Computer and communication department Year Two

Programming II (CC272)

Lab 6 – SkillForge UML Design

Team members:

• Bassant Salah Abdelsabour	9357
• Zeina Mohammed Elbanna	9536
• Sara Mohamed Bassiouni	9311
• Sara Ahmed Zaghlool	9325

Date of submission: November 2nd, 2025

1.Introduction

This report presents the UML analysis and design of the SkillForge platform — an online learning system that allows students to enroll in courses, complete lessons and quizzes, and earn certificates. The purpose of this lab is to model the system's structure and behavior using standard UML diagrams.

2.Description

The SkillForge platform enables three main types of users: Students, Instructors, and Admins.

- **Students** can browse and enroll in courses, complete lessons and quizzes, and earn certificates upon completion.
- **Instructors** are responsible for creating and managing courses, uploading lessons, and designing quizzes to assess learners' understanding.
- **Admins** oversee the platform, managing users, approving or removing courses, and tracking system analytics.

Each course contains multiple lessons and quizzes, and when a student completes all lessons and passes all quizzes, a certificate is automatically generated.

The system uses a JSON-based database for data storage, managed by a JsonDatabaseManager class, while service classes such as UserService, CourseService, and QuizService handle the business logic and communication with the data layer.

3.UML Diagrams

First Use Case Diagram:

First the actors:

We have 3 primary actors:

1-Student

2-Instructor

3-Admin

Each actor performs different actions (use cases), shown as blue ovals

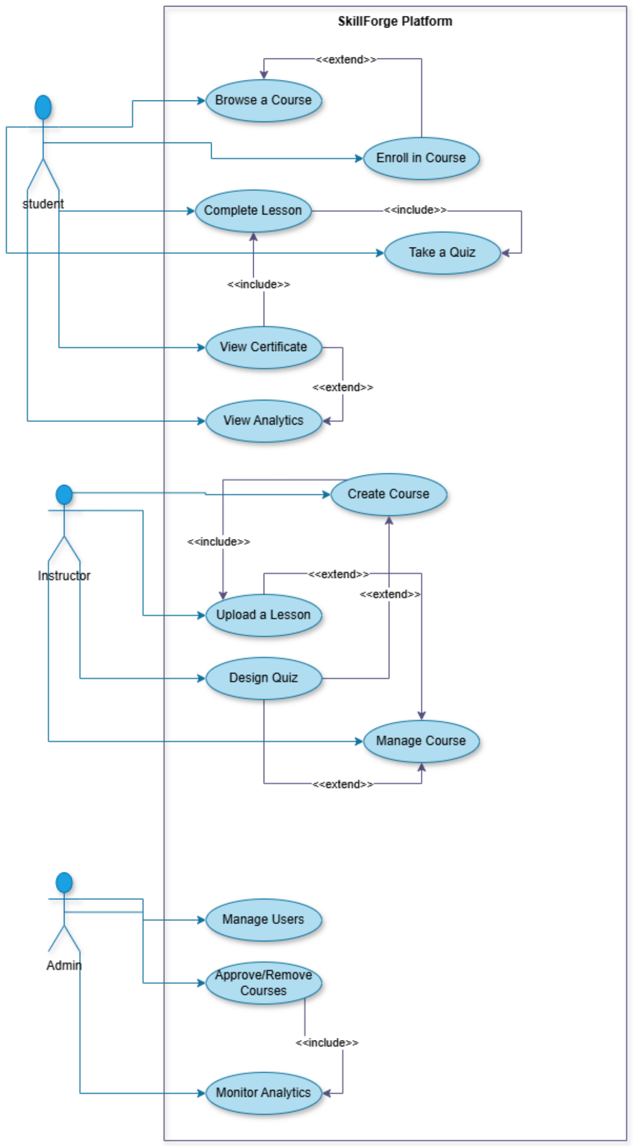

Fig3.1.use case diagram

1. Student Use Cases:

Browse a Course:
The student searches or views available courses.

Enroll in Course:
The student joins a selected course (required before accessing content).

Complete Lesson:
The student studies or finishes a lesson within the course.

Take a Quiz:
The student attempts quizzes related to lessons.

View Certificate:
After completing a course successfully, the student can view or download a certificate.

View Analytics:
The student can track learning progress, performance, ….

- **Now discussing their relationships:**

1-Enroll in Course <<extend>> → Browse a Course

→ browsing is optional, but enrolling may happen as an extension of browsing.

2-Complete Lesson <<include>> → Take a Quiz

→ completing a lesson includes taking a quiz (quiz is part of the lesson).

3-View Certificate <<include>> → Complete Lesson

→ you must complete lessons (and possibly quizzes) before you can view your certificate. (so basically, to generate a certificate you must complete the lesson and completing the lesson requires taking and passing the quiz)

4-View Certificate <<extend>> → View Analytics

→ analytics can include certificate-related progress or extend the certificate feature.

2. Instructor Use Cases:

Create Course: Instructor sets up a new course.

Upload a Lesson: Instructor adds lesson materials.

Design Quiz: Instructor creates quizzes for students.

Manage Course: Instructor updates or edits course details.

- **Now discussing their relationships:**

1-Create Course <<include>> → Upload a Lesson

→ when creating a course, uploading lessons is included. (scenario is that an instructor can’t create an empty course so that’s why it’s an include relationship)

2- Upload a lesson + Design a quiz <<extend>> → Manage course

→ managing a course may involve any of these actions (adding lessons, editing quizzes, ….)

3-Design Quiz <<extend>> →Create Course

→Designing a Quiz is optional when creating a course (scenario is that when the instructor Creates a course, he/she has the option to either include quizzes in it or no)

3. Admin Use Cases:

Manage Users: Add, remove, or edit student/instructor accounts.

Approve/Remove Courses: Approve instructor-created courses or delete inappropriate ones.

Monitor Analytics: Oversee platform-wide statistics and usage.

- **Now discussing their relationships:**

Approve/Remove Courses <<include>> → Monitor Analytics

→ analytics can be used to make informed decisions about which courses to approve or remove. (scenario is as follows → when an instructor Creates a course and wants to publish it to our platform the admin must first check the analytics of the instructor (bad reviews of inappropriate content, ….))

Second Activity Diagram:

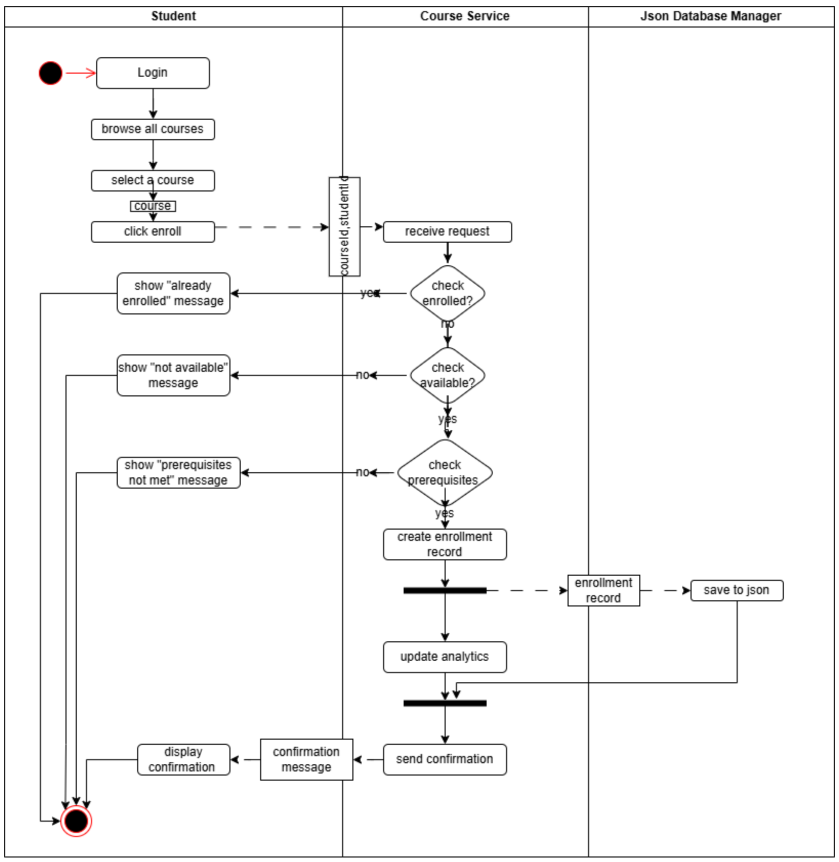

Fig.3.2. activity diagram

This activity diagram illustrates the workflow of enroll in a course use case from the system's perspective. The process begins when the **student** logs in to the platform, browses available courses, selects a desired course, and initiates the enrollment request. Once the request is made, the **Course Service** receives it and performs a series of validation steps, including checking whether the student is already enrolled, confirming course availability, and verifying prerequisite completion.

If any condition fails, the system displays an appropriate message and terminates the process. Otherwise, the **Course Service** proceeds to create an Enrollment Record, which is passed to the **JsonDatabaseManager** for storage. In parallel, the **Analytics** component updates enrollment statistics to reflect the new record. After both tasks complete successfully, the **Course Service** sends a confirmation message back to the **student**, indicating that the enrollment has been successfully processed.
Object flow represents the transfer of data objects between system components

Third Class Diagram:

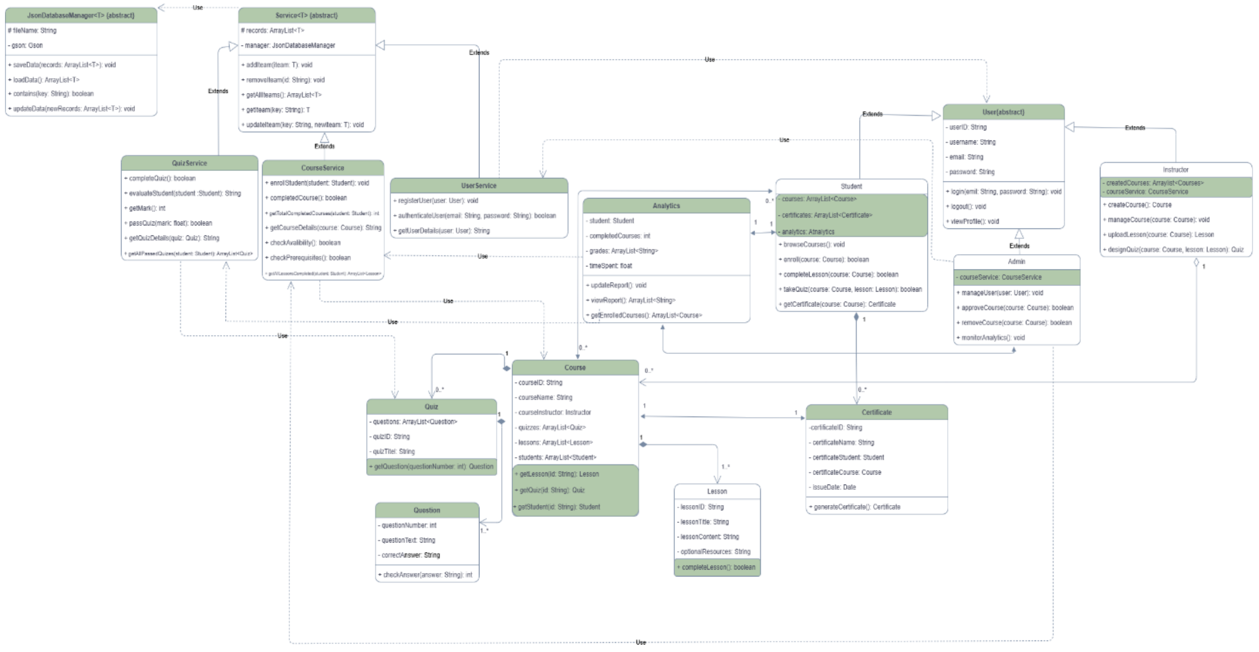
JsonDatabaseManager<T> {abstract}
# fileName: String
- gson: Gson
+ saveData(records: ArrayList<T>): void
+ loadData(): ArrayList<T>
+ contains(key: String): boolean
+ updateData(newRecords: ArrayList<T>): void
Use
Service<T> {abstract}
# records: ArrayList<T>
- manager: JsonDatabaseManager
+ addIteam(iteam: T): void
+ removeIteam(id: String): void
+ getAllIteams(): ArrayList<T>
+ getIteam(key: String): T
+ updateIteam(key: String, newIteam: T): void
Extends
Extends
Extends
QuizService
+ completeQuiz(): boolean
+ evaluateStudent(student :Student): String
+ getMark(): int
+ passQuiz(mark: float): boolean
+ getQuizDetails(quiz: Quiz): String
+ getAllPassedQuizes(student: Student): ArrayList<Quiz>
CourseService
+ enrollStudent(student: Student): void
+ completedCourse(): boolean
+ getTotalCompletedCourses(student: Student): int
+ getCourseDetails(course: Course): String
+ checkAvalibility(): boolean
+ checkPrerequisites(): boolean
+ getAllLessonsCompleted(student: Student): ArrayList<Lesson>
UserService
+ registerUser(user: User): void
+ authenticateUser(email: String, password: String): boolean
+ getUserDetails(user: User): String
Use
Use
Use
Use
Use
Use
User{abstract}
- userID: String
- username: String
- email: String
- password: String
+ login(emil: String, password: String): void
+ logout(): void
+ viewProfile(): void
Extends
Extends
Extends
Student
- courses: ArrayList<Course>
- certificates: ArrayList<Certificate>
- analytics: Atnalytics
+ browseCourses(): void
+ enroll(course: Course): boolean
+ completeLesson(course: Course): boolean
+ takeQuiz(course: Course, lesson: Lesson): boolean
+ getCertificate(course: Course): Certificate
Instructor
- createdCourses: Arraylist<Courses>
- courseService: CourseService
+ createCourse(): Course
+ manageCourse(course: Course): void
+ uploadLesson(course: Course): Lesson
+ designQuiz(course: Course, lesson: Lesson): Quiz
Admin
- courseService: CourseService
+ manageUser(user: User): void
+ approveCourse(course: Course): boolean
+ removeCourse(course: Course): boolean
+ monitorAnalytics(): void
Analytics
- student: Student
- completedCourses: int
- grades: ArrayList<String>
- timeSpent: float
+ updateReport(): void
+ viewReport(): ArrayList<String>
+ getEnrolledCourses(): ArrayList<Course>
1
1
0..*
0..*
1
0..*
0..*
1
Course
- courseID: String
- courseName: String
- courseInstructor: Instructor
- quizzes: ArrayList<Quiz>
- lessons: ArrayList<Lesson>
- students: ArrayList<Student>
+ getLesson(id: String): Lesson
+ getQuiz(id: String): Quiz
+ getStudent(id: String): Student
1
0..*
1
1
1
1
1..*
Quiz
- questions: ArrayList<Question>
- quizID: String
- quizTitel: String
+ getQuestion(questionNumber: int): Question
1
1..*
Question
- questionNumber: int
- questionText: String
- correctAnswer: String
+ checkAnswer(answer: String): int
Lesson
- lessonID: String
- lessonTitle: String
- lessonContent: String
- optionalResources: String
+ completeLesson(): boolean
Certificate
-certificateID: String
- certificateName: String
- certificateStudent: Student
- certificateCourse: Course
- issueDate: Date
+ generateCertificate(): Certificate
Use

Fig.3.3. class diagram

This class diagram represents the structural design of our system, the **SkillForge Platform**. It illustrates the main classes, their attributes, and the relationships between them, showing how the system components interact. Only the **most essential and meaningful methods** are included to highlight core functionality—common utility methods such as setters and getters were intentionally omitted to keep the diagram concise and focused on the system's logic.

The diagram is organized into **three main layers**:

- **Database Layer:**
  This layer includes the JsonDatabaseManager class, which manages all file operations such as loading, reading, and saving data in Array Lists.
- **Service and Business Layer:**
  This layer contains the service classes and the Analytics module, responsible for executing the main system operations—from adding new courses to issuing certificates—and then communicating with the database layer to update stored data.

- **Entities Layer:**
  This layer defines the data entities such as users, courses, quizzes, and certificates. It mainly handles data representation and acts as the foundation upon which the service layer operates.

In conclusion, the class diagram presents the **core architecture and data relationships** of the SkillForge platform, focusing on the **most relevant classes and methods** that enable the platform's main features.

Finally Sequence Diagram:

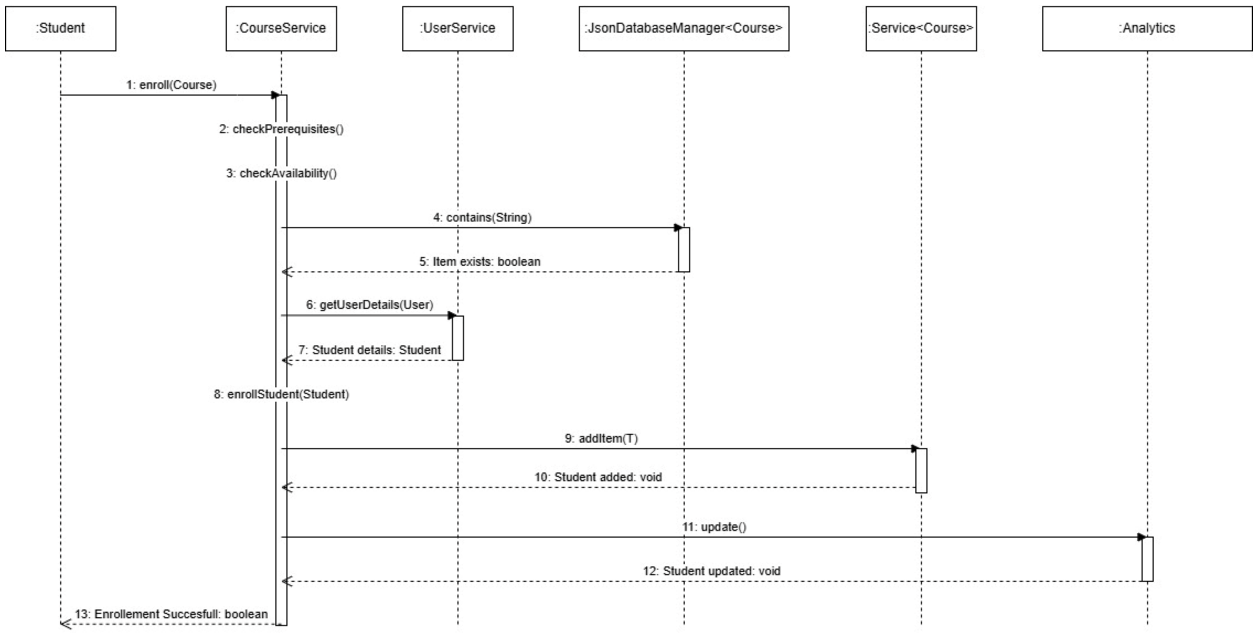

Fig.3.4. sequence diagram

This sequence diagram explains the process of student enrollment in a course from a system perspective.
When the **Student** calls the enroll (Course) method, the **Course Service** first checks if the student meets the **prerequisites** and whether the course is **available**. It then verifies the course's existence in the database by calling the contains (String) method in **JsonDatabaseManager<Course>**.

Next, the **Course Service** requests the student's information from the **User Service** using getUserDetails (User). After receiving the student details, the service proceeds with enrollStudent (Student) to register the student.

The **JsonDatabaseManager<Course>** adds the student record with addItem(T), and the **Service<Course>** updates the system data using update ().

Finally, the system confirms that the enrollment was successful and returns a boolean response to the **Student**.

4.Conclusion

This report provided a full UML analysis and design for the SkillForge system, covering the use case, activity, class, and sequence diagrams. The diagrams collectively illustrate how system components interact, how data flows, and how functionality is achieved through a layered design.

5.References

- Dr. Layla’s lectures.
- The TA’s videos on drive and YouTube