2025 / 2026

UniLife

Project Documentation

Lorenzo Cellitti

Marco Zirilli

INDEX

Contents

1. SOFTWARE REQUIREMENT SPECIFICATION

1.1 INTRODUCTION

1.1.1 AIM OF THE DOCUMENT

The purpose of this document is to outline the software requirements and architectural implementation of “UniLife”, an academic tutoring platform developed as part of the Software Engineering and Web Design course (A.Y. 2025–2026).

This document details the functional specifications and design patterns adopted to ensure a scalable and robust solution.

1.1.2 OVERVIEW OF THE SYSTEM

UniLife is designed to bridge the gap between Universities and the academic community, supporting undergraduate and graduate students throughout their entire educational path, while also assisting them in defining their future careers.

From a technical perspective, the system is a Java-based application engineered according to the Model-View-Controller (MVC) architectural pattern. The project is managed using Maven for build automation and features a Graphical User Interface (GUI) implemented in JavaFX.

The system supports the following main functionalities:

- **User Management:** Role-based authentication (Student, Tutor, University Employee).
- **Lesson Scheduling:** Creation, validation, and booking of tutoring sessions.
- **Payment Processing:** Secure integration for transaction management.
- **Session Tracking:** Handling of concurrent user sessions via token-based mechanisms.

1.1.3 HW & SW REQUIREMENTS

To ensure the proper functioning of the UniLife application for both development and end-user execution, the following environments are defined.

Software Requirements

- **Operating System:** Cross-platform compatibility (Windows 10/11, macOS 12+, Linux Ubuntu 20.04+).
- **Development Kit (JDK):** Java Development Kit 21 (LTS) or higher.
- **Build Automation:** Apache Maven 3.8+.
- **User Interface:** JavaFX SDK 21.
- **Database:** MySQL Community Server 8.0 or MariaDB 10.6+.

- **IDE (Optional):** IntelliJ IDEA or Eclipse IDE for development.
- **Version Control:** Git.

Hardware Requirements

- **Processor:** Modern Multi-core Processor (Intel Core i5 / AMD Ryzen 5 or equivalent).
- **RAM:** Minimum 8GB (16GB Recommended for development).
- **Storage:** At least 500MB of free disk space for application and local database storage.
- **Display:** Minimum resolution of 1366x768 (1920x1080 recommended for optimal GUI experience).
- **Network:** Active internet connection required for external API integrations.

1.1.4 RELATED SYSTEMS, PROS AND CONS

- **Studyportals**

  Pros:

  - User friendly interface
  - Check matching with the desired course through AI student advisor
  - Integrated student reviews
  - Chat with students

  Cons:

  - Information about admission requirements is not sufficient
  - Presence of outdated data

- **Mentor Collective**

  Pros:

  - In-app "Conversation sparks" feature, which suggests questions to ask mentors
  - Highly personalized student-mentor matching

  Cons:

  - Limited access to partner universities' students
  - No university discovery feature

1.2 USER STORIES

- **US-1:** As a student, I want to apply to a university course, so that I can progress my academic path.
- **US-2:** As a student, I want to book a tutor lesson, so that I can improve my grades.
- **US-3:** As a tutor, I want to offer tutor lessons, so that I can save more money for everyday life.
- **US-4:** As a student, I want to select the lessons available on specific days, so that I can find those that fit my schedule
- **US-5:** As a University Employee, I want to check the documents attached to an application, so that I can properly evaluate the student application.
- **US-6:** As a University Employee, I want to add a course, so that I can enrich the course catalogue of the university.

1.3 FUNCTIONAL REQUIREMENTS

- **FR-1:** The system shall provide the selection of filters in the course research.
- **FR-2:** The system shall check payments success.
- **FR-3:** The system shall send a notification to the student when his course application is evaluated with the result.
- **FR-4:** The system shall send a notification to the tutor when his lesson is booked with the purchaser's username.
- **FR-5:** The system shall provide the reservation of one or more tutor lessons.
- **FR-6:** The system shall store an application as evaluated when its assessed.

1.4 USE CASES

1.4.1 OVERVIEW DIAGRAM

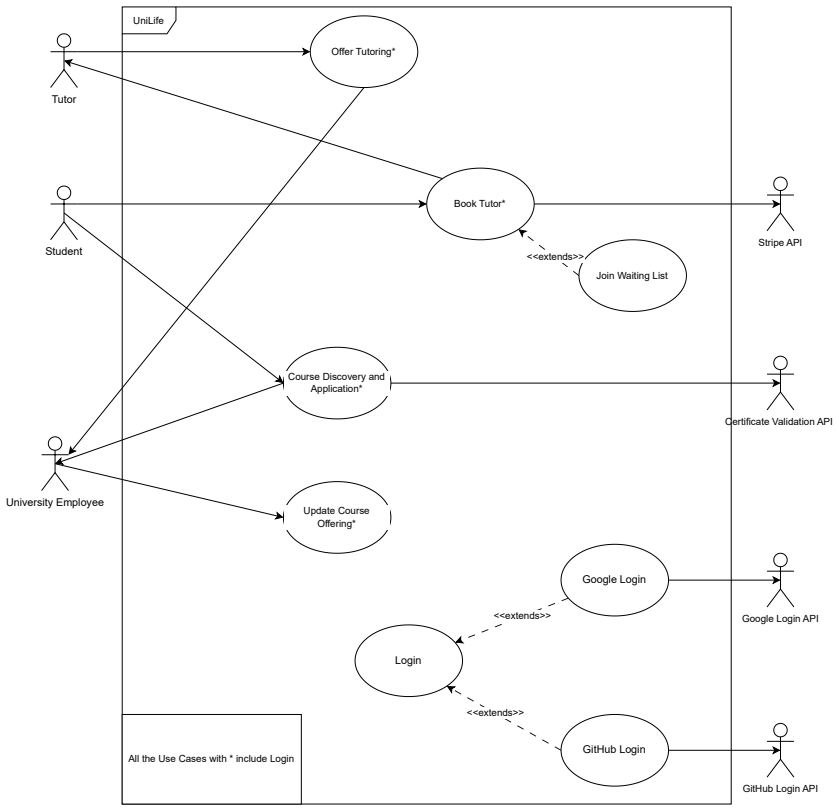


Figure 1: Use Case Overview

1.4.2 INTERNAL STEPS

UC-1: Course Discovery and Application

Main Flow:

1. The student selects the search filters.
2. The system displays the matching courses.
3. The student selects a specific course.

4. The system retrieves the course application details.
5. The student enters the requested application documents.
6. The system sends a notification to the University Employee with the application details.
7. The University Employee evaluates the application.
8. The system sends a notification to the student with the evaluation result.

Extensions:

2a. The course catalog system doesn't respond: The system displays an error message asking to try later and terminates the use case.

2b. No matching course is found: The system displays an error message and asks to select different filters.

5a. The student submits invalid documents: The system displays an error message listing the invalid documents.

UC-2: Book Tutor

Main Flow:

1. The student selects research filters.
2. The system gets matching tutor's lessons from the Tutor Catalog System.
3. The student selects an available lesson.
4. The system sends the reservation request to the tutor with the lesson details.
5. The tutor approves the request.
6. The system prepares a payment form.
7. The student selects the payment method and authorizes the payment.
8. The system validates the transaction with the Payment Gateway.
9. The system saves the schedule.

Extensions:

2a. Tutor Catalog System does not respond: The System notifies the student with an error message and terminates the use case.

2b. No matching lesson is found: The system displays an error message and asks to select different filters.

8a. Payment fails: The System notifies the Student with the failure cause and requests to retry payment or change method.

2. STORYBOARDS

The visual representation of the user interaction flows has been designed using Penpot. Below is the link to access the interactive storyboards:

- **Tool Used:** Penpot
- **Interactive Prototype:** Click here to view the Storyboards

3. DESIGN

3.1 CLASS DIAGRAM

The architectural design of UniLife adheres to standard software engineering patterns to ensure scalability, maintainability, and a clear separation of concerns.

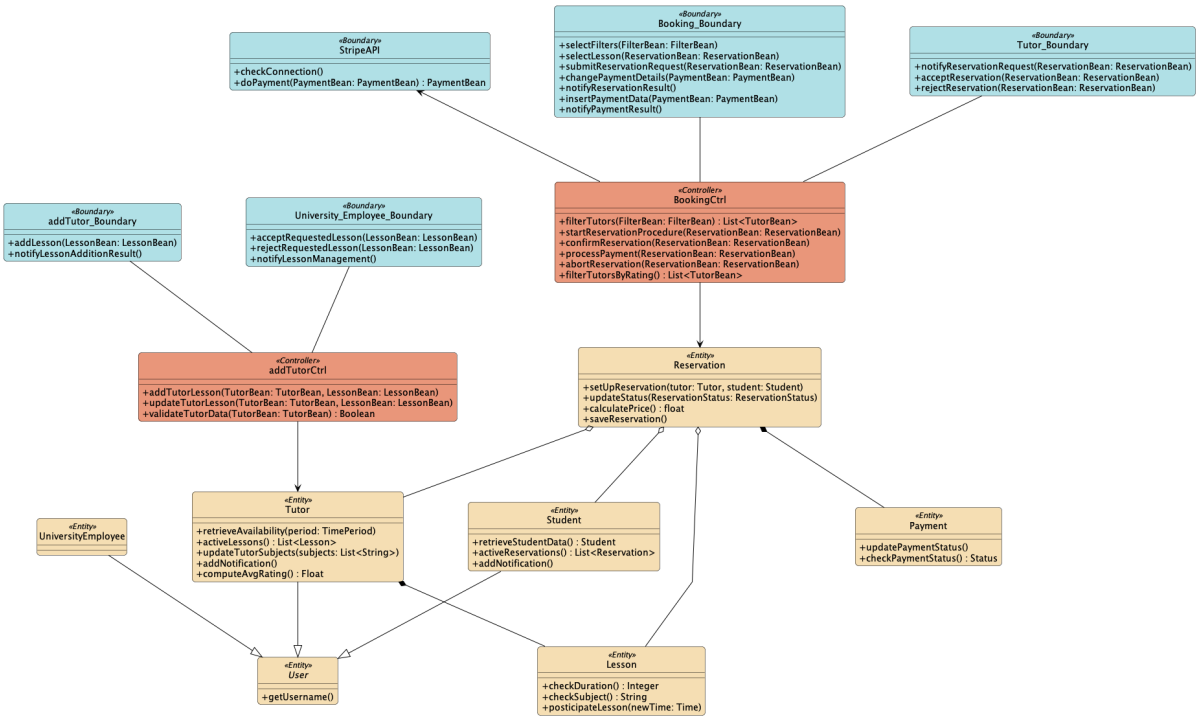


Figure 2: BCE Class Diagram: Book Tutor

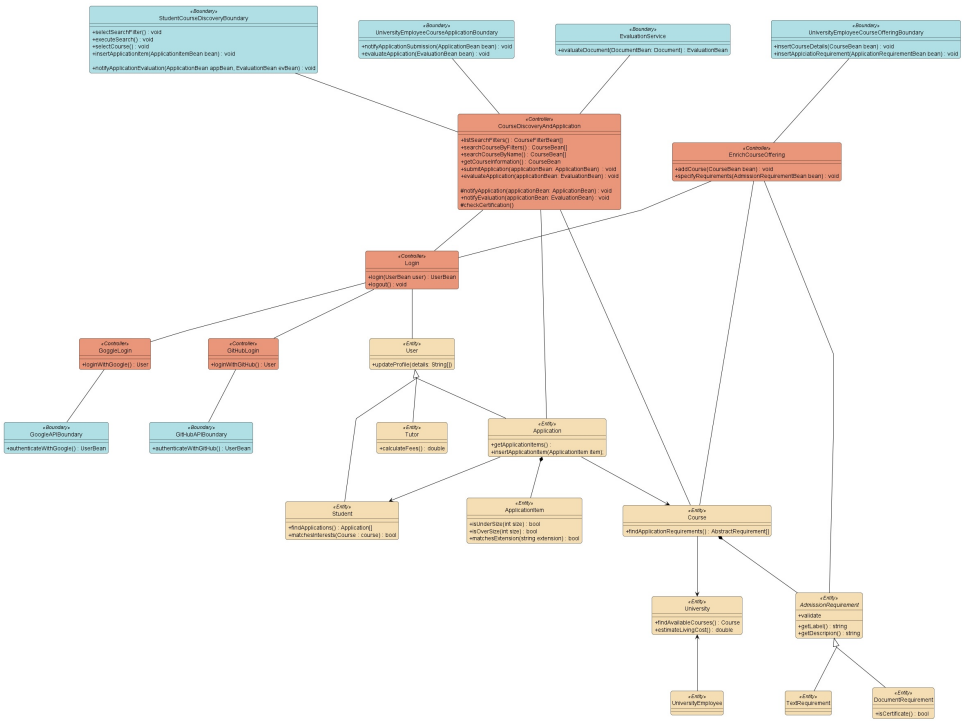
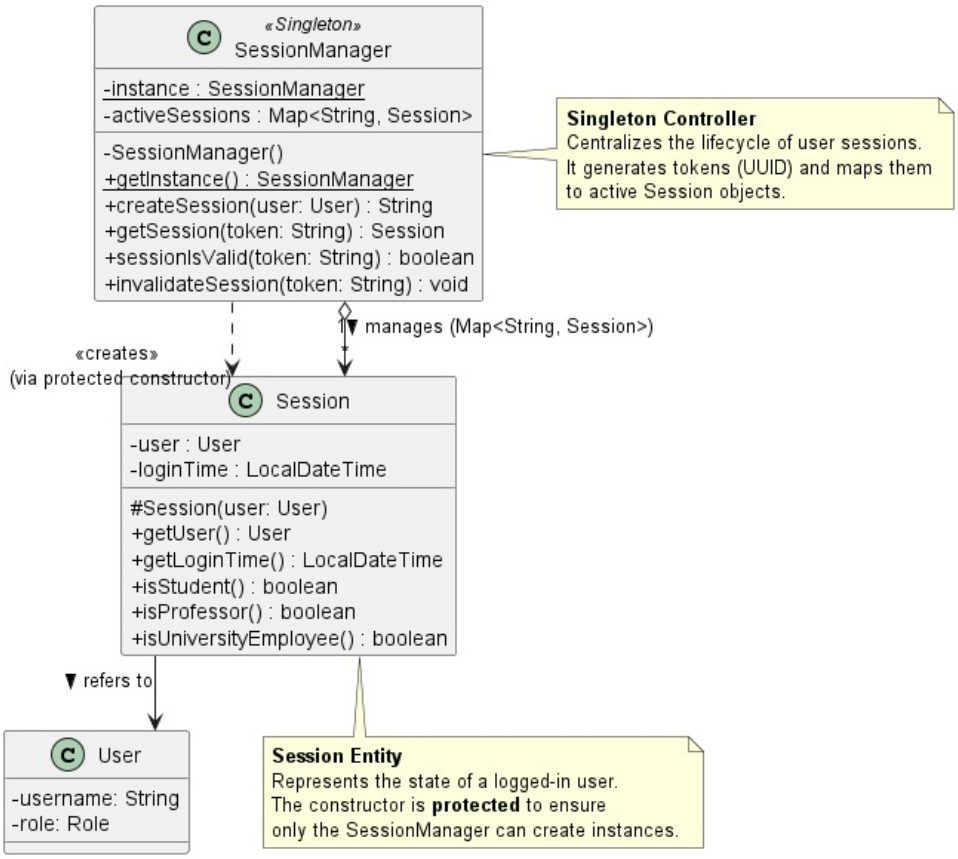


Figure 3: BCE Class Diagram: Courses

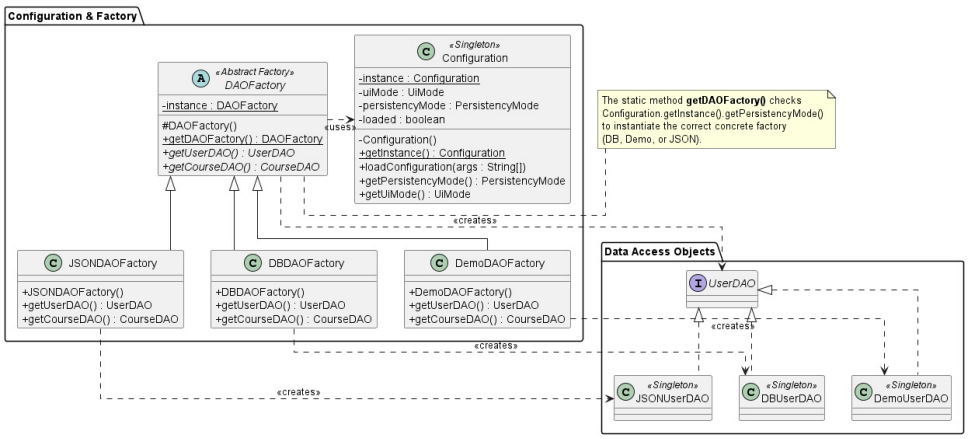
3.1.1 Design Patterns Implementation

Singleton Pattern In UniLIFE, we adopted the Singleton pattern to ensure that critical system components have a single, globally accessible instance.

- **Configuration Management:** Applied in the `Configuration` class.
- **Session Handling:** Applied to the `SessionManager`.

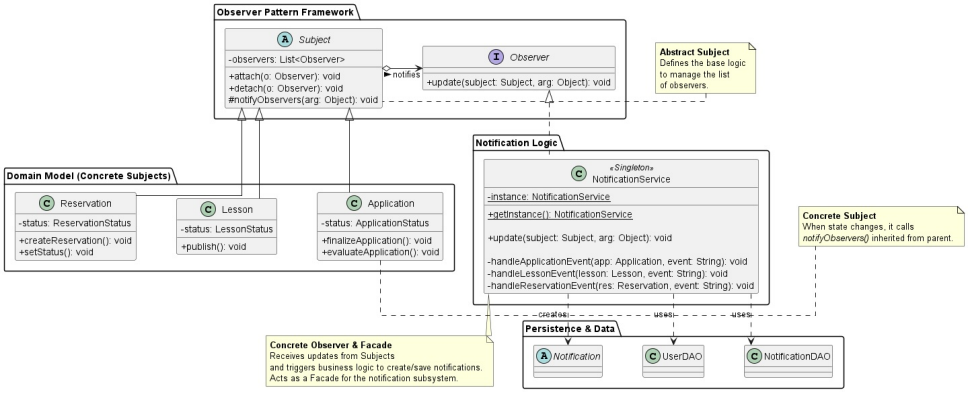


Abstract Factory Pattern We applied the Abstract Factory pattern to manage the creation of the data access layer. The `DAOFactory` acts as an abstract interface for creating families of related Data Access Objects (DAOs).



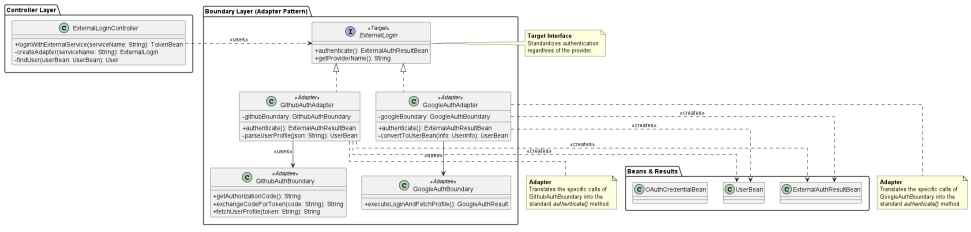
Observer Pattern We implemented the Observer pattern to handle the system's reactive notification logic.

- **Subjects:** Core domain entities act as concrete Subjects.
- **Observer:** The `NotificationService` acts as the Concrete Observer.



Facade Pattern The Facade pattern provides a simplified interface over the notification subsystem. The `NotificationService` encapsulates the complex operations required to process a notification.

Adapter Pattern We used the Adapter pattern to integrate heterogeneous external authentication services (GitHub and Google OAuth).



3.2 ACTIVITY DIAGRAM

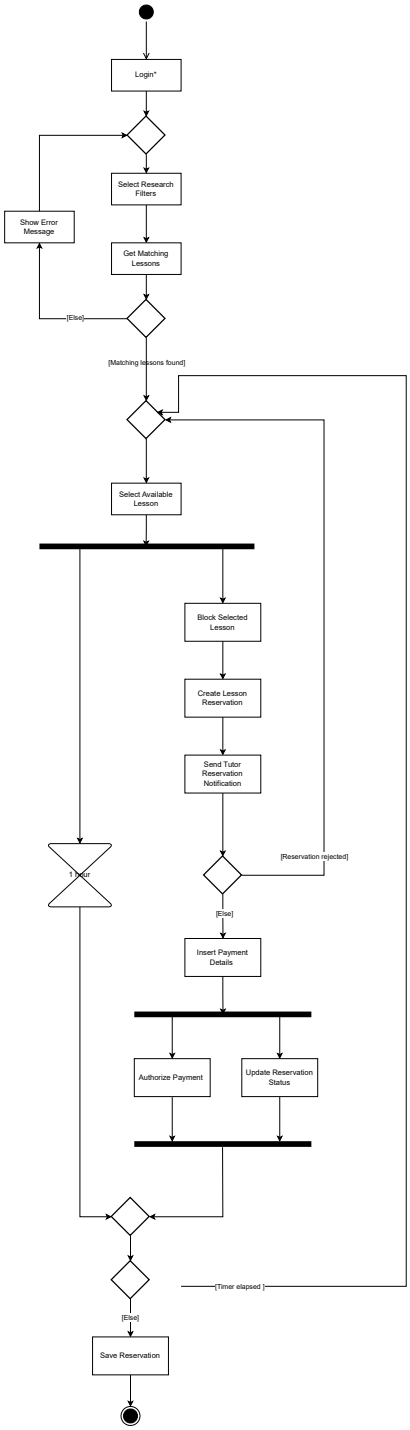


Figure 4: Activity Diagram: Book Tutor

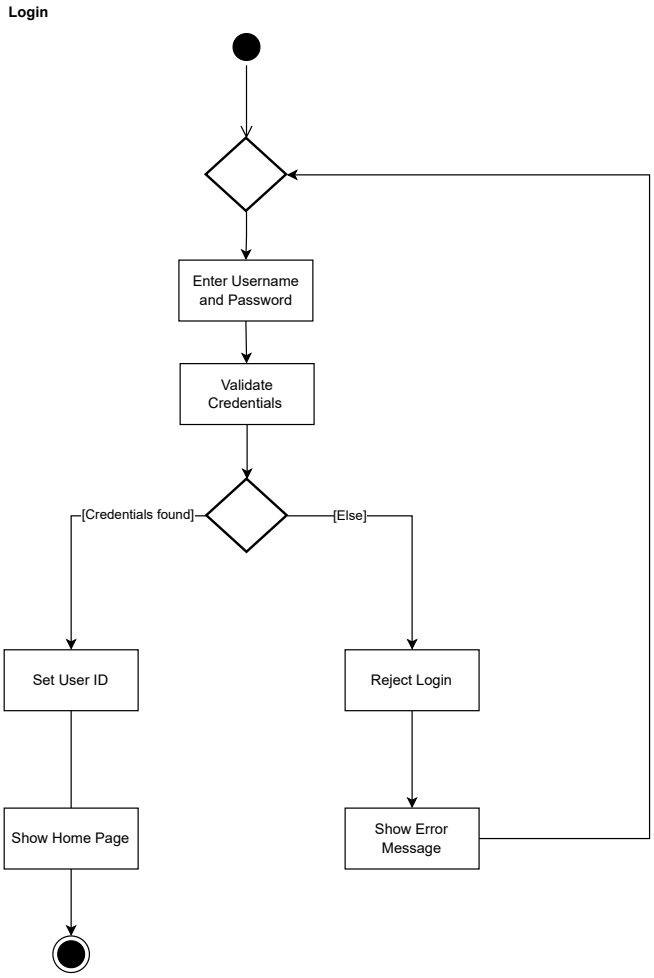


Figure 5: Activity Diagram: Login

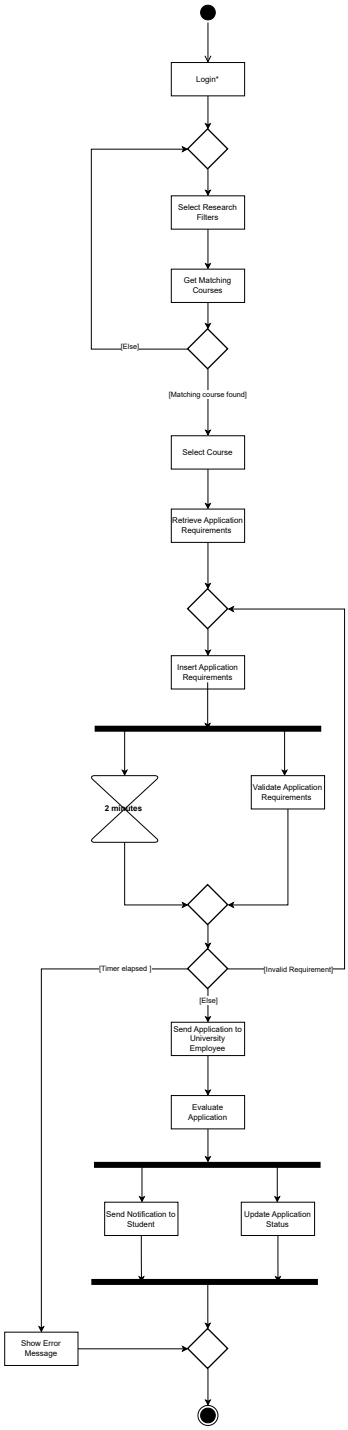


Figure 6: Activity Diagram: Course Application

3.3 SEQUENCE DIAGRAM

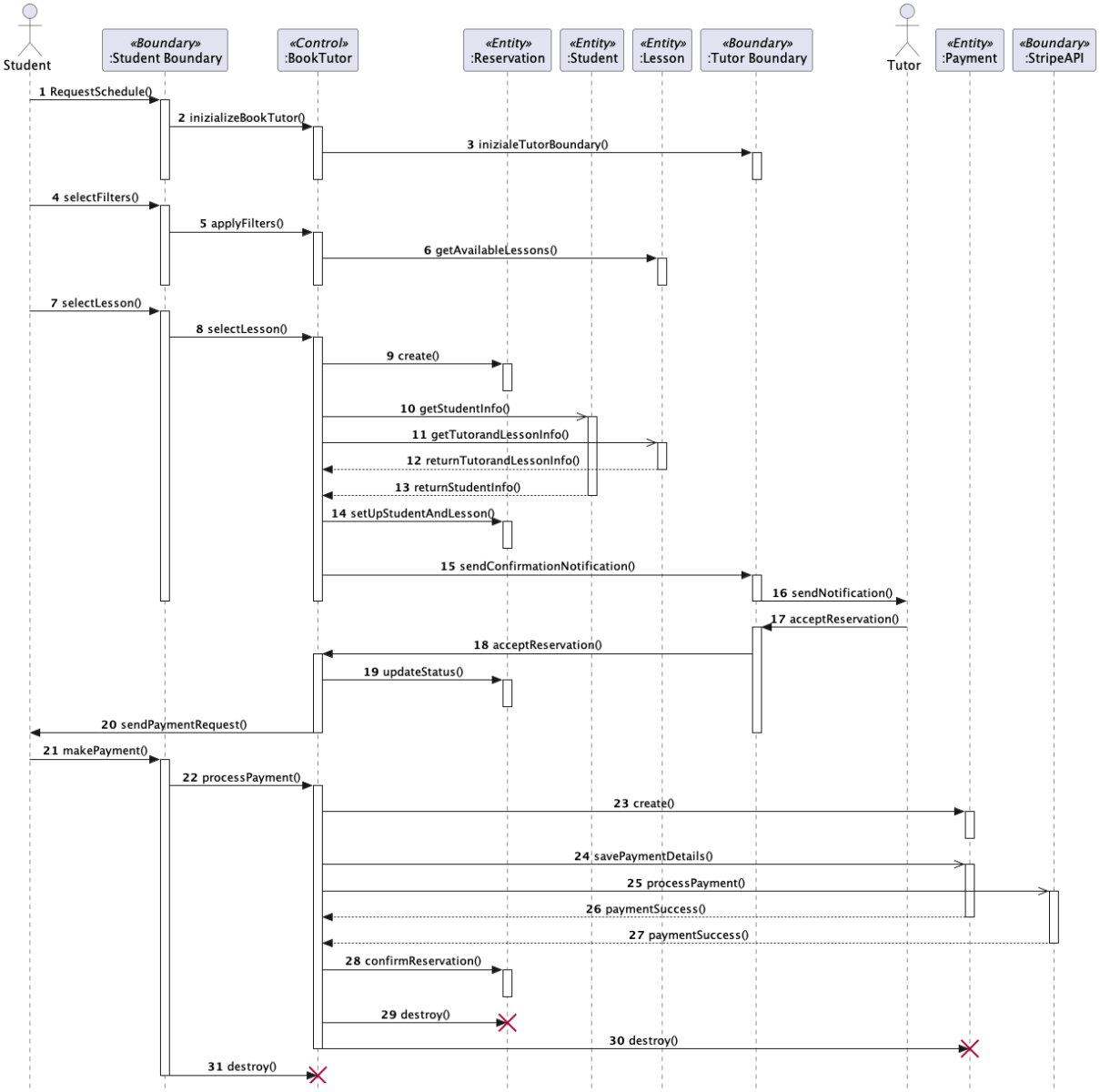


Figure 7: Sequence Diagram: Book Tutor

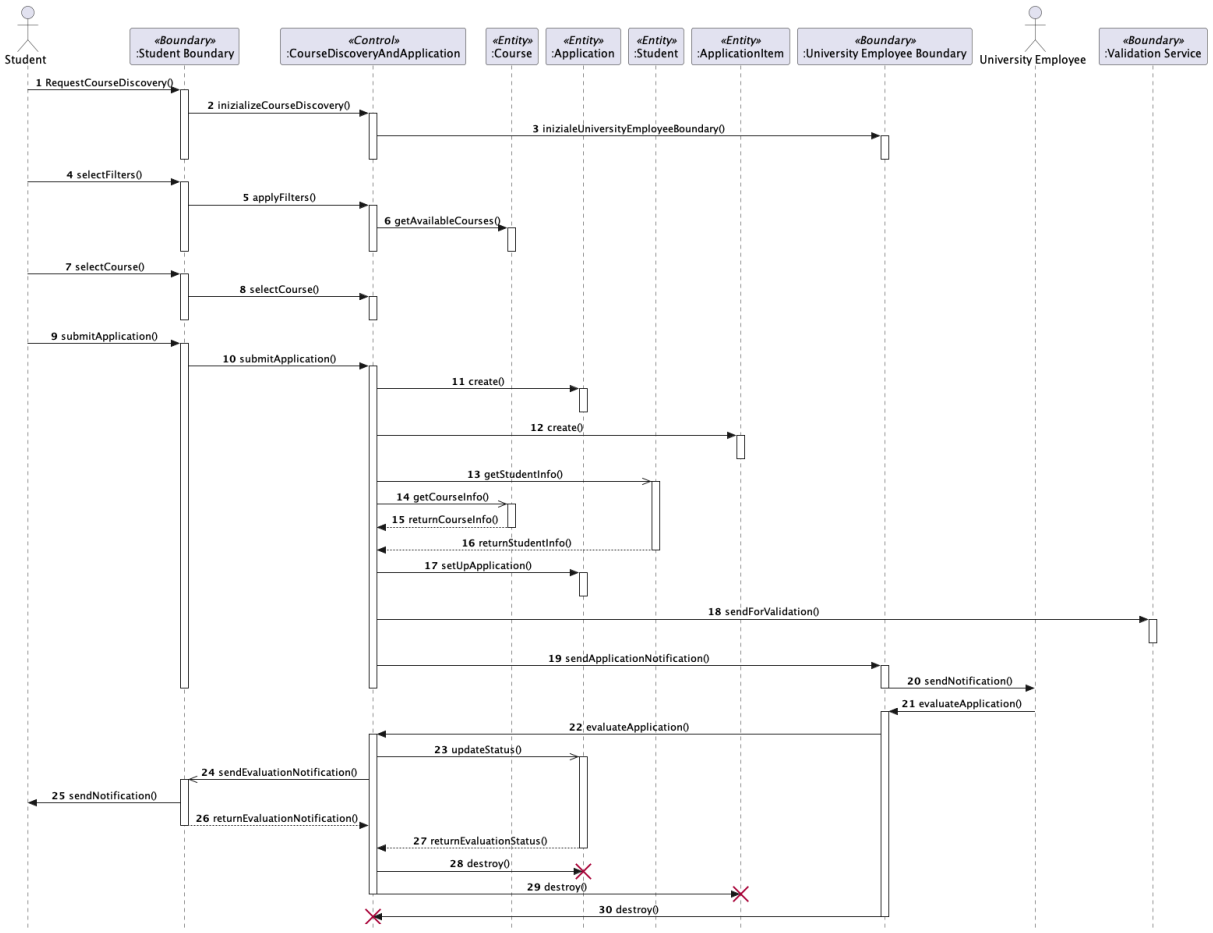


Figure 8: Sequence Diagram: Discover Courses and Application

3.4 STATE DIAGRAM

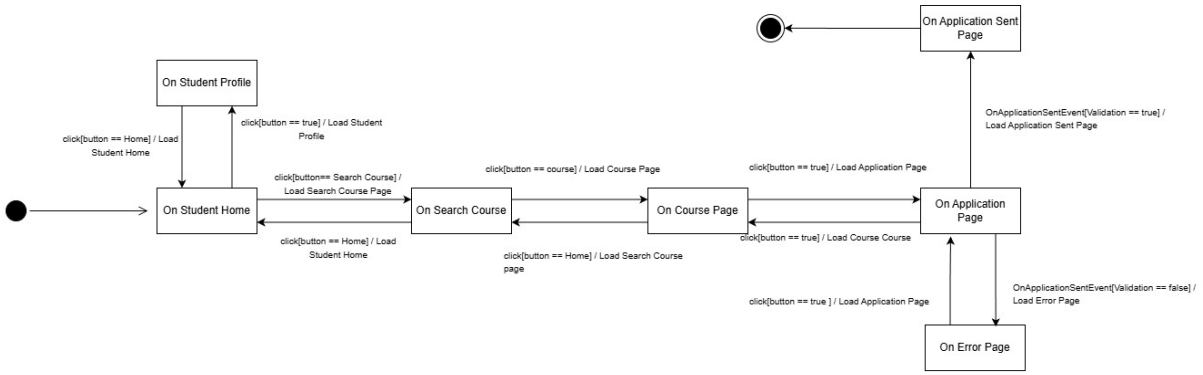


Figure 9: State Diagram: Discover Courses and Application

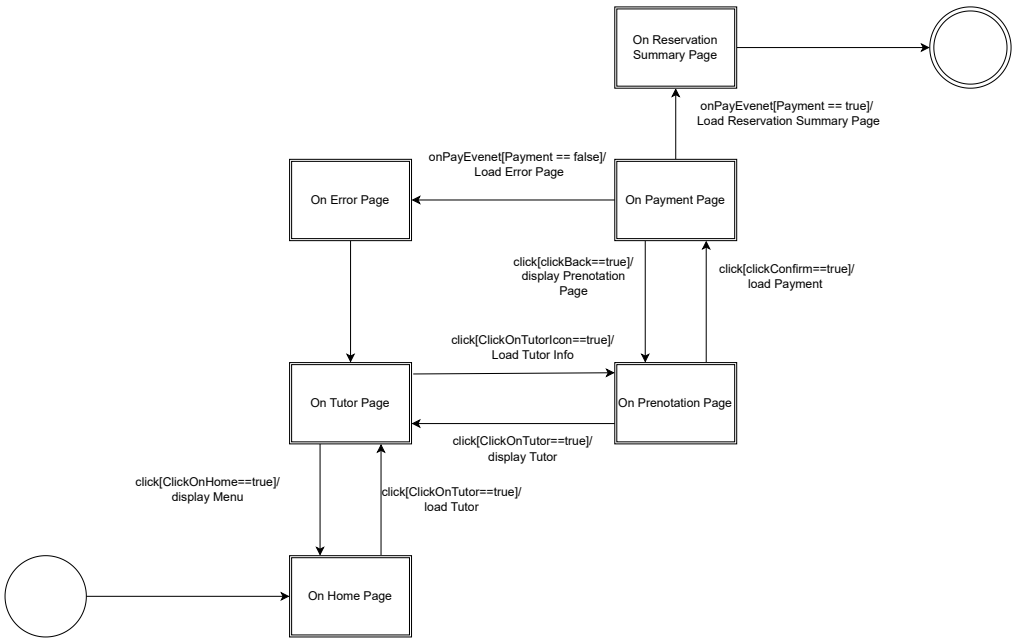


Figure 10: State Diagram: Book Tutor

4. TESTING

Course admission requirements retrieval: Marco Zirilli
We tested the controller logic responsible for retrieving admission requirements for specific courses. The tests verify that valid course identifiers correctly return the associated requirements list, ensuring data availability. Furthermore, the tests confirm that the system robustly handles queries for non-existent courses by raising the appropriate data access exceptions, preventing inconsistent states.

Course information details: Marco Zirilli
We tested the functionality allowing users to view detailed information about a selected course. The tests verify that, given a valid course and university selection, the system accurately retrieves and maps all relevant data—including specific university details and living costs—ensuring data consistency between the backend model and the returned objects.

Search filter generation: Marco Zirilli
We tested the dynamic generation of search filters based on the available course data. The tests ensure that the system correctly analyzes existing courses to categorize quantitative data, such as university rankings and course durations, into logical ranges (e.g., "Short", "Medium"), allowing the frontend to display accurate filter options.

Course search by name: Lorenzo Cellitti
We tested the search mechanism that allows users to find courses via textual queries. The tests verify that partial string matches are handled correctly, ensuring that the system retrieves all courses containing the specified search term in their title.

Course filtering by filters: Lorenzo Cellitti
We tested the filtering subsystem to ensure courses can be retrieved based on specific criteria. The tests verify that applying specific filters, such as the language of instruction, correctly narrows down the search results, returning only the courses that match the selected parameters.

Document requirement validation
We tested the validation logic for document-based admission requirements, covering file presence, size, and format. The tests verify that mandatory documents cannot be omitted and that files exceeding the maximum defined size or possessing incorrect file extensions are rejected with specific error messages. Conversely, the tests confirm that valid files within limits are accepted without issues.

Text requirement validation
We tested the validation mechanism for text-based admission inputs, such as motivation letters or biographies. The tests verify that user input adheres to defined character limits. Specifically, the system is tested to ensure it rejects text that is below the minimum required length while correctly validating text that falls within the acceptable range.

Summary of Test Organization

The tests were organized into specific test suites, separating the logic for course discovery (Controller layer) from the logic for requirement validation (Model/Bean layer) to ensure modularity and isolation of test cases.

5. CODE

The complete source code is available at the following repository:

https://github.com/Marcoos4/ISPW-UNILIFE-PROJECT.git

6. VIDEO

7. SONAR CLOUD

The code quality and static analysis have been monitored using SonarCloud.

- **Project Key:** Marcoos4_ISPW-UNILIFE-PROJECT
- **Organization:** marcoos4
- **Dashboard URL:** Click here to view the SonarCloud Dashboard