student.
* **email**: String, Unique
  + The email address of the student. It must be unique within the system.
* **registration number**: String, Unique
  + The unique registration number for each student.
* **grade**: String
  + The current grade of the student (e.g., Freshman, Sophomore, etc.).
* **course**: String (Optional)
  + The name of the course the student is currently enrolled in. It is often used for convenience but is managed in the enrolment table in practice.

**Relationships:**

* **Payments**: A student can make multiple payments, so there is a one-to-many relationship between Student and Payment.
* **Enrolments**: A student can be enrolled in many courses, which is managed through the Enrolment table (many-to-many relationship).

**Course Entity**

The Course entity represents various courses available for students. Each course can have a list of enrolled students, and a lecturer responsible for the course.

**Attributes:**

* **id**: Integer, Auto-increment, Primary Key
  + A unique identifier for each course.
* **name**: String
  + The name of the course (e.g., "Mathematics 101", "Computer Science").
* **description**: Text
  + A brief description of the course, explaining its content and objectives.

**Relationships:**

* **Students**: A course can have many students enrolled. This is a many-to-many relationship via the Enrolment table.
* **Lecturer**: Each course can have one lecturer, establishing a one-to-many relationship between Lecturer and Course.
* **Timetable**: A course can have a timetable, which specifies when the course is scheduled.

**Payment Entity**

The Payment entity tracks payments made by students for various services, such as tuition fees, course fees, or fines.

**Attributes:**

* **id**: Integer, Auto-increment, Primary Key
  + A unique identifier for each payment.
* **amount**: Decimal (10,2)
  + The amount of money paid by the student.
* **renumber**: String
  + A unique reference number for each payment transaction.
* **date**: Date
  + The date the payment was made.
* **student**: Foreign Key
  + The ID of the student who made the payment. This is a foreign key referencing the student’s table.

**Relationships:**

* **Student**: A payment belongs to one student, establishing a one-to-many relationship from Student to Payment.

**Enrolment Entity**

The Enrolment entity manages the relationship between students and courses. It also stores additional information such as the grade a student received in a course.

**Attributes:**

* **id**: Integer, Auto-increment, Primary Key
  + A unique identifier for each enrolment record.
* **student**: Foreign Key
  + The ID of the student enrolled in the course. This is a foreign key referencing the students table.
* **Course\_id**: Foreign Key
  + The ID of the course the student is enrolled in. This is a foreign key referencing the courses table.
* **grade**: String
  + The grade the student has received in the course.

**Relationships:**

* **Student**: An enrolment belongs to one student.
* **Course**: An enrolment belongs to one course.
* **Student-Course**: A student can enroll in multiple courses, and a course can have many students (many-to-many relationship).

**Entity**

The Timetable entity defines the schedule for courses, specifying when and where each course is held.

**Attributes:**

* **id**: Integer, Auto-increment, Primary Key
  + A unique identifier for each timetable entry.
* **course\_id**: Foreign Key
  + The ID of the course this timetable entry is associated with. This is a foreign key referencing the courses table.
* **day**: String
  + The day of the week the course is scheduled to take place (e.g., Monday, Tuesday, etc.).
* **time**: Time
  + The time the course is scheduled (e.g., 10:00 AM, 2:00 PM, etc.).
* **room**: String
  + The room or location where the course is being taught.

**Relationships:**

* **Course**: A timetable belongs to one course.

**Lecturer Entity**

The Lecturer entity represents academic staff responsible for teaching courses. Lecturers are linked to the courses they teach.

**Attributes:**

* **id**: Integer, Auto-increment, Primary Key
  + A unique identifier for each lecturer.
* **name**: String
  + The name of the lecturer.
* **email**: String, Unique
  + The email address of the lecturer. It must be unique.
* **department**: String
  + The department the lecturer belongs to (e.g., Computer Science, Mathematics).
* **contact**: String
  + The contact number of the lecturer.

**Relationships:**

* **Courses**: A lecturer can teach many courses, establishing a one-to-many relationship between Lecturer and Course.

**Database Relationships and Schema**

**Student - Enrollment - Course (Many-to-Many)**

Students can enroll in multiple courses, and courses can have multiple students enrolled. This many-to-many relationship is managed through the Enrollment table.

* **Student** — has many — **Enrollments**
* **Course** — has many — **Enrollments**
* **Enrollment** — belongs to — **Student**
* **Enrollment** — belongs to — **Course**

**Payment - Student (One-to-Many)**

Each payment is linked to a single student, but a student can make multiple payments.

* **Student** — has many — **Payments**
* **Payment** — belongs to — **Student**

**Course - Timetable (One-to-Many)**

Each course has a timetable specifying when and where it is held.

* **Course** — has many — **Timetables**
* **Timetable** — belongs to — **Course**

**Lecturer - Course (One-to-Many)**

Each course is taught by a lecturer, and a lecturer can teach many courses.

* **Lecturer** — has many — **Courses**
* **Course** — belongs to — **Lecturer**

**System Features**

**Add and Manage Students**

* Add new students by entering their details (name, email, registration number, grade).
* View, edit, or delete student records.

**Add and Manage Courses**

* Add new courses with descriptions.
* View, edit, or delete course records.
* Assign courses to lecturers.

**Manage Student Enrollments**

* Enroll students in courses.
* Assign grades to students for specific courses.
* View the list of students enrolled in a particular course.

**Process Payments**

* Record payments made by students (tuition, course fees).
* Generate a payment reference number.
* Track payment history for each student.

**Manage Timetables**

* Create and update timetables for courses (schedule and room allocation).
* View course timetables for specific days and times.

**Manage Lecturers**

* Add, update, and delete lecturer information.
* Assign lecturers to specific courses.

**Print desired reports**

**Functional Requirements**

Functional requirements define the specific behaviors and features of the system. Below are the core functional requirements for the **Student Management System**:

**User Roles and Access Control**

* **Administrator**: Has full access to all system features, including managing students, courses, payments, enrollments, lecturers, and timetables.
* **Lecturer**: Can view and manage the courses they are assigned to, including course schedules and student grades.
* **Student**: Can view their enrolled courses, timetable, payment history, and grades.

**Student Management**

* **Add New Student**: The administrator can add a new student by providing personal information such as name, email, registration number, grade, and course.
* **Edit Student Information**: The administrator can edit student details, including updating their grades and enrolled courses.
* **View Student Details**: Administrators, lecturers, and students can view student profiles with relevant details like grades, enrollment status, and payment history.
* **Delete Student**: The administrator can delete a student from the system (after proper checks).

**Course Management**

* **Add New Course**: Administrators can create new courses, specifying the course name, description, and other details.
* **Assign Courses to Lecturers**: Administrators can assign courses to specific lecturers.
* **View Course Information**: Administrators and lecturers can view course details, including the list of enrolled students and timetables.

**Payment Management**

* **Record Payments**: Administrators can record payments made by students, including payment amount, reference number, and date.
* **View Payment History**: Students and administrators can view payment history for each student, including past transactions.
* **Generate Payment Receipt**: A payment receipt is generated for students after successful transactions.

**Enrollment Management**

* **Enroll Students in Courses**: Administrators can enroll students in various courses.
* **Assign Grades to Students**: Lecturers can assign grades to students after completing a course.
* **View Enrollment Details**: Administrators and lecturers can view the list of students enrolled in a course, along with their grades.

**Timetable Management**

* **Create and Update Timetables**: Administrators can create and update course timetables (course days, times, and rooms).
* **View Timetables**: Students and lecturers can view the timetable for their enrolled courses and assigned lectures.

**Lecturer Management**

* **Add and Edit Lecturer Details**: Administrators can add new lecturers and edit their details (name, email, department, contact).
* **Assign Lecturers to Courses**: Administrators can assign courses to specific lecturers.

**Non-Functional Requirements**

Non-functional requirements define the system’s operational attributes, such as performance, security, and usability. Below are the key non-functional requirements for the **Student Management System**:

**Performance**

* **System Response Time**: The system should respond to user interactions within **2 seconds** for most actions (e.g., loading a student record, submitting a payment).
* **Concurrent Users**: The system should be able to handle **up to 100 concurrent users** without significant performance degradation.
* **Database Efficiency**: All queries should be optimized for performance, especially for handling large datasets (e.g., student records, payments).

**Security**

* **Authentication and Authorization**: The system should implement role-based access control (RBAC) to ensure that users can only access authorized areas (e.g., only administrators can add students or manage payments).
* **Data Encryption**: Sensitive information, such as student email, payment data, and grades, should be encrypted and stored securely.
* **CSRF Protection**: All forms in the application should include CSRF tokens to protect against cross-site request forgery.
* **Password Storage**: All user passwords should be securely hashed using Laravel’s built-in hashing functions (e.g., bcrypt).
* **SSL Encryption**: The application should be accessible only via HTTPS, ensuring that all data transmitted between the client and server is encrypted.

**Usability**

* **User Interface**: The system should have a user-friendly interface that is easy to navigate for all types of users (administrators, lecturers, students).
* **Accessibility**: The system should comply with accessibility standards, ensuring that users with disabilities can access the system using screen readers and other assistive technologies.
* **Mobile Responsiveness**: The system should be fully responsive, providing an optimal experience for users accessing the system from mobile devices or tablets.

**Reliability**

* **System Uptime**: The system should have an uptime of **99.9%** to ensure it is available for use most of the time.
* **Data Integrity**: The system should ensure the integrity of all data by using database constraints (e.g., foreign keys, unique constraints) and by implementing validation checks on user inputs.
* **Error Handling**: The system should handle errors gracefully, providing meaningful error messages to users while logging technical details for developers.

**Scalability**

* **Database Scalability**: The database should be able to scale to accommodate an increasing number of students, courses, payments, and other records.
* **Modular Architecture**: The system should be designed in a modular way to allow easy addition of new features (e.g., adding new entities, extending the system for new roles).

**Maintainability**

* **Code Quality**: The system should be developed following best coding practices, with clean, readable, and well-documented code.
* **Testing**: The system should include automated unit and integration tests to ensure the stability and reliability of the application.
* **Version Control**: The system should use version control (e.g., Git) to manage changes in the codebase and enable collaboration among developers.

**System Architecture**

The **Student Management System** is designed with a **3-tier architecture** consisting of:

1. **Presentation Layer (Frontend)**: Built with **Laravel Blade** templates and basic HTML/CSS for a responsive user interface.
2. **Business Logic Layer (Backend)**: The backend is powered by **Laravel controllers**, **models**, and **services**, ensuring separation of concerns and maintainable code.
3. **Data Layer (Database)**: The system uses a **MySQL** (or other relational) database to store and manage data, with tables for students, courses, payments, enrollments, and timetables.

**Technology Stack:**

* **Frontend**: Laravel Blade, HTML5, CSS3, Bootstrap (for responsiveness)
* **Backend**: Laravel PHP framework (MVC architecture)
* **Database**: MySQL (or other relational databases)
* **Version Control**: Git/GitHub
* **Authentication**: Laravel Breeze (or Laravel Sanctum for API-based auth)

**Conclusion**

The **Student Management System (SMS)** is a robust, secure, and scalable application designed to meet the needs of educational institutions. It provides features for managing students, courses, payments, timetables, and lecturers in a streamlined and efficient way. The system adheres to both functional and non-functional requirements, ensuring that it is performant, secure, user-friendly, and maintainable.

This documentation provides a detailed overview of the system, its functionality, and its requirements, ensuring that stakeholders can fully understand how the system operates and its key capabilities.