MEKELLE UNVIERSITY

EITM
SCHOOL OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

SOFTWARE DOCUMENTATION SPECIFICSTION(SRS) OF STUDENT REGISTRATION SYSTEM

- Name: Semere Herruy
- ID: ugr/177912/12

Submission date: 08/02/2017 E.c
Submitted to : Inst. Mesele

Table of Contents

Student Registration System

1. Introduction

This document provides the Software Requirements Specification (SRS) for the **Student Registration System**. The system is designed to automate the student enrollment process by enabling students to register for courses, make payments, and manage their registration details. It addresses the functional and non-functional requirements, as well as the overall system design and behavior. The focus of this document is on defining the system's capabilities, constraints, and expected functionality, ensuring that all stakeholders have a clear understanding of what the system should accomplish.

1.1 Purpose

The purpose of this document is to outline the software requirements for the Student Registration System. It defines the system's functionality, performance, and other requirements, including security, usability, and reliability, which the system must meet. The document serves as a comprehensive guide for developers, testers, and stakeholders involved in the development, deployment, and maintenance of the system.

1.2 Overview of the System

The **Student Registration System** allows students to register for courses through an online interface. It supports student registration, course management and secure payment processing. The system is built on Laravel and follows a modular architecture with core components such as API Gateway, Student Registration, Course Registration, Payment Service, Session Store, and a centralized database. The system ensures smooth and reliable operation through secure payment processing, session management, and notification services to confirm successful registrations and payments.

1.3 Scope of the Document

This document focuses on specifying the software requirements for the **Student Registration System**. It outlines functional and non-functional requirements, system constraints, and so on. The document also includes descriptions of use cases, data handling, and testing strategies. It provides the necessary foundation to guide the development and ensure that the system meets its intended goals.

The scope does not cover detailed design or user interface specifics, which will be documented separately. The document also suggests future enhancements such as integration of an admin dashboard and scalability improvements.

2. Architectural Approach

The system employs a layered architectural style to ensure maintainability and scalability while using Laravel as the development framework. Although microservices principles are partially adopted, the system maintains a focus on robust session management and seamless integration of components.

2.1 System Architecture

The system uses a layered architecture with key components like the API Gateway, Student and Course Registration, Payment Service, Session Store, and Database. These components communicate via defined interfaces, ensuring scalability and maintainability. The architecture is designed to handle session management, secure payments, and notifications efficiently.

2.2 Design Considerations

Key considerations include ensuring system scalability, handling large amounts of concurrent users, maintaining data security, providing fault tolerance, and optimizing session management for a seamless user experience.

3. System Components

3.1 **API Gateway:** Acts as a single entry point for client requests, handling routing, authentication, and other cross-cutting concerns.

3.2 **Student Registration Component:** Handles student input validation and stores data temporarily in sessions.

3.3 **Course Registration Component:** Manages course selection and integrates it into session data.

3.4 **Payment Service Component:** Integrates with Chapa for payment processing and manages session data during the payment workflow.

3.5 **Session Store:** Stores session data using Redis or a database to maintain state across HTTP requests.

3.6 **Database:** Persistent storage for students, courses, registrations, and payments.

3.7 **Central Logging Service:** Collects logs for debugging and performance monitoring.

4. System Architecture

4.1 Data Flow

Data flows through the system from student registration to course selection, payment processing, and confirmation. The API Gateway routes requests, while session data is temporarily stored and later persisted in the database after successful payment.

4.2 Component Interaction

Components interact via well-defined APIs. The API Gateway manages requests, the Registration and Payment Services handle business logic, the Session Store stores temporary data, and the Database retains permanent records. Notifications and error handling are managed centrally.

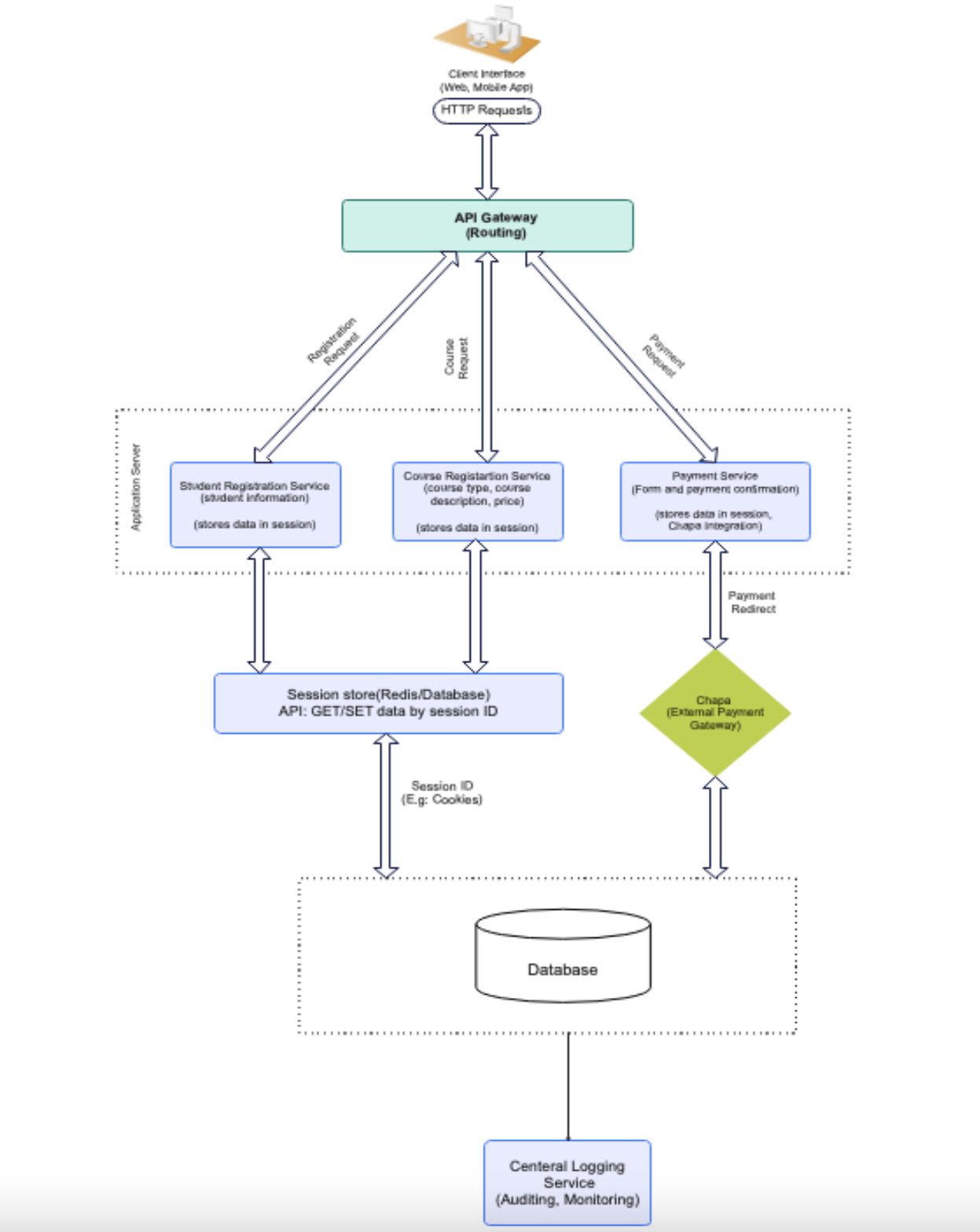

Figure 1 : System architecture

5. UML Diagrams

◆ 5.1 Class Diagram:

```
class Student {
  - id: Integer
  - name: String
  - email: String
  - age: Integer
  - phoneNumber: String
  + registerForCourse(course: Course): void
}

class Course {
  - id: Integer
  - name: String
  - description: String
  - amount: Float
  + checkAvailability(): Boolean
}

class Payment {
  - id: Integer
  - registrationId: Integer
  - status: String
  - amountPaid: Float
  + processPayment(): Boolean
}

class Session {
  - sessionId: String
  - data: Map
  + storeSessionData(key: String, value: String): void
  + getSessionData(key: String): String
}

class APIService {
  + routeRequest(endpoint: String): void
}

class Database {
  + saveData(entity: Object): void
  + retrieveData(entityType: String, id: Integer): Object
}

' Relationships
Student --> Course : "registers for"
Student --> Session : "stores session data"
Course --> Session : "stores session data"
Payment --> Session : "uses session data"
APIService --> Student : "routes requests for"
```

APIService --> Course : "routes requests for"
APIService --> Payment : "routes requests for"
Database --> Student : "stores"
Database --> Course : "stores"
Database --> Payment : "stores"

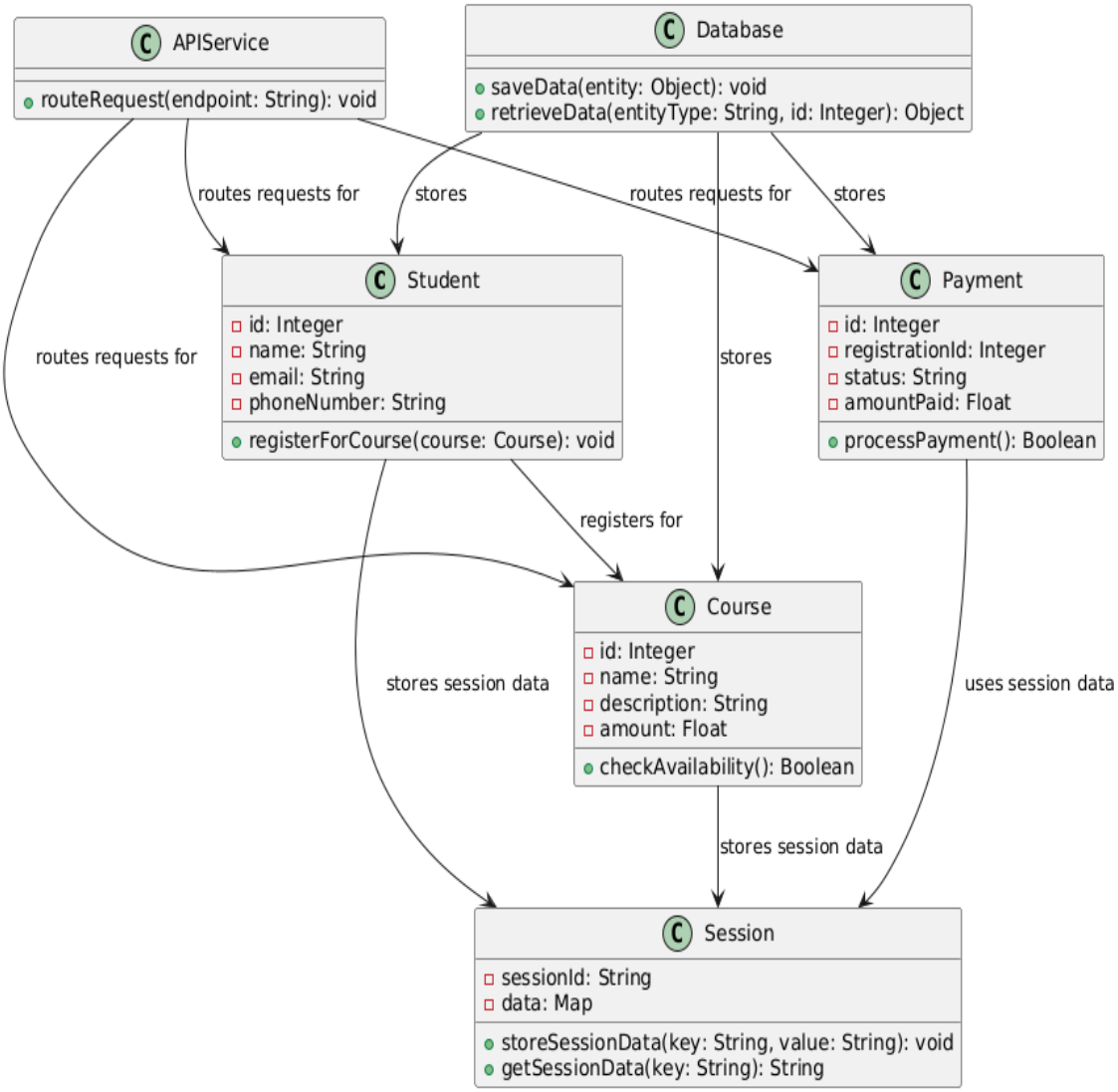

Figure 2 : class diagram

◆ 5.2 Sequence Diagram:

- Code snippet

```
participant Client
participant App (Payment Service)
participant Session Store
participant Chapa

Client ->> App: Initiate Registration
App ->> SessionStore: Store Session Data
App -->> Client: Redirect to Chapa
Client ->> Chapa: Complete Payment
Chapa -->> App: Callback with Payment Data
App ->> SessionStore: Retrieve Session Data
App ->> Database: Store Registration and Payment
App ->> NotificationService: Send Confirmation    //future consideration
App -->> Client: Show Success/Failure
```

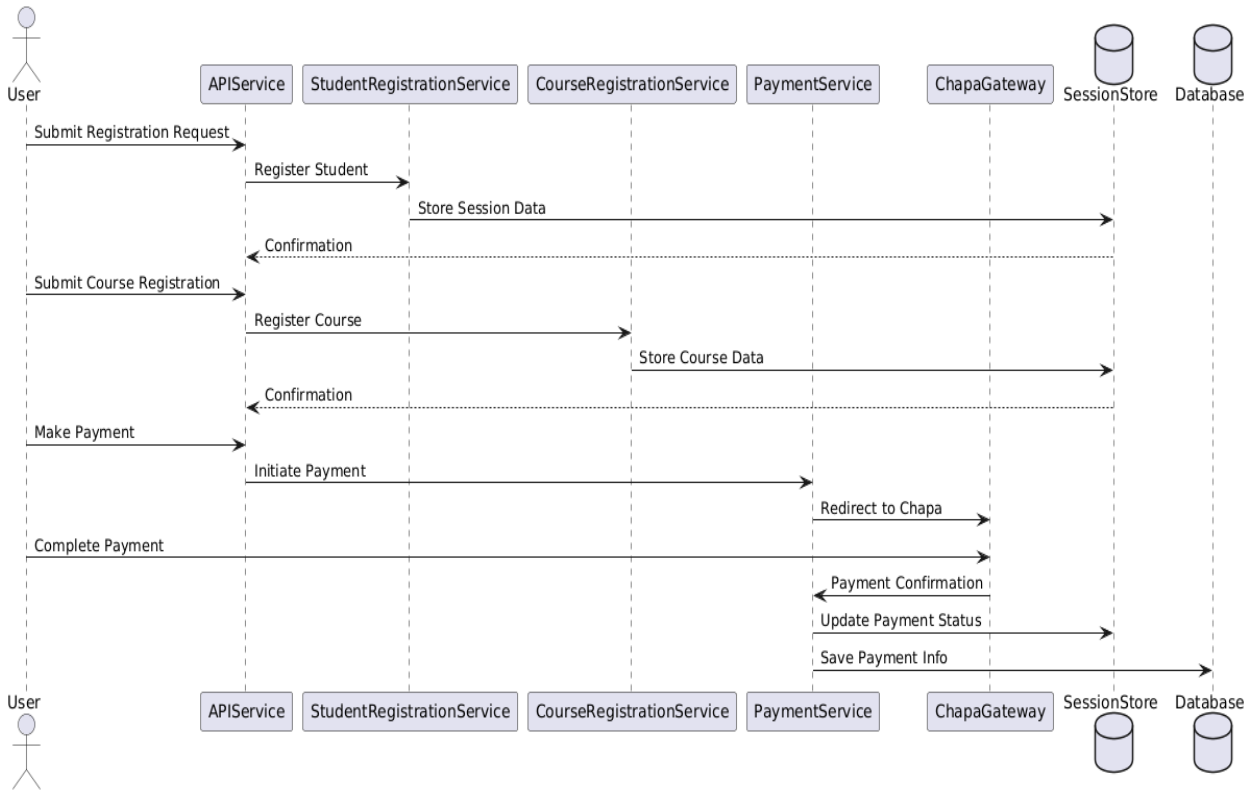

Figure 3 : sequence diagram

◆ 5.3 Activity Diagram:

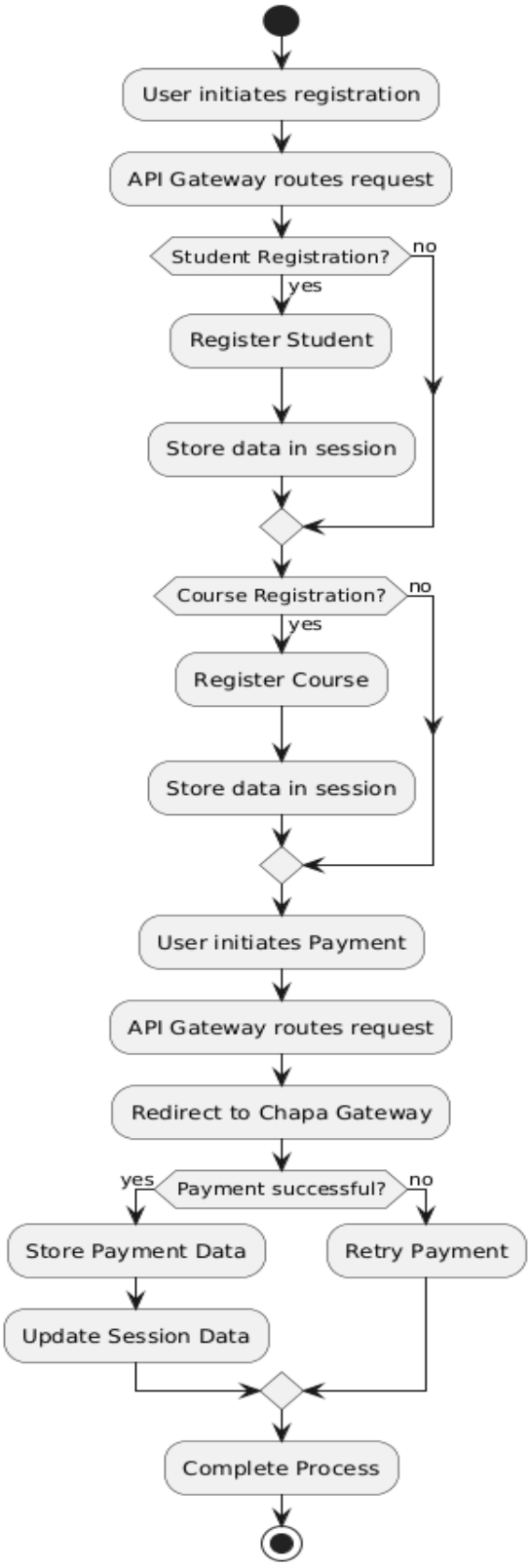

Figure 4 : Activity diagram

6. Functional Requirements

1. The system shall allow students to register for courses.
2. The system shall send confirmation emails upon successful payment.
3. The system shall validate student input during registration.
4. The system shall maintain session data securely during payment processes.

7. Non-Functional Requirements

1. **Performance:** The system must support concurrent users.
2. **Scalability:** The architecture must allow for the addition of new services with minimal configuration.
3. **Security:** All sensitive data must be encrypted both in transit and at rest.
4. **Usability:** Ensure intuitive interfaces for students and administrators.
5. **Availability:** The system must maintain 99.9% uptime.

8. Use Case Descriptions

8.1 Use Case: Register for a Course

Actor: Student
Preconditions:

- The system is operational and accessible via the client interface.
- The student must have access to the system via the **API Gateway**.

Steps:

1. The student accesses the course catalog through the **Course Registration Service**.
2. The system retrieves the available courses and displays them to the student.
3. The student selects a course and initiates the registration process.
4. The selected course and student details (entered during registration) are temporarily stored in the **Session Store**.
5. The system prompts the student to proceed to the payment process.

Postconditions:

- The selected course and student registration data remain in the session (not in the database).
- The student is redirected to the payment process.

8.2 Use Case: Process Payment

Actor: Student
Preconditions:

- Registration data (student and course details) must exist in the **Session Store**.
- The student must proceed to the payment process within the session's validity period.

Steps:

1. The student initiates the payment process via the **Payment Service**.
2. The system redirects the student to the **Chapa Payment Gateway**.
3. The payment gateway processes the payment and sends a callback to the system with the payment status.
4. The system verifies the payment status (success or failure).
    - **If payment is successful**:
        - The student registration and course details are saved to the **Database**.
        - The payment record is created and linked to the registration.
    - **If payment fails**:
        - The session data remains intact for a retry.
        - An error message is displayed to the student.

Postconditions:

- For successful payments:
    - Student and course data are recorded in the **Database**.
    - Payment status is updated, and a redirect to home page.
- For failed payments:
    - The session retains the registration data for retrying the payment process.

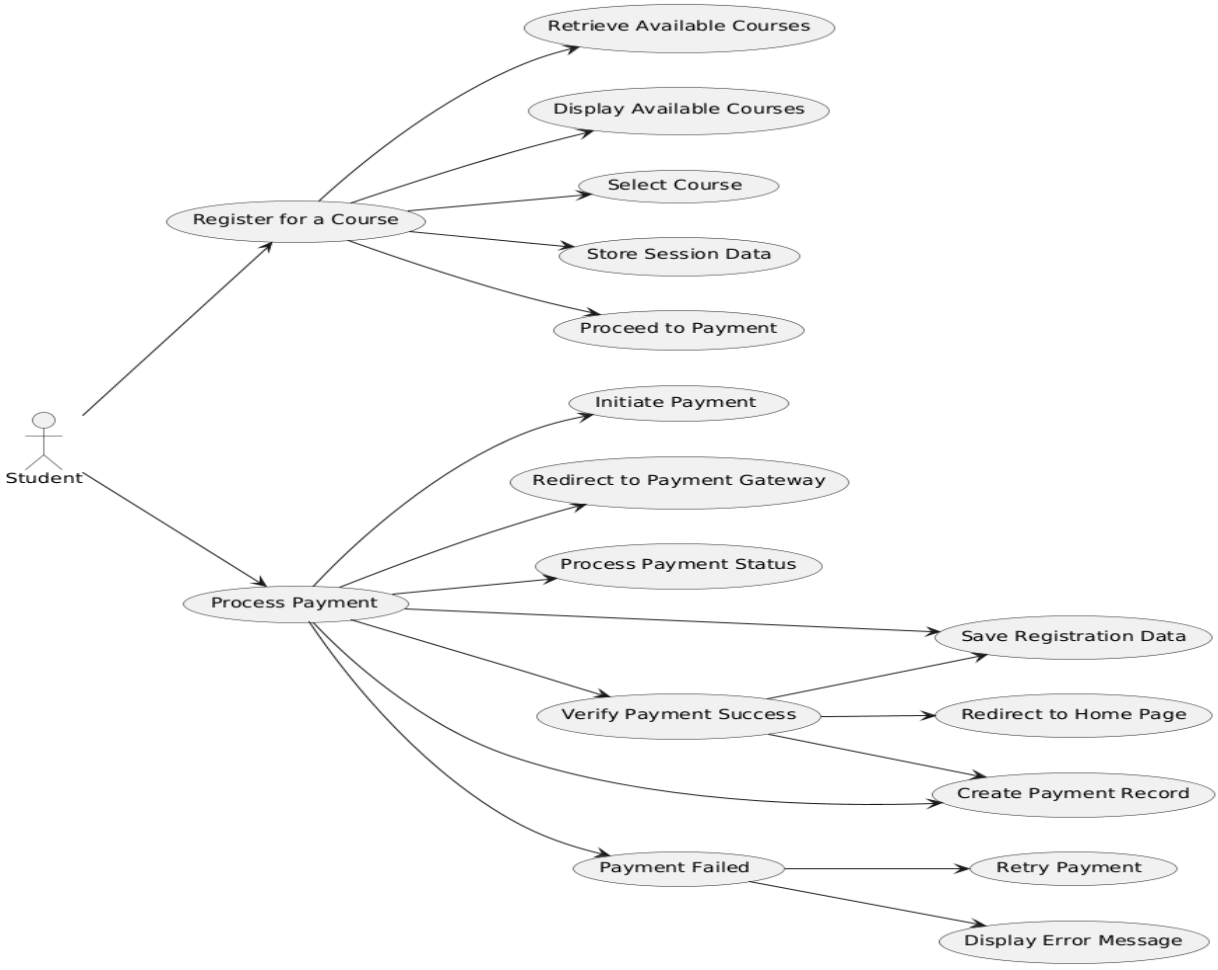

Figure 5 : Usecase diagram

9. Data Dictionary

Key Entities and Attributes:

- **Student:** {id: Integer, name: String, email: String, phoneNumber: String}
- **Course:** {id: Integer, name: String, description: String, amount: Float}
- **Payment:** {id: Integer, registrationId: Integer, status: String, amountPaid: Float}

10. Testing and Validation Plan

1. **Unit Testing:** Validate individual components.
2. **Integration Testing:** Test interactions between modules like registration and payment.
3. **User Acceptance Testing (UAT):** Verify the system meets end-user needs.
4. **Error Handling Testing**: Test scenarios like invalid input, payment failures, and expired sessions to verify proper error handling.
5. **Basic User Testing**: Ensure that basic flows like course selection, session handling, and payment redirection work as expected.
6. **End-to-End Testing (Simplified)**: Validate the complete flow from course selection to successful payment and database record creation.

11. Implementation Plan

The system will be implemented in the following phases:

1. Develop core services: API Gateway, Student Registration, and Course Registration.
2. Integrate with Chapa payment gateway.
3. Deploy session store and database.
4. Conduct thorough testing (unit, integration, and UAT and others).
5. Launch and monitor the system.

12. Error Handling and Logging Strategy

1. Payment Failures: The system will log payment failures and notify users to retry.
2. Input Validation Errors: Display errors to users for correction.
3. Server Errors: Log errors for debugging.
4. Monitoring: Centralized logging ensures error traceability.

13. Future Enhancements

1. Implementing a full API Gateway for enhanced scalability.
2. Adding a reporting module for administrative insights.
3. Utilizing a message queue for asynchronous notifications.
4. Add admin dashboard that provide administrators with access to student and course data.
5. Add additional features like Notification Service, User service (log in, sign up, logout)

14. Architectural Style Reference

The system follows a layered architecture with a strong focus on separation of concerns. It incorporates elements of a microservices approach by separating key functionalities into distinct components, but it is not a fully distributed microservices system.

15. Conclusion

This revised document provides a clearer and more robust design for the Student Registration System. By addressing session management concerns and explicitly showing the data flow and API interactions, it provides a solid foundation for a production-ready applic