STUDENT COURSE REGISTRATION SYSTEM

A Project Report Submitted by

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# TABLE OF CONTENTS

1. Introduction
2. Objectives
3. Software and tools used
4. System design and architecture
5. Data structure used
6. Module Description
7. Exception Handling
8. Testing
9. User stories
10. Uml diagrams
11. Future Enhancements
12. Conclusion

**ABSTRACT**

The Student Course Registration System is a Java-based console application designed to automate and simplify the process of student enrollment in courses.  
The system allows students to register, log in, and manage their courses efficiently. Administrators can add, update, or delete course details as required.  
The project uses **AWS DynamoDB** as a cloud database for persistent storage, ensuring reliability and scalability.  
Key features include **BCrypt password encryption, custom exception handling**, and **layered architecture** for better modularity.  
The project demonstrates principles of **object-oriented programming**, **SOLID design**, and **data validation**.  
This system aims to enhance efficiency, maintain security, and provide a user-friendly interface for course management.

1. INTRODUCTION

The **Student Course Registration System** is a Java-based application that helps colleges and universities manage student course registrations easily. It replaces the old manual method with a **computerized system** that saves time and reduces mistakes.

Using this system, **students** can create an account, view available courses, enroll in the ones they like, and drop courses when needed — all through a simple text-based menu on the console.

**Administrators** can use the same system to manage course details, check how many students have enrolled, and see how many seats are left in each course. The system also makes sure that no student is enrolled in the same course twice and that seat limits are not crossed.

The system uses **AWS DynamoDB**, a cloud-based database, to store and manage information safely and efficiently. Because it runs on clouds, it can handle large amounts of data and multiple users at the same time without slowing down.

The project is built using **Java** and follows a **layered architecture**, which divides the program into different parts — such as data handling, business logic, and user interface. This design makes the system easier to update, maintain, and test in the future.

2. OBJECTIVES

* To automate student and course registration using a database-driven approach.
* To separate business logic from database logic through a layered architecture.
* To maintain data integrity and prevent duplication using indexed email management.
* To implement robust validation and error-handling mechanisms.
* To build a scalable and maintainable Java application.

**3. SOFTWARE AND TOOLS USED**

**3.1 Programming Language**

Java (JDK 17 or above)  
The entire project is developed using Java because of its platform independence, strong object-oriented features, and reliability in enterprise-level applications.

**3.2 Integrated Development Environment (IDE)**

IntelliJ IDEA Community Edition  
Used for writing, debugging, and executing Java code.  
IntelliJ IDEA offers intelligent code completion, Maven integration, and built-in debugging tools for better development efficiency.

**3.3 Build and Dependency Management**

Apache Maven  
Maven is used for managing dependencies, building the project, and maintaining a clean directory structure.  
It simplifies integration with external libraries such as AWS SDK and jBCrypt.

**3.4 Database**

AWS DynamoDB  
A NoSQL cloud-based database used to store all student and course-related data.  
It offers fast performance, scalability, and flexible schema design.  
DynamoDB stores data as key–value pairs, which align well with Java’s Map data structure.

**3.5 Version Control System**

Git and GitHub  
Used for source code version control and project collaboration.  
GitHub helps track changes, maintain backups, and support teamwork throughout the development process.

**3.6 Testing Framework**

JUnit (Java Testing Framework)  
Used to test individual modules such as registration, login, and course enrollment.  
Ensures all methods perform correctly and the application behaves as expected.

**3.7 Operating System**

Windows 10 / 11  
Used as the main development and testing platform.  
Since Java is platform-independent, the system can also run on Linux or macOS environments.

4. SYSTEM DESIGN AND ARCHITECTURE

The system follows a layered architecture, dividing the application into multiple layers, each handling a specific responsibility. This design promotes code reusability, testability, and scalability.

The main layers are:

* Model Layer - Defines entities like Student, Course, and Enrollment.
* DAO Layer - Manages data access and interactions with AWS DynamoDB.
* Service Layer - Contains business logic and validation mechanisms.
* Configuration Layer - Handles database setup and connectivity.
* Presentation Layer - Provides console-based user interface.

5. DATA STRUCTURES USED

The Student Course Registration System utilizes both Java Collection Framework data structures and AWS DynamoDB data models to efficiently store, organize, and manage data.  
These structures are chosen to improve data access speed, memory management, and system scalability.  
They also provide flexibility for storing data temporarily during runtime and for maintaining permanent storage in the database.

**5.1 List (ArrayList)**

The List interface, specifically implemented through the ArrayList class, is used to store and retrieve multiple student and course objects in an ordered manner.  
Lists maintain the insertion order and allow duplicate elements, making them suitable for representing data such as available courses or lists of enrolled students.

Example:

List<Course> courseList = new ArrayList<>();

Usage in Project:  
Used in the CourseService and StudentService classes to display multiple course records and maintain ordered data for output purposes.

**5.2 Map (HashMap)**

The Map interface, implemented using the HashMap class, is used to store data in key–value pairs.  
It allows quick access and retrieval of data using a unique key, such as a studentId or courseId.  
Maps are ideal for storing database records in memory for fast lookups and modifications.

Example: Map<String, Student> studentMap = new HashMap<>();

Usage in Project:  
Used in the StudentDao and CourseDao classes to map DynamoDB table attributes to Java objects.  
For example, course IDs can be mapped to their respective course names or details.

**5.3 Set (HashSet)**

The Set interface, implemented by the HashSet class, is used to maintain collections of unique elements.  
Since a Set does not allow duplicate values, it is particularly useful for ensuring that a student cannot enroll in the same course multiple times.

Example:

Set<String> enrolledCourses = new HashSet<>();

Usage in Project:  
Used in the RegistrationService class to manage student course enrollments and avoid duplicate entries.

6. MODULE DESCRIPTION

* Student Module - Handles student registration and authentication.
* Course Module - Manages course creation, updates, and availability.
* Enrollment Module - Processes student enrollments and waitlists.
* Admin Module - Allows administrators to manage students and courses.
* Drop Module - Enables students to drop courses and updates seat counts.

7. EXCEPTION HANDLING

* Exception handling ensures smooth execution of the program even when unexpected errors occur.
* The project uses custom exceptions to identify and handle different types of errors.
* The main custom exceptions are ValidationException, DataNotFoundException, and DuplicateEntryException.
* ValidationException is raised when user input is invalid or incomplete.
* DataNotFoundException is triggered when a student or course record is missing in the database.
* DuplicateEntryException prevents a student from enrolling in the same course twice.
* All exceptions are handled using try–catch blocks for safe execution.
* Error messages are displayed clearly to guide the user in correcting input.
* Exception handling improves the stability and reliability of the system.
* It ensures that the application does not crash and provides a better user experience.

8. TESTING

* Testing was performed to ensure the project works correctly and meets all requirements.
* Both manual testing and automated testing were carried out using JUnit 5.
* Unit testing was done on individual methods like registration and login.
* Integration testing verified the connection between DAO, Service, and Database layers.
* Validation testing checked input formats and error messages for wrong data.
* Exception testing confirmed that all custom exceptions trigger correctly.
* Database testing verified correct storage and retrieval in AWS DynamoDB.
* All test cases passed successfully with expected outputs.
* The application handled both valid and invalid inputs effectively.
* Testing ensured that the system is stable, reliable, and ready for deployment.

**9. USER STORIES**

* User stories describe how different users interact with the Student Course Registration System.They represent the system requirements from the end-user's perspective, explaining what the user wants to do and why.

Each story follows the format:"As a [type of user], I want to [perform an action], so that [I can achieve a goal]."

**9.1 Student User Stories**

* US-01: As a student, I want to register an account so that I can access the system.
* US-02: As a student, I want to log in securely using my email and password.
* US-03: As a student, I want to view all available courses.
* US-04: As a student, I want to enroll in selected courses.
* US-05: As a student, I want to see a list of courses I have enrolled in.
* US-06: As a student, I want to drop a course if needed.
* US-07: As a student, I want to reset my password if I forget it.

**9.2 Admin User Stories**

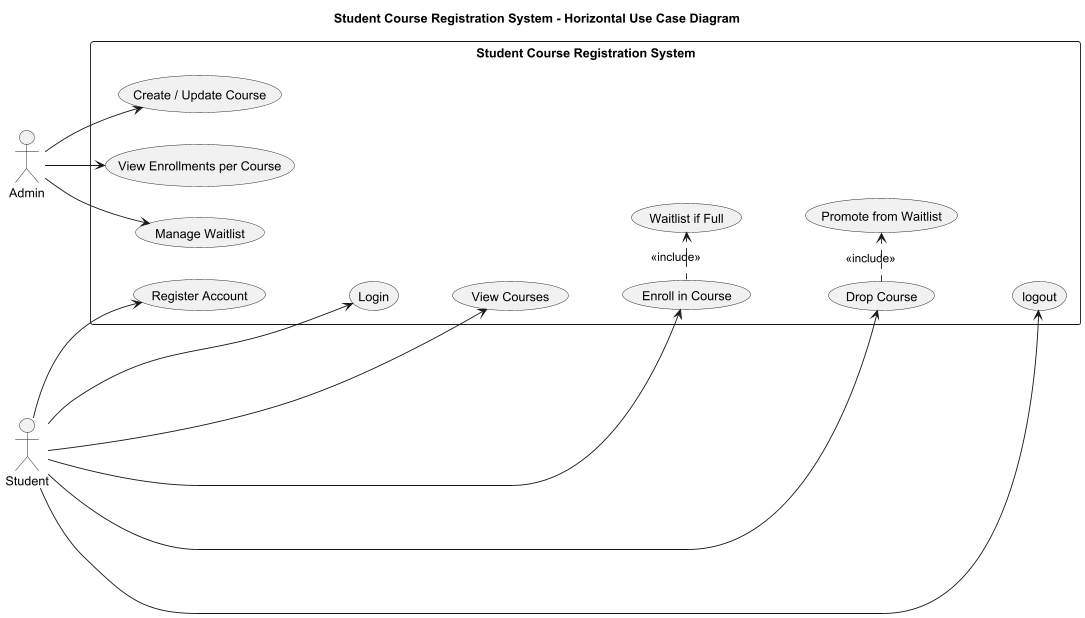
* US-08: As an admin, I want to add new courses.
* US-09: As an admin, I want to update or delete existing courses.
* US-10: As an admin, I want to view all registered students.
* US-11: As an admin, I want to remove incorrect or duplicate registrations.

**9.3 System Stories (Technical)**

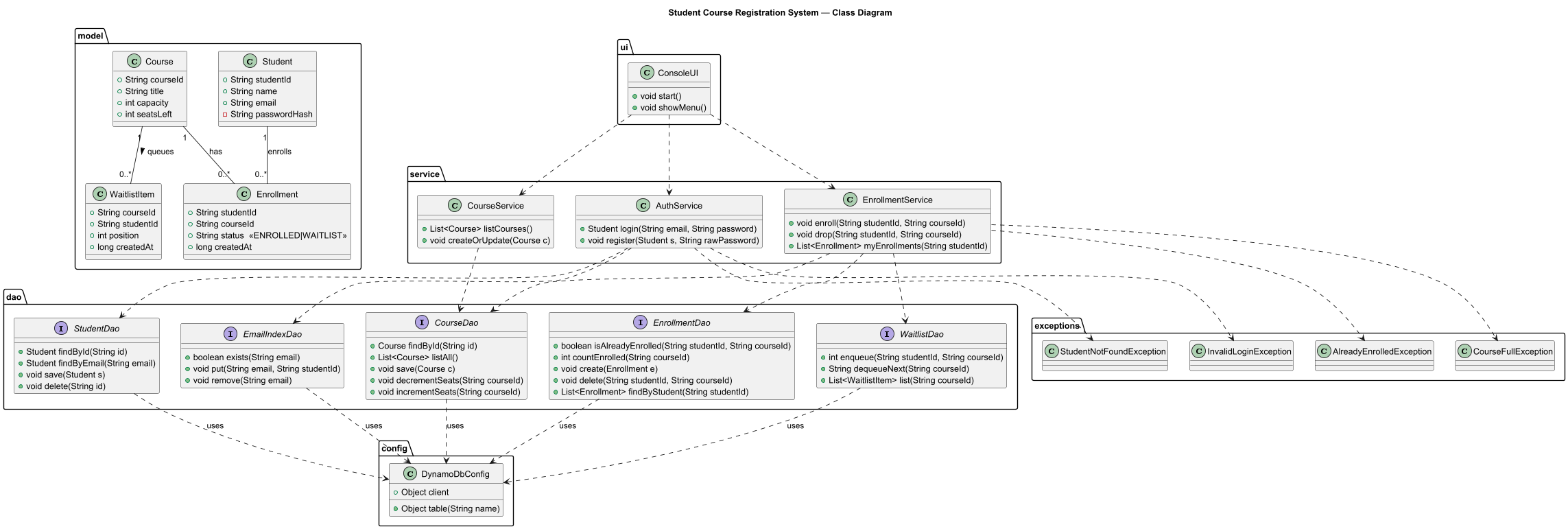
* US-12: As a system, I want to validate user input.
* US-13: As a system, I want to encrypt passwords using BCrypt.
* US-14: As a system, I want to connect to AWS DynamoDB.

10. UML Diagrams

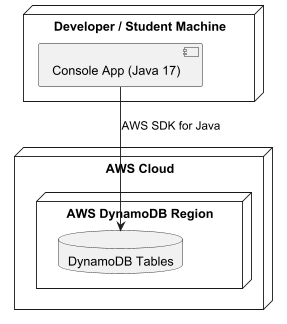
10.1 use\_case diagram



10.2: class Diagrams



10.3 Deployement



11. FUTURE ENHANCEMENTS

1. Develop a web-based or mobile interface for better accessibility.
2. Add email/SMS notifications for successful registrations.
3. Implement advanced search and filter options for courses.
4. Create an admin dashboard with analytics and reports.
5. Integrate payment gateways for paid courses.

12. CONCLUSION

The Student Course Registration System successfully automates the process of managing student registrations and course enrollments.  
It ensures data security using BCrypt, maintains modularity through SOLID design, and provides reliable storage via AWS DynamoDB.  
The project demonstrates the practical implementation of **object-oriented principles**, **exception handling**, and **Java collections**.  
It can be further enhanced into a complete web or mobile application to serve educational institutions on a larger scale.