Object-Oriented Programming
Semester 2025-III

Workshop No. 2 — Object-Oriented Design (Updated Version)

Alejandro Escobar 20251020094
Jhon Gonzalez 20251020087
Sebastián Zambrano 20251020102

Computer Engineering Program
Universidad Distrital Francisco José de Caldas

1 Introduction

The **GetClasses** project is an online platform designed to connect students and tutors in a smooth, intuitive and secure learning environment. The system aims to create an ecosystem where students can easily find qualified tutors according to subject, price, or location, while tutors can promote their academic skills and manage their teaching schedules.

The motivation for this project arises from the growing demand for remote learning solutions that offer accessibility and trust. The main objective of GetClasses is to facilitate academic connection, communication, and payment between users within a unified platform. The scope includes web operations, multilingual support, and user management for tutors and students.

2 Technical Design (UML Diagrams)

The following UML class diagram represents the conceptual structure of the **GetClasses** system. It illustrates inheritance relationships, attributes, methods, and associations between main entities.

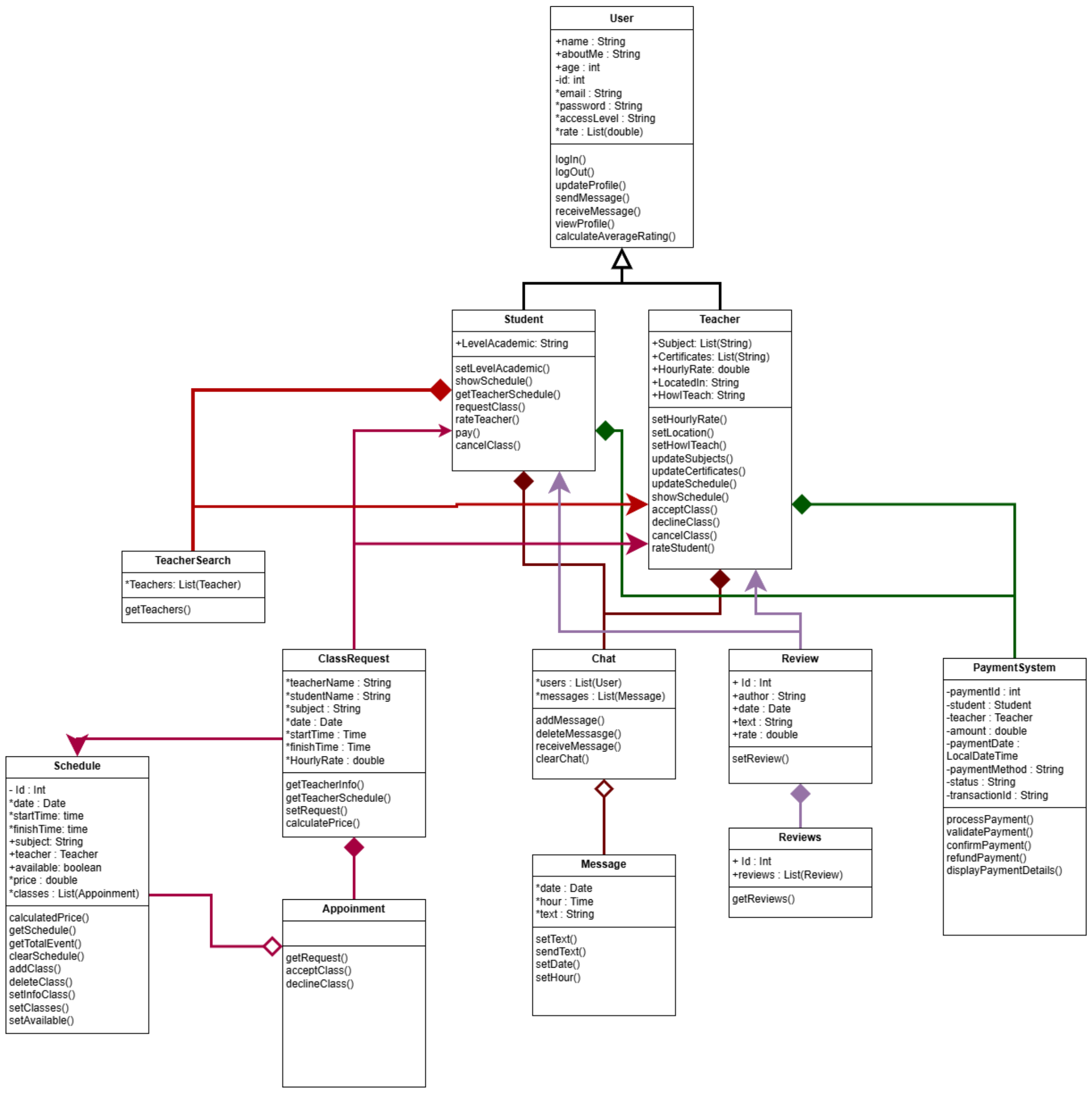


Figure 1: UML class diagram showing inheritance, composition, and relationships among system classes.

The diagram highlights class hierarchies where the base class `User` acts as the parent of specialized subclasses `StudentUser`, `TutorUser`, and `AdminUser`. Each subclass extends the base behavior to meet specific user needs, demonstrating OOP principles such as inheritance, polymorphism, and encapsulation.

3 OOP Principles in Action

Inheritance

The `User` class defines shared attributes like `id`, `name`, `email`, and `password`, and methods like `login()` and `logout()`. Subclasses such as `StudentUser`, `TutorUser`, and `AdminUser` inherit these and add their own properties and behaviors:

- **StudentUser:** Adds `level`, `bookClass()`, and `sendReview()`.
- **TutorUser:** Adds `specialty`, `availability`, and `createClass()`.
- **AdminUser:** Includes `manageUsers()` and `reviewReports()`.

Polymorphism

Polymorphism allows functions to treat subclass instances as generic `User` objects. For example, `showProfile(User u)` can call subclass-specific implementations.

Encapsulation

Private and protected attributes ensure controlled access using getters and setters, maintaining security and consistency.

System Overview

The **GetClasses** system is composed of several entities that interact to support its core functionalities. These entities include users (students and tutors), administrators, and subsystems for scheduling, payments, messaging, and reviews.

The relationships among these entities can be observed in the following system overview diagram:

This modular design allows flexibility in maintaining and scaling the platform, ensuring each module can evolve independently while maintaining integration with the overall architecture.

System Behavior and Data Flow

In **GetClasses**, the system's main operations are driven by user interactions. Students can create accounts, search for tutors, book classes, and complete payments through integrated modules. Tutors, in turn, manage their profiles, set availability, and interact with students through chat and virtual sessions. The administrator ensures compliance and resolves disputes.

The data flow of the platform is summarized as follows: users enter their data through web forms, which are processed and validated by the backend. Once authenticated, the system stores user information, manages session scheduling, and logs transaction records. Notifications are automatically triggered to keep users informed of bookings, payments, and reviews.

4 Requirements Documentation

Functional Requirements

1. Registration and login for students and tutors.
2. Profile management including experience, degrees, rate, and availability.
3. Tutor search with filters by subject, rate, language, and country.
4. Scheduling with notifications.
5. Secure payments (credit card, e-wallet).
6. Virtual classes with video call, screen sharing, and whiteboard.
7. Chat and messaging between users.
8. Reviews and ratings after sessions.
9. Dispute management by the administrator.

Non-Functional Requirements

- **Performance:** The system must respond to search queries in less than three seconds to ensure smooth navigation and usability. It should support at least 1000 concurrent users without degradation in performance.
- **Usability:** A clean and intuitive interface accessible from both web and mobile devices, supporting multiple languages (English/Spanish), to guarantee accessibility for diverse users.
- **Security:** Data encryption, credential protection, and tutor identity verification to ensure user trust and compliance with PCI DSS standards.
- **Availability:** 99% uptime is required for service reliability, supported by automated daily backups and error recovery systems.
- **Flexibility:** The architecture should allow integration of future features (e.g., group sessions or additional payment methods) without major structural changes.

5 User Stories

ID	**User Story**	**Priority**	**Effort (hrs)**	**Acceptance Criteria**
1	Tutor Registration	High	6	The tutor account is created after entering valid data.
2	Tutor Profile Picture	Medium	3	The image appears correctly on the profile.
3	Tutor Search	High	8	Tutors matching filters are shown correctly.
4	Tutor Listing	High	5	A list with tutor info is displayed.
5	Chat with Tutor	High	6	Real-time messaging works.
6	Tutor Rates Student	Medium	3	The tutor can rate the student after class.
7	Student Rates Tutor	Medium	3	The student can rate the tutor after class.
8	View Ratings	Medium	3	Ratings and comments are visible.
9	Tutor Auto-Responder	Low	2	Sends auto message when unavailable.
10	Contact Support/Admin	High	4	Support requests are sent and confirmed.

Table 1: Summary of user stories with priorities and effort.

6 CRC Cards (Tabular Format)

Class	**Responsibilities**	**Collaborators**
Tutor	Manage profile, availability, pricing, and classes.	Profile, Student, Chat, Reviews.
Student	Search tutors, schedule, rate tutors.	Tutor, Chat, Reviews, Support.
Profile	Display user info and ratings.	Tutor, Student, Reviews.
Chat	Real-time communication.	Tutor, Student.
Reviews	Show all the teacher's review.	Tutor, Student, Review.
Review	Write a review and rate.	Tutor, Student, Reviews.
ClassRequest	Request a class at a specific time, choose time, date and subject.	Tutor, Student, schedule, appointment.
Appointment	Accept or decline class request, add to the teacher's schedule.	ClassRequest, Schedule.
TeacherSearch	Search teachers with filters.	Tutor, Student.
PaymentSystem	Save banking information and redirect to the banking page.	Tutor, Student.

Table 2: CRC cards summary.

7 Mockups

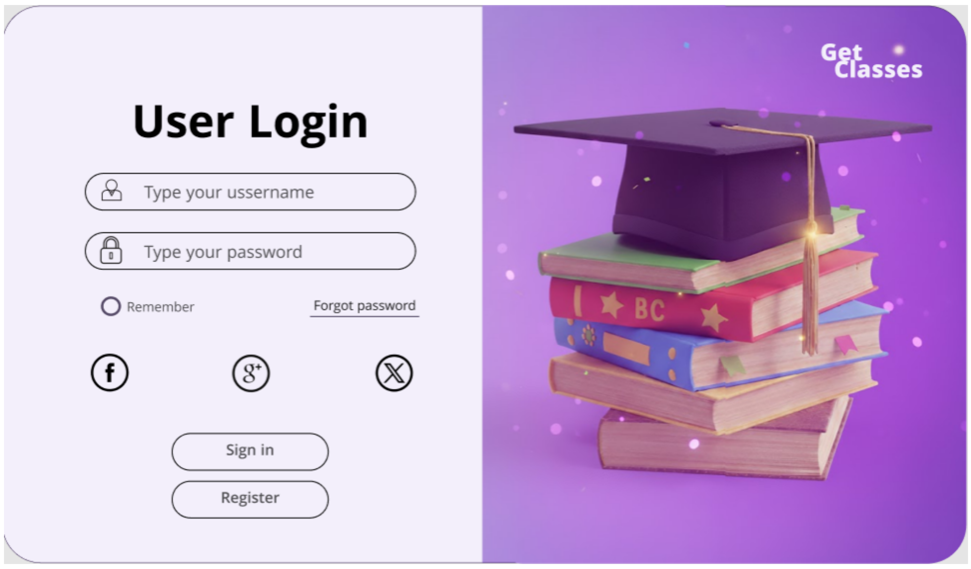


Figure 2: Login Page

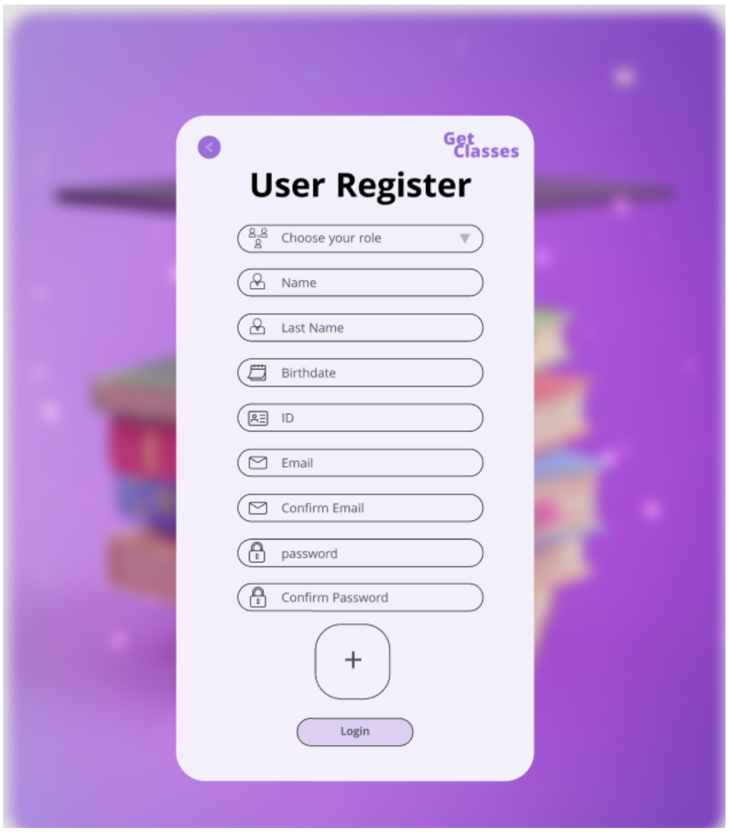


Figure 3: Register Page

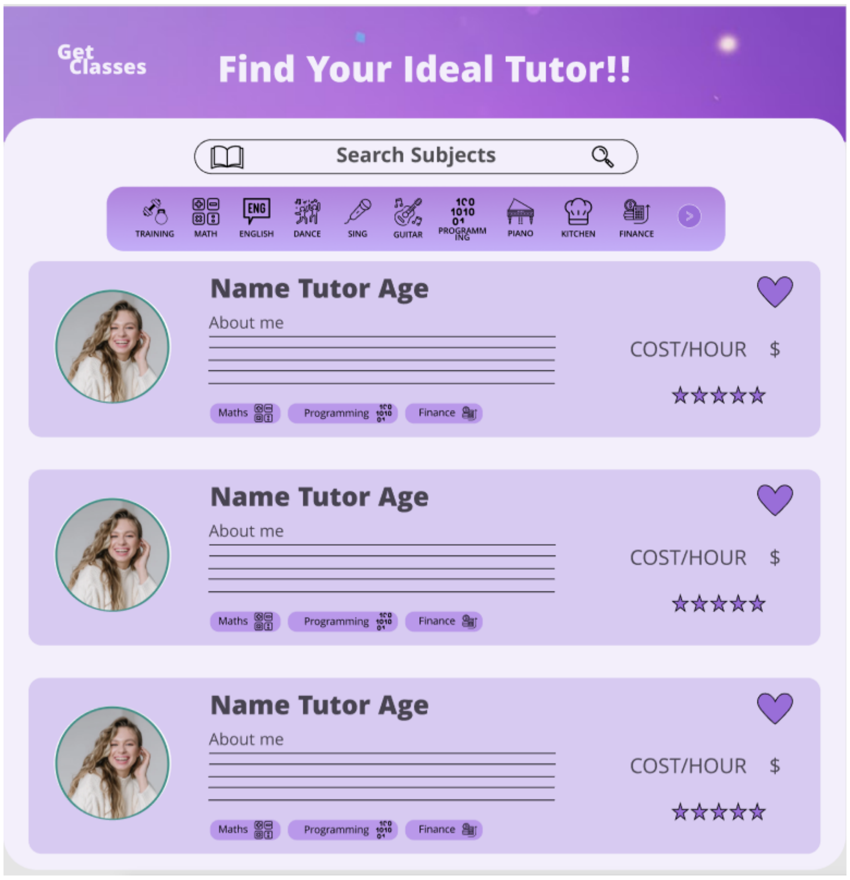


Figure 4: Main Page

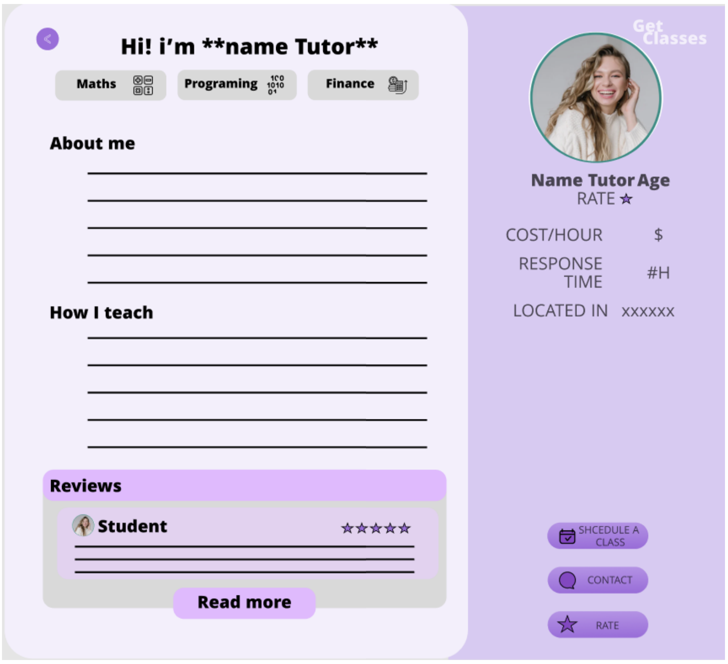


Figure 5: tutor profile Page

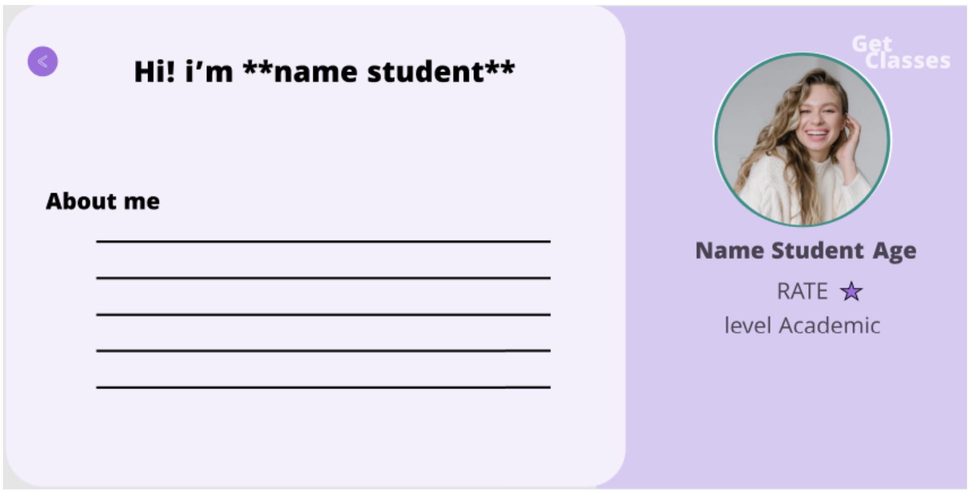


Figure 6: Student Profile Page

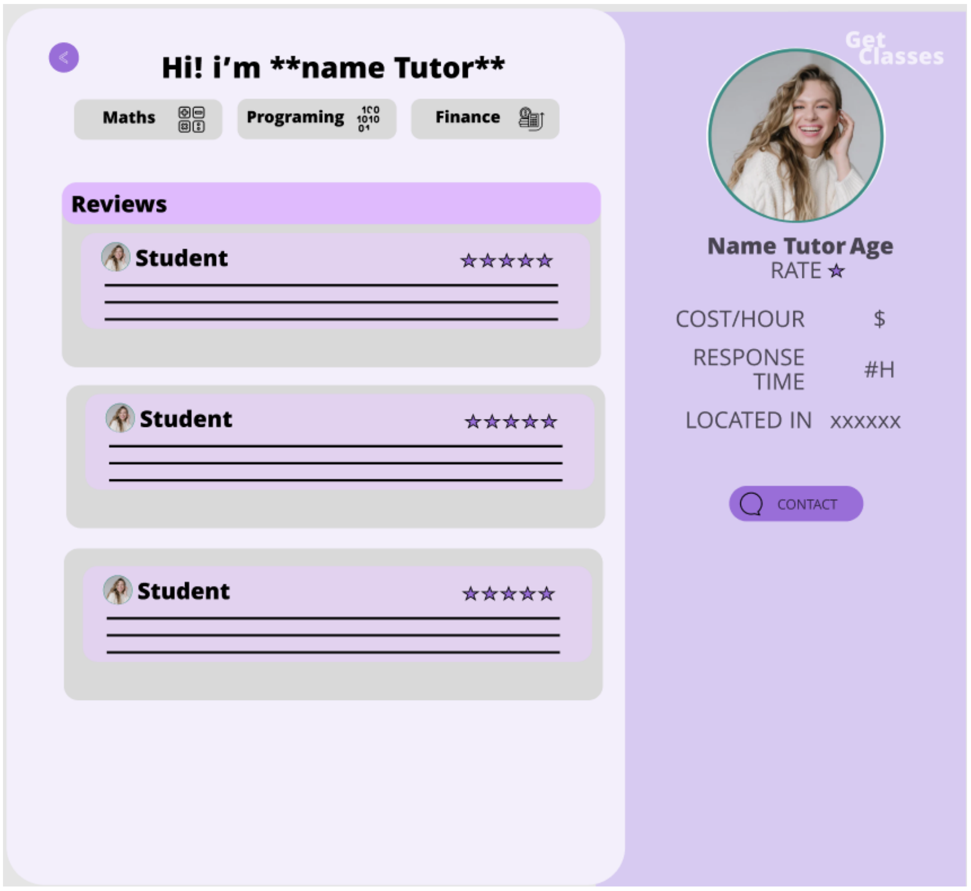


Figure 7: Review List Page

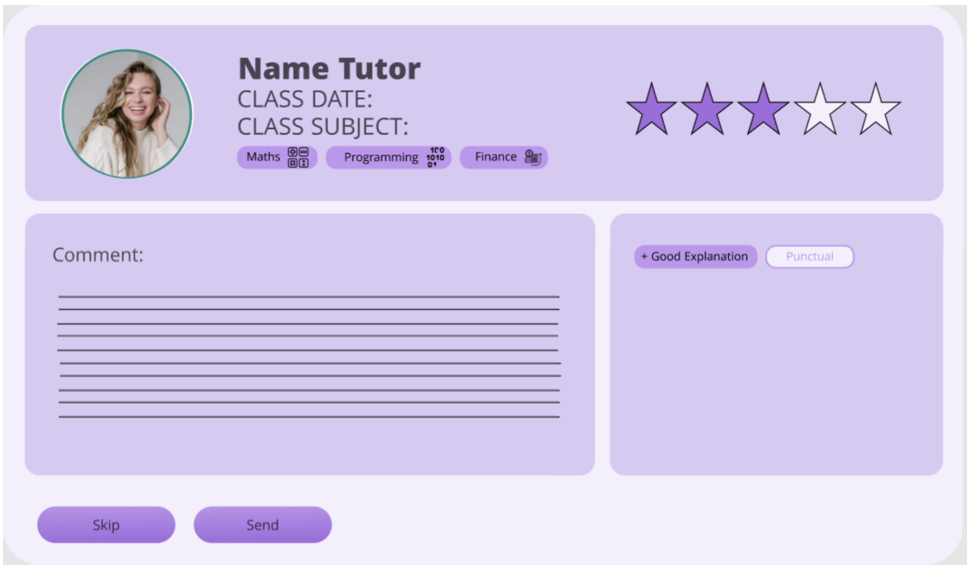
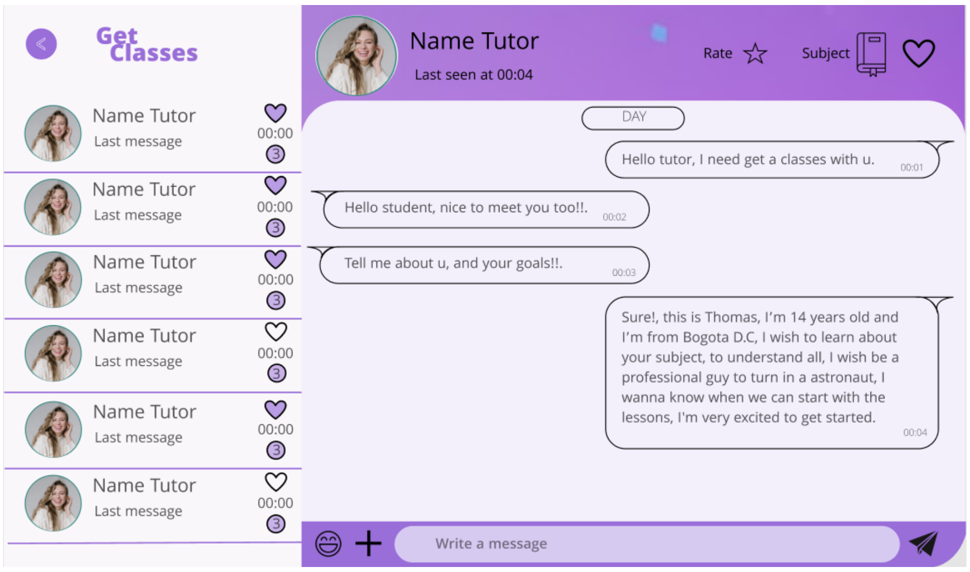


Figure 8: Make Review page



Hello tutor, I need get a classes with u. 00:01

Hello student, nice to meet you too!!. 00:02

Tell me about u, and your goals!!. 00:03

Sure!, this is Thomas, I'm 14 years old and I'm from Bogota D.C, I wish to learn about your subject, to understand all, I wish be a professional guy to turn in a astronaut, I wanna know when we can start with the lessons, I'm very excited to get started. 00:04

Figure 9: Chat Page

Notes and Reflection

Conceptual Design Updates

This updated version incorporates feedback and development from Workshop No. 1. The following updates were made:

- **Requirements:** Functional and non-functional requirements were refined to better represent real-time communication and secure payment handling.
- **CRC Cards:** Relationships between classes such as `User`, `StudentUser`, `TutorUser`, and `AdminUser` were added to emphasize inheritance and responsibilities.
- **User Stories:** Acceptance criteria were expanded, and effort and priority estimations were added for each story.
- **Design Decisions:** After planning discussions, inheritance hierarchies and encapsulated attributes were introduced to promote modularity and code reuse.

Notes

- All documentation is written in English for consistency.
- References should be added if external resources are used.
- The current version serves as a foundation for future refinement.

Reflection: During this workshop, the main focus was to define the system architecture and ensure consistency between user requirements and class design. The process strengthened our understanding of software modeling through object-oriented principles. Future improvements will include advanced UML diagrams, a more refined GUI prototype, and validation testing to verify usability and flexibility in real scenarios.

References

- Overleaf. (2024). *LaTeX tutorial: Learn LaTeX step by step.* Retrieved from https://www.overleaf.com/learn
- Lamport, L. (1994). *LaTeX: A Document Preparation System.* Addison-Wesley.
- Lucidchart. (2023). *UML Diagram Tutorial: Learn How to Draw UML Diagrams.* Retrieved from https://www.lucidchart.com/pages/uml-diagram
- Ambler, S. W. (2023). *Agile Modeling: UML Diagrams and Class Design.* Retrieved from http://www.agilemodeling.com/artifacts/classDiagram.htm
- Beck, K., & Fowler, M. (2000). *Planning Extreme Programming.* Addison-Wesley. (User Stories methodology)
- Mountain Goat Software. (2022). *User Stories and How to Write Them.* Retrieved from https://www.mountaingoatsoftware.com/agile/user-stories
- Coad, P., Yourdon, E. (1991). *Object-Oriented Design.* Prentice Hall. (CRC Cards introduction)
- Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson. (System modeling and UML principles)
- Visual Paradigm. (2023). *CRC Cards Tutorial and Examples.* Retrieved from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-crc-card/
- IEEE. (2023). *Guide to Software Design Documentation (IEEE 1016-2020).* IEEE Standards Association.