Object-Oriented Programming
Semester 2025-III

Workshop No. 1 — 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 through 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 efficiently.

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-based operations, multilingual support, and comprehensive 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 the main entities.

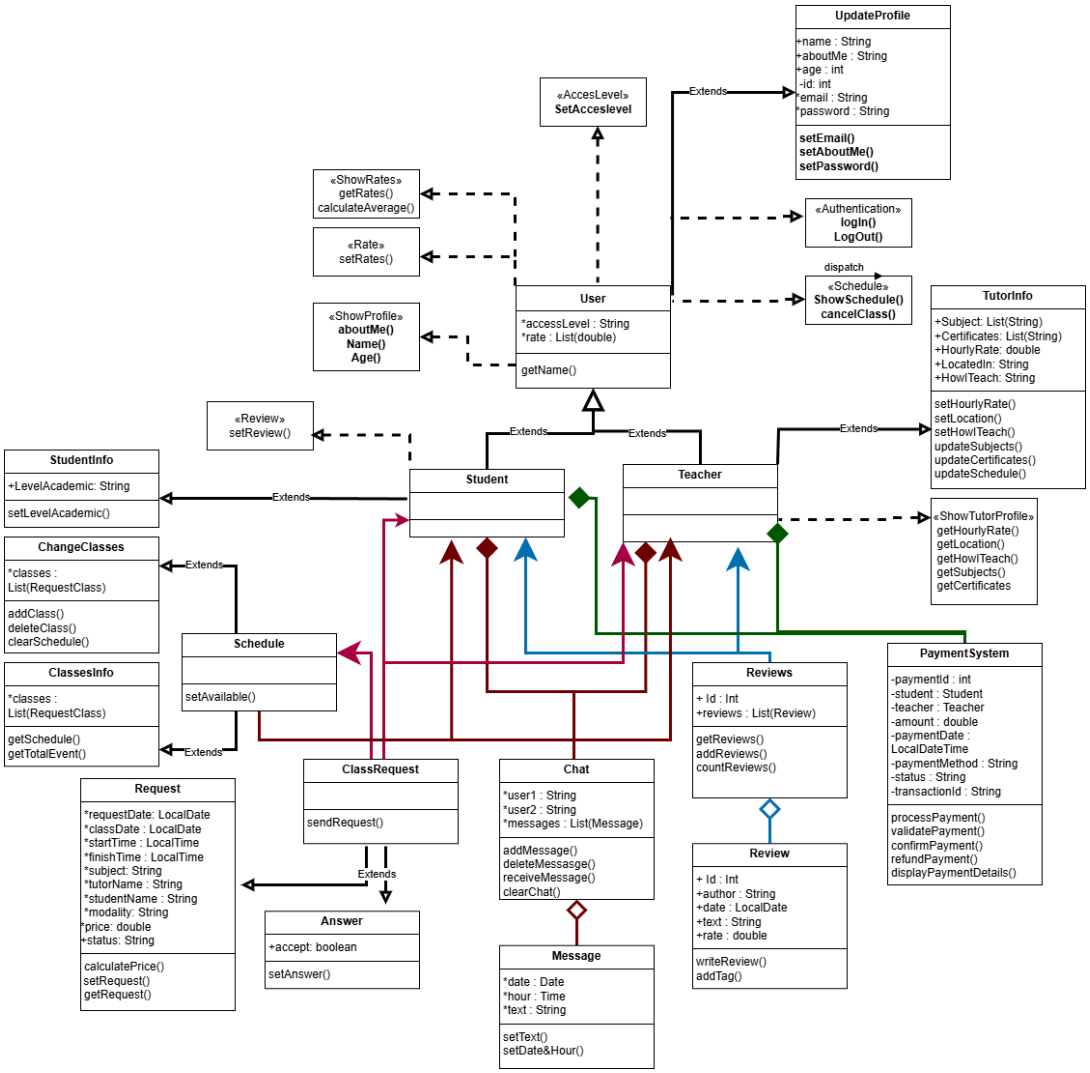

Figure 1: UML class diagram showing inheritance, composition, and relationships among system classes.

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.

SOLID application

The final UML model was thoroughly refined to ensure a rigorous application of the SOLID principles, resulting in a clearer, more modular, and extensible design. Each class was redefined with a unique and well-delimited responsibility, reducing coupling and increasing the overall cohesion of the system. The User class was simplified to represent only shared attributes and behaviors, functioning as the base entity from which Student and Teacher inherit. This guarantees compliance with the **Single Responsibility Principle (SRP)** and the **Liskov Substitution Principle (LSP)**, allowing subclasses to replace User without altering the system's behavior.

The Chat class was redesigned to handle interactions between two instances of User instead of using text strings for names, improving object relationships and future extensibility under the **Open/Closed Principle (OCP)**. Message was refined to encapsulate only the content, date, and time of the message, maintaining a single reason for change and facilitating future extensions such as message types or attachments. Similarly, Review was redefined with precise attributes (id, author, date, text, rate) and focused methods such as writeReview() and addTag(), ensuring extensibility without modifying its core structure. In addition, Reviews was introduced as an aggregator class responsible for managing multiple instances of Review, offering operations for querying and analyzing ratings in accordance with **SRP** and the Dependency **Inversion Principle (DIP)**.

On the other hand, the classes Request and Answer were introduced, from which ClassRequest inherits. Request manages information associated with class requests (dates, schedules, participants, subject, modality, price, and status), while Answer encapsulates the response (attribute accept and method setAnswer()). As a result, ClassRequest now contains only the sendRequest() method, adhering to **SRP** and reinforcing the hierarchy under **LSP** and **OCP**.

Finally, the Schedule class was simplified to handle only availability through setAvailable(), while two new classes, ChangeClasses and ClassesInfo, were added to distribute responsibilities for modification and query operations. ChangeClasses manages the addition, deletion, or clearing of classes, and ClassesInfo handles queries such as retrieving schedules or counting total events. This refactoring promotes a highly modular, clear, and extensible structure, exemplifying the application of **SRP**, **OCP**, and **DIP**, and achieving a clean, flexible, and easily maintainable architecture.

In addition, a set of interfaces was incorporated to reinforce the application of the **Interface Segregation Principle (ISP)** and the **Dependency Inversion Principle (DIP)**. Interfaces such as IShowRates, IRate, IShowProfile, IAccessLevel, IAuthentication, ISchedule, IShowTutorProfile, and IReview ensure that each class implements only the behaviors it truly requires, avoiding unnecessary dependencies and improving system flexibility. For instance, User implements interfaces related to profile visualization, ratings, and access control, while TutorInfo uses its own interface to display tutor-specific information. This strategic use of interfaces guarantees low coupling, promotes code reusability, and allows new functionalities to be added or modified without affecting existing classes, consolidating a scalable, coherent model aligned with the core principles of object-oriented design.

3 OOP Principles in Action

Inheritance

The `User` class defines shared attributes such as `id`, `name`, `email`, and `password`, as well as 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 depending on the object type.

Encapsulation

Private and protected attributes ensure controlled access through getters and setters, maintaining security and consistency across the system.

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.

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

System Behavior and Data Flow

In **GetClasses**, the main operations are driven by user interactions. Students can create accounts, search for tutors, book classes, and complete payments through integrated modules. Tutors 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 input their data through web forms, which are processed and validated by the backend. Once authenticated, the system stores user information, manages session scheduling, and records transactions. Notifications are automatically triggered to keep users informed of bookings, payments, and reviews.

4 Requirements Documentation

Functional Requirements

1. **User Registration and Authentication:** The system must allow both students and tutors to register using an email address or third-party authentication (e.g., Google). It should securely validate credentials and provide password recovery options.

2. **Profile management:** Users can manage their personal profiles, including photo, biography, experience, academic degrees, hourly rate, availability schedule, and preferred teaching subjects. Tutors must be able to verify credentials through documentation uploads.

3. **Tutor Search and Filtering:** Students must be able to search for tutors using filters such as subject, hourly rate, spoken language, country, and rating. The system should display ranked results with basic tutor information.

4. **Scheduling and Notifications:** Students can request sessions with tutors based on their availability. Both users will receive real-time notifications (email and in-app) about session confirmations, cancellations, or rescheduling.

5. **Payment Processing:** The platform must support secure online payments through credit card and e-wallets, ensuring transaction integrity and confirmation receipts.

6. **Messaging System:** Students and tutors should have access to a private chat feature for pre- and post-session communication, with message history stored securely.

7. **Reviews and Ratings:** After each completed session, students can rate their tutors and leave feedback. Ratings will be averaged and displayed publicly on tutor profiles.

8. **Dispute Resolution:** An administrator must have tools to manage user complaints and disputes, including viewing session logs and communication records for fair resolution.

Non-Functional Requirements

- **Performance:** The application must start and load its main interface in under 5 seconds on a standard computer. Core operations such as searching tutors or loading schedules should complete in under 3 seconds. It should function smoothly on systems with at least 4 GB of RAM.
- **Usability:** The interface must be clear, intuitive, and consistent, allowing users to navigate key features (registration, search, scheduling) without prior training. The design should follow standard desktop UI conventions (menus, dialogs, toolbars). Language support in English is required; Spanish may be added optionally.
- **Security:** User credentials must be encrypted or hashed before being stored locally. Communication features (e.g., messaging or class scheduling) should not expose private data. Any payment simulation or sensitive operation should be represented through mock interfaces for safety.
- **Availability:** The system should operate without crashes or data loss during typical use. Session and profile data must be saved persistently in a local database or file system. A manual backup option should be available to prevent accidental data loss.
- **Flexibility:** The architecture should allow gradual integration of advanced features (e.g., online payments, video calls, or multi-language support) in future versions, while keeping the current version lightweight and functional for local use.

5 User Stories

ID: 1	**Tutor Registration**
Priority:	High
Effort (hrs):	6
Description:	As a tutor, I want to register on the platform by entering my personal and professional information so that I can offer tutoring sessions.
Acceptance Criteria:	The tutor account is successfully created after entering valid information.

Table 1: User Story 1 – Tutor Registration

ID: 2	**Tutor Profile Picture**
Priority:	Medium
Effort (hrs):	3
Description:	As a tutor, I want to upload a profile picture so that students can identify me easily.
Acceptance Criteria:	The uploaded image appears correctly on the tutor's profile page.

Table 2: User Story 2 – Tutor Profile Picture

ID: 3	**Tutor Search**
Priority:	High
Effort (hrs):	8
Description:	As a student, I want to search for tutors by subject, rate, or language so that I can find the best match for my learning needs.
Acceptance Criteria:	Tutors matching the selected filters are displayed correctly in the search results.

Table 3: User Story 3 – Tutor Search

ID: 4	**Tutor Listing**
Priority:	High
Effort (hrs):	5
Description:	As a student, I want to view a list of available tutors so that I can compare their profiles and select one.
Acceptance Criteria:	The system displays a complete list of tutors with names, subjects, and ratings.

Table 4: User Story 4 – Tutor Listing

ID: 5	**Chat with Tutor**
Priority:	High
Effort (hrs):	6
Description:	As a student, I want to chat in real time with tutors so that I can clarify doubts before booking a session.
Acceptance Criteria:	The chat allows real-time message exchange between tutor and student.

Table 5: User Story 5 – Chat with Tutor

ID: 6	**Tutor Rates Student**
Priority:	Medium
Effort (hrs):	3
Description:	As a tutor, I want to rate students after a session to maintain platform quality and accountability.
Acceptance Criteria:	The tutor can submit a rating and review after completing a class.

Table 6: User Story 6 – Tutor Rates Student

ID: 7	**Student Rates Tutor**
Priority:	Medium
Effort (hrs):	3
Description:	As a student, I want to rate tutors after a session so that other users can make informed decisions.
Acceptance Criteria:	The student can rate the tutor and leave a comment after class.

Table 7: User Story 7 – Student Rates Tutor

ID: 8	**View Ratings**
Priority:	Medium
Effort (hrs):	3
Description:	As a user, I want to view tutor and student ratings so that I can evaluate trust and performance.
Acceptance Criteria:	Ratings and feedback are visible and linked to corresponding user profiles.

Table 8: User Story 8 – View Ratings

ID: 9	**Tutor Auto-Responder**
Priority:	Low
Effort (hrs):	2
Description:	As a tutor, I want to enable an auto-responder when I am unavailable so that students receive immediate feedback.
Acceptance Criteria:	The system automatically sends a pre-defined message when the tutor is offline.

Table 9: User Story 9 – Tutor Auto-Responder

ID: 10	**Contact Support/Admin**
Priority:	High
Effort (hrs):	4
Description:	As a user, I want to contact platform support or administrators to report issues or request assistance.
Acceptance Criteria:	Support requests are sent successfully and confirmation is received.

Table 10: User Story 10 – Contact Support/Admin

6 CRC Cards (Tabular Format)

Class: ProfileUpdate	
Responsibility	**Collaborators**
Register the basic profile data	User

Table 11: ProfileUpdate CRC

Class: User	
Responsibility	**Collaborators**
Representing the general data and behaviors of a user within the system	ProfileUpdate Teacher Student

Table 12: User CRC

Class: TutorInfo	
Responsibility	**Collaborators**
Manage the data that only the tutor should have	Teacher

Table 13: TutorInfo CRC

Class: StudentInfo	
Responsibility	**Collaborators**
Manage the data that only the student should have	Student

Table 14: StudentInfo CRC

Class: Teacher	
Responsibility	**Collaborators**
Base of the tutor user; inherits attributes and methods from User	User TutorInfo Reviews ClassRequest Schedule PaymentSystem

Table 15: Teacher CRC

Class: Student	
Responsibility	**Collaborators**
Base of the student user; inherits attributes and methods from User	User StudentInfo Reviews ClassRequest Schedule PaymentSystem

Table 16: Student CRC

Class: PaymentSystem	
Responsibility	**Collaborators**
Handle payments	Teacher Student

Table 17: PaymentSystem CRC

Class: Reviews	
Responsibility	**Collaborators**
Add, delete or edit tutor reviews	Review Student Teacher

Table 18: Reviews CRC

Class: Review	
Responsibility	**Collaborators**
Create the review and assign a rating	Reviews

Table 19: Review CRC

Class: Chat	
Responsibility	**Collaborators**
Store and represent chat messages	Student Teacher Message

Table 20: Chat CRC

Class: Message	
Responsibility	**Collaborators**
Create and edit the message before sending	Chat

Table 21: Message CRC

Class: Request	
Responsibility	**Collaborators**
Create the class request	ClassRequest

Table 22: Request CRC

Class: Answer	
Responsibility	**Collaborators**
Establish the tutor's response to the class request	ClassRequest

Table 23: Answer CRC

Class: ClassRequest	
Responsibility	**Collaborators**
Send and handle class requests	Student Teacher Answer Request

Table 24: ClassRequest CRC

Class: ChangeClasses	
Responsibility	**Collaborators**
Add and remove classes from the schedule	Schedule

Table 25: ChangeClasses CRC

Class: ClassesInfo	
Responsibility	**Collaborators**
Show the information of assigned classes	Schedule

Table 26: ClassesInfo CRC

Class: Schedule	
Responsibility	**Collaborators**
Set availability and manage the base schedule	Teacher Student ChangeClasses ClassesInfo

Table 27: Schedule CRC

7 Mockups

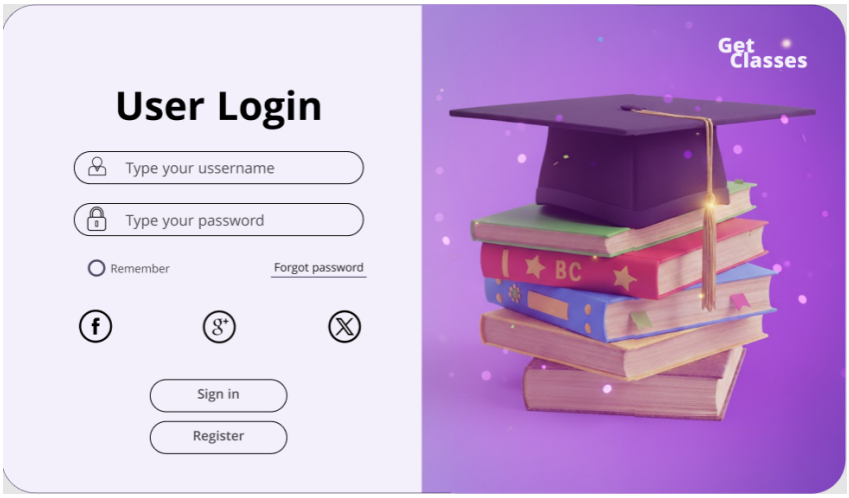

Figure 2: Login Page

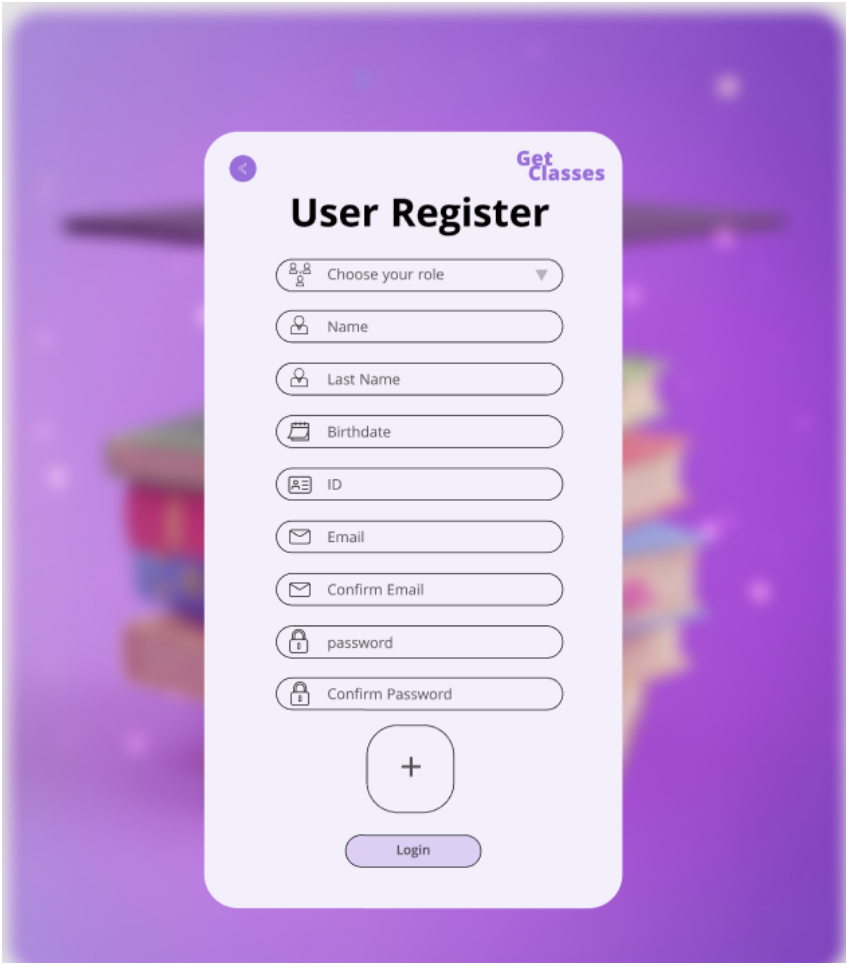

Figure 3: Register Page

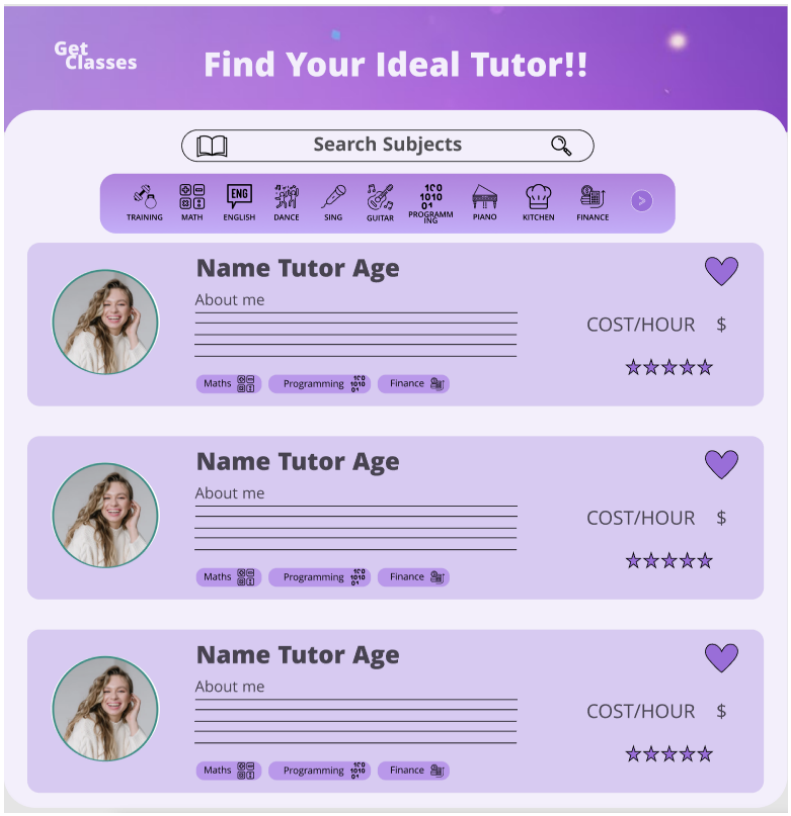

Figure 4: Main Page

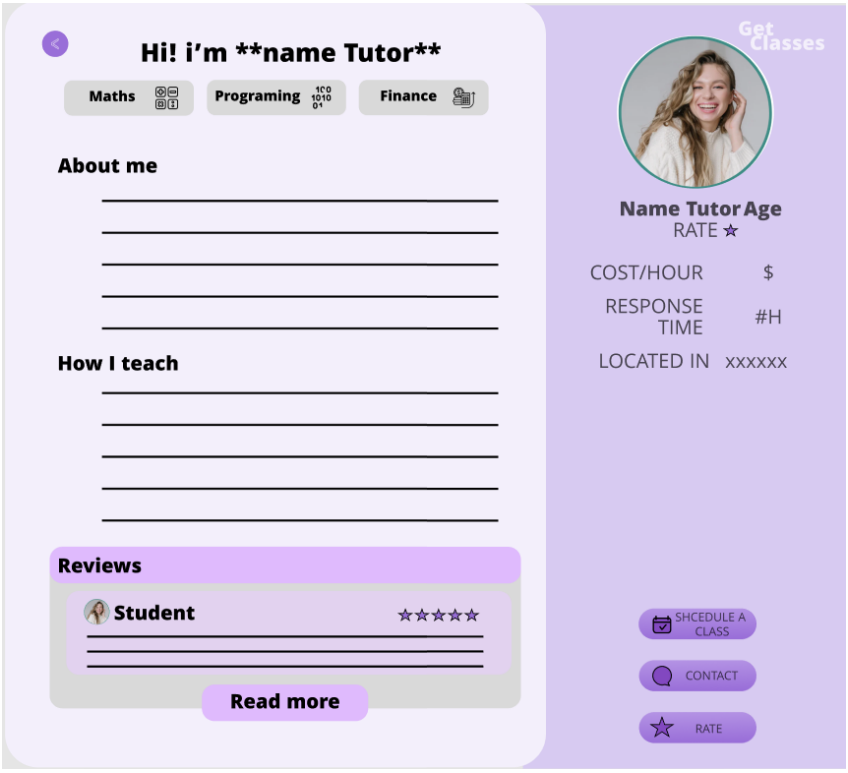

Figure 5: Tutor Profile

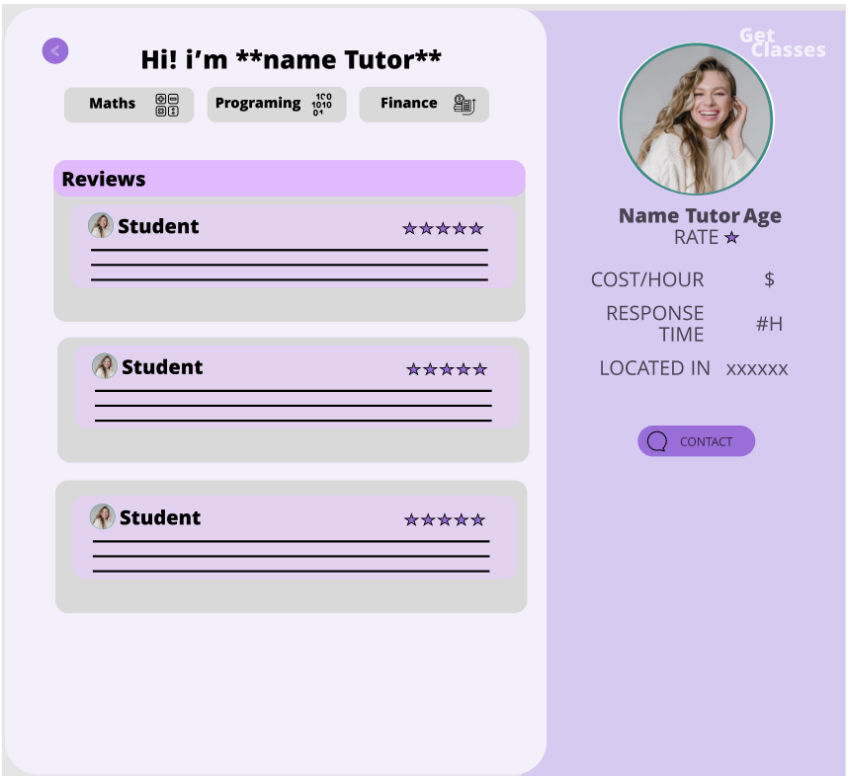

Figure 6: Reviews

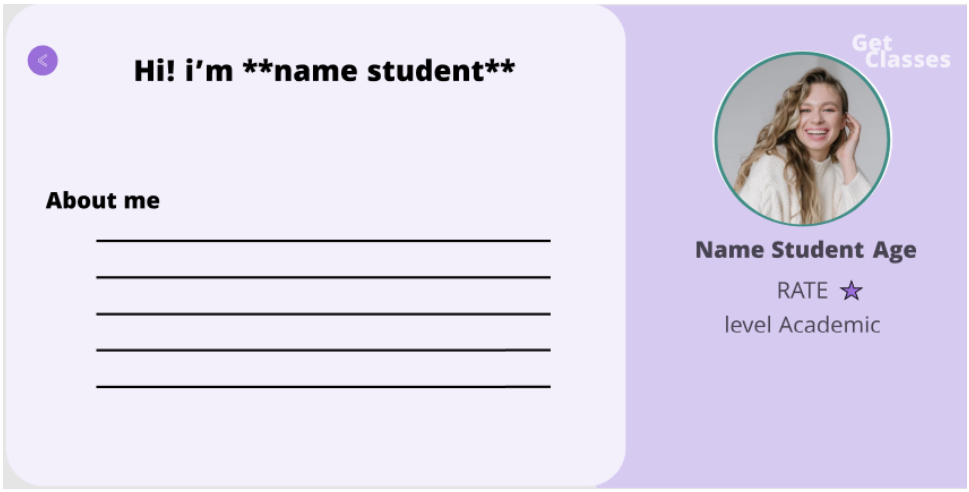

Figure 7: Student Profile

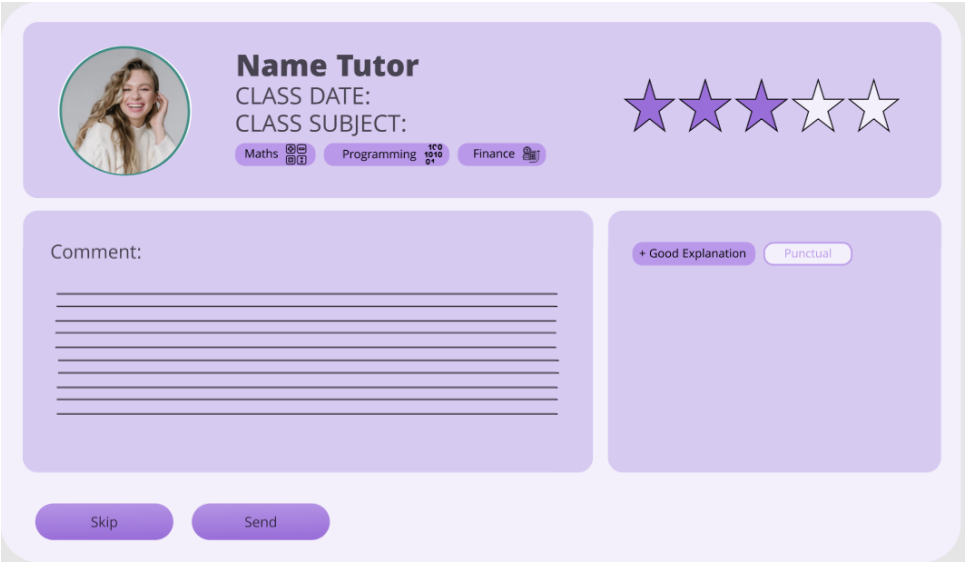

Figure 8: Make Review

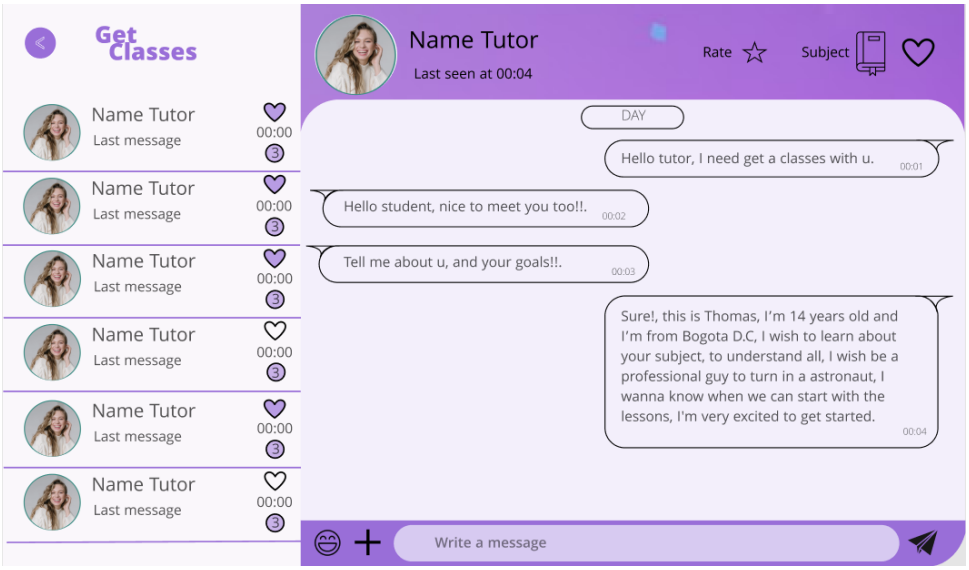

Figure 9: Chat Page

8 code snippets

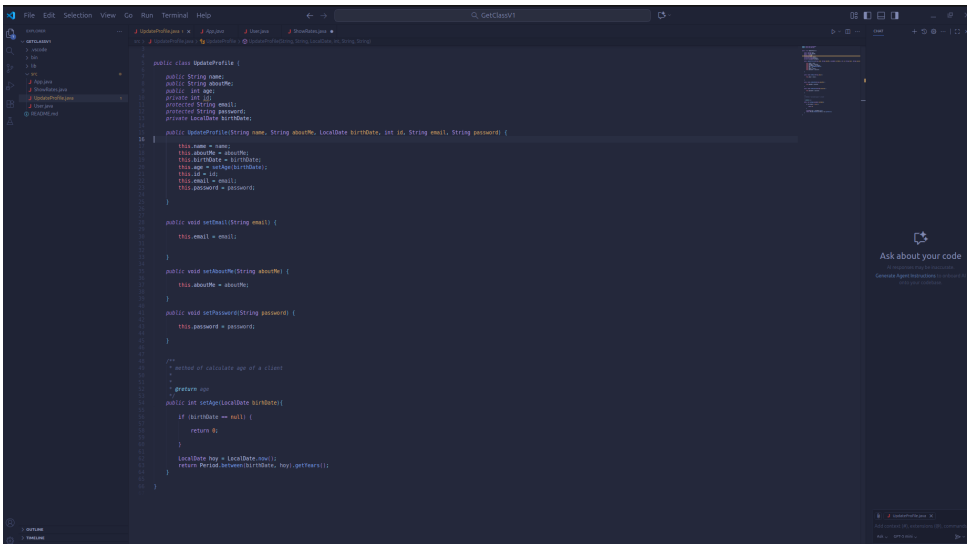

Figure 10: UpdateProfile Code

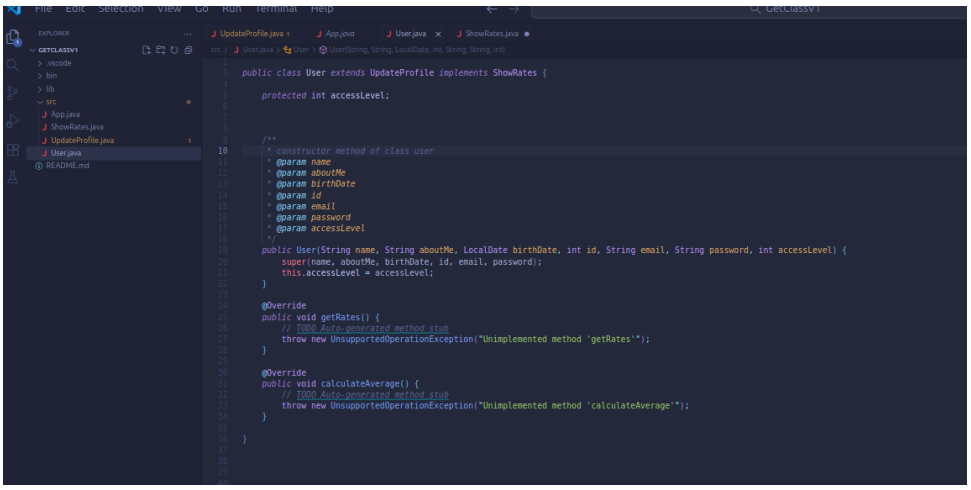

Figure 11: User Code

Figure 12: ShowRate Code

Notes and Reflection

Conceptual Design Updates

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

- **Requirements:** Functional and non-functional requirements were refined to better represent real-time communication and secure payment processing.
- **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 both effort and priority estimations were added for each story.
- **Design Decisions:** Inheritance hierarchies and encapsulated attributes were introduced to promote modularity and code reuse.

Notes

- All documentation is written in English for consistency and clarity.
- References should be added if external resources are used.
- This version serves as the foundation for future refinements and implementations.

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.