Technical Report – GetClasses
Academic Tutoring Platform

Jhon Gonzalez 20251020087
Alejandro Escobar 20251020094
Sebastian Zambrano 20251020102

Computer Engineering Program
Universidad Distrital Francisco José de Caldas
December 10, 2025

Abstract

The GetClasses platform is a lightweight academic tutoring system designed to connect students with qualified monitors and tutors through an accessible, secure, and easy-to-use environment. This technical report documents the conceptual design, layered architecture, and implementation strategies used to construct the application, emphasizing clarity, low coupling, and maintainability.

The project adopts object-oriented principles and SOLID design guidelines to ensure a flexible and extensible system capable of supporting user registration, profile management, session scheduling, reviews, and basic communication features. A JavaFX monolithic architecture was selected to provide a simple and intuitive structure that facilitates rapid development and straightforward integration between the GUI, business logic, and data persistence layers.

This report also presents updated UML diagrams, mockups, file-based storage mechanisms, and preliminary code implementations that illustrate the evolution of the system throughout the workshop series. The central outcome highlights that the GetClasses platform successfully addresses the need for an organized and efficient academic monitoring tool, supporting both tutors and students while laying the groundwork for future enhancements such as advanced communication features, analytics, and improved transactional workflows.

1 Introduction

The GetClasses platform is a digital academic support system designed to address a growing institutional need: providing students with fast, reliable, and structured access to academic monitoring services. In many universities, tutoring and monitoring programs are essential for improving student performance, yet they often suffer from fragmented communication channels, inconsistent scheduling, and limited visibility of qualified academic monitors. These inefficiencies create barriers that affect learning continuity and reduce the overall effectiveness of academic reinforcement programs.

GetClasses emerges as a solution to this problem by offering a centralized, intuitive, and business-oriented platform that connects students with certified academic monitors across different subjects. The system enables students to search for monitors based on expertise, availability, and pricing, while monitors receive a professional tool to manage sessions, showcase their qualifications, and build credibility through reviews. The platform seeks to optimize how academic support is requested, delivered, and evaluated within an educational ecosystem.

The justification for this project is driven by high student demand for accessible academic assistance, the institutional need for structured monitoring programs, and the growing adoption of digital solutions in higher education. As universities expand remote and hybrid learning environments, the need for a reliable, trackable, and user-friendly academic support system becomes even more critical.

From a technological standpoint, GetClasses is developed using a layered architecture that enhances maintainability and role separation. It incorporates object-oriented design and SOLID principles to ensure a scalable core that can grow as new services are integrated. Its JavaFX monolithic implementation supports simplicity and usability, allowing the platform to serve as a practical academic tool while supporting future extensions such as analytics, automated session tracking, or advanced communication systems.

Overall, GetClasses provides an effective business-driven response to an educational challenge, offering a structured solution that benefits both students and academic monitors while strengthening institutional support mechanisms.

2 Literature Review

The development of academic support platforms such as *GetClasses* is grounded in several areas of research, including digital learning environments, student–tutor interaction, usability design, and software architecture for educational systems. This literature review summarizes key findings and theoretical foundations that inform the design, structure, and functional requirements of the system.

2.1 Digital Learning and Academic Support Systems

Studies in modern higher education highlight the increasing relevance of digital tools to facilitate academic tutoring and peer-assisted learning. According to research on remote learning environments, students benefit from platforms that provide accessible academic reinforcement, structured communication channels, and transparent evaluation methods [**?**]. Educational platforms such as Moodle, Coursera, and peer tutoring systems demonstrate the effectiveness of centralized digital environments for connecting learners with subject experts. These platforms also emphasize the importance of user experience and intuitive design to increase engagement and reduce friction during academic support processes.

2.2 Tutor–Student Interaction and Learning Outcomes

The interaction between students and tutors plays a critical role in academic performance. Research in tutoring methodologies shows that timely access to qualified mentors significantly improves comprehension, motivation, and retention [**?**]. Effective tutoring systems promote structured scheduling, clear communication, and feedback mechanisms—elements adopted in the *GetClasses* application through rating systems, session management, and profile transparency. Studies on peer monitoring within universities further show that students prefer platforms where monitors demonstrate credibility through qualifications, reviews, and teaching history.

2.3 User Experience, Interface Design, and HCI Principles

Human–computer interaction (HCI) principles guide the implementation of intuitive interfaces in educational applications. Literature on usability emphasizes minimalistic design, reduced cognitive load, and consistent navigation as key determinants of user satisfaction. JavaFX, the framework selected for this project, aligns with these principles through reusable components, clear event-driven models, and simple layout management. The mockups and GUI prototypes for *GetClasses* reflect best practices derived from UX research, aiming for clarity, accessibility, and streamlined workflows that support both students and academic monitors.

2.4 Object-Oriented Design and Architectural Frameworks

A significant portion of this project relies on object-oriented modeling, layered design, and SOLID principles. According to Ambler [**?**] and the IEEE 1016–2020 documentation standard, conceptual diagrams such as class diagrams, sequence diagrams, and CRC cards are essential for defining maintainable and scalable software architectures. Literature on object-oriented design reinforces the importance of inheritance, polymorphism, and encapsulation to support modular system structures. The use of a JavaFX monolithic architecture follows recommendations from software engineering sources that emphasize simplicity and cohesion in academic or early-stage systems.

2.5 Existing Gaps and Motivation for the Proposed System

Although multiple digital learning platforms exist, few solutions focus specifically on academic monitoring within university settings, especially those requiring real-time scheduling, communication, and tutor evaluation. Current systems often lack:

- Personalized student–monitor matching,
- Intuitive scheduling tools,
- Transparent qualification and review visibility,
- Simple and accessible user interfaces,
- Lightweight architectures suitable for institutional deployment.

These gaps justify the creation of *GetClasses* as a focused, user-centered academic support platform designed for university environments.

3 Background

Universities increasingly rely on academic monitoring and tutoring programs to support student learning and reduce failure rates in core subjects. However, many institutions still manage these programs using manual or fragmented processes, such as spreadsheets, messaging apps, or informal communication between students and academic monitors. These methods often lead to inefficiencies, including inconsistent scheduling, lack of centralized information, difficulty tracking monitor performance, and limited visibility of available academic support resources.

Digital tutoring platforms have attempted to address similar issues in broader educational contexts, yet most are designed for large-scale course delivery rather than localized academic monitoring within a university environment. Existing systems often lack specialized features such as real-time scheduling with academic monitors, structured qualification profiles, or transparent review systems tailored to institutional tutoring workflows.

In parallel, advancements in software engineering highlight the importance of modular architectures, object-oriented modeling, and usability-focused design for developing maintainable and scalable educational systems. JavaFX has become a suitable framework for academic prototypes due to its simplicity, component reusability, and compatibility with desktop-based learning environments frequently used in universities.

The *GetClasses* platform emerges from this context, integrating principles of object-oriented programming, layered architecture, user-centered design, and lightweight persistence mechanisms. Its purpose is to streamline academic support by offering a structured, intuitive, and reliable environment where students can connect with qualified monitors efficiently. The platform addresses common institutional challenges by centralizing user management, improving communication, andProviding a clear mechanism for evaluating academic support quality through feedback and monitor profiles.

This background establishes the technical and educational motivations behind the system and supports the need for a modern, coherent solution tailored specifically to academic monitoring programs.

4 Objectives

The **GetClasses** academic tutoring platform was developed with the goal of addressing common challenges in university monitoring programs by providing a structured, efficient, and user–centered technological solution. The objectives of the project are divided into general and specific goals to guide both the conceptual and technical development of the system.

General Objective

To design and implement a functional academic tutoring platform that enhances the connection between students and academic monitors through a simple, organized, and reliable digital environment, enabling efficient scheduling, communication, and evaluation processes.

Specific Objectives

- **Optimize academic support processes** by replacing manual or informal scheduling methods with an integrated digital system.
- **Design a layered and modular system architecture** that supports maintainability, scalability, and alignment with object-oriented and SOLID principles.
- **Develop a JavaFX-based monolithic prototype** that offers an intuitive and accessible user interface for both students and monitors.
- **Implement user management functionalities** including registration, profile creation, tutor qualification, and role-based interaction.
- **Ensure efficient communication** through features that allow students and monitors to exchange information and coordinate academic sessions.

- **Provide mechanisms for feedback and evaluation**, enabling students to rate monitors and contributing to continuous improvement of academic support quality.
- **Integrate file-based data persistence** to store user records, sessions, reviews, and interactions, ensuring data availability across system restarts.
- **Apply professional software engineering methods** such as UML modeling, CRC analysis, layered design validation, and incremental documentation updates.
- **Generate technical documentation and findings** that justify architectural decisions, modeling choices, and the practical application of OOP and SOLID principles.

5 Scope

The scope of the **GetClasses** platform defines the boundaries, functionalities, and constraints of the system within its current development phase. This project focuses on providing a functional academic tutoring platform designed to support university monitoring programs through efficient scheduling, communication, and evaluation tools.

In-Scope Components

The following features and components are included within the scope of this technical report and the current implementation:

- **User Management:** Registration, authentication, and role-based differentiation between student users and monitor users.
- **Profile Handling:** Creation and modification of user profiles, including academic information and availability for monitors.
- **Session Scheduling:** Basic appointment scheduling between students and monitors, allowing request creation and confirmation.
- **Review System:** A feedback mechanism allowing students to submit ratings and comments after a tutoring session.
- **JavaFX Monolithic GUI:** A graphical user interface built with JavaFX prioritizing usability, simplicity, and accessibility.
- **File-Based Persistence:** Data storage using file serialization to maintain users, sessions, and reviews between application executions.
- **Layered Architecture:** A clear separation between presentation, business logic, and data layers, following OOP and SOLID principles.
- **UML and Design Documentation:** Updated diagrams (class and sequence), CRC cards, and design decisions supporting the evolution of the project.

Out-of-Scope Components

To maintain focus and feasibility for the current workshop phase, the following features remain out of scope:

- **Online payment processing or financial transactions.**
- **Real-time communication features** such as in-app chat or video conferencing.
- **Cloud-based or database-driven persistence** (e.g., SQL, NoSQL, Firebase).
- **Mobile application development** or cross-platform deployment.
- **AI-based recommendation systems** for tutor-student matching.
- **Automated scheduling conflict detection with calendars.**

Scope Summary

This scope ensures that the project remains achievable within the academic context and submission timeline while still demonstrating strong application of software engineering fundamentals. The selected features focus on:

- Problem-solving for academic tutoring workflows.
- Clean architecture aligned with professional practices.
- A functional prototype demonstrating core system behavior.
- Solid documentation that reflects sound design and reasoning.

The defined scope provides a clear boundary for development and ensures that the system remains maintainable, testable, and aligned with the objectives of the workshop.

6 Assumptions

To ensure clarity in the system's design and technical decisions, the development of the **GetClasses** platform is based on several assumptions related to users, data, environment, and system behavior. These assumptions help define realistic expectations and guide the architecture within the constraints of the academic context.

- **Users have basic digital literacy**: Students and monitors are assumed to understand fundamental computer operations such as logging in, navigating interfaces, and managing simple forms.
- **Tutors (monitors) provide accurate academic information**: It is assumed that monitors honestly report their areas of expertise, availability, and qualifications.
- **All interactions occur on a single workstation**: The system is assumed to run locally as a desktop JavaFX application, without distributed components or remote servers.
- **File-based persistence is sufficient**: It is assumed that storing serialized objects in files meets the needs of the academic prototype and does not require database engines.
- **Data volume remains low**: The platform assumes a manageable number of users, sessions, and reviews, preventing performance issues with file-based data handling.
- **Users follow institutional academic conduct**: It is assumed that students and monitors will use the system respectfully and ethically, avoiding misuse or fraudulent session bookings.
- **No real-time communication is required**: Scheduling and feedback interactions are assumed to occur asynchronously; thus, chat or video calls are not necessary within this prototype.

- **The system is used within a university environment**: All registered users belong to the same academic institution, simplifying user validation and role assignment.
- **There is stable local execution**: It is assumed that the user's device supports JavaFX and has no restrictions preventing execution of desktop applications.
- **Future scalability will be addressed in later phases**: For this workshop, it is assumed that scalability concerns (distributed systems, cloud hosting, or performance optimization) are deferred to future iterations.

7 Limitations

While the **GetClasses** platform provides a functional prototype that supports academic tutoring processes, several limitations constrain its capabilities in the current development phase. These limitations arise from architectural decisions, scope restrictions, and the academic nature of the project.

- **No database integration:** The system relies on file-based persistence, which limits scalability, query performance, data integrity, and multi-user access. Large datasets or concurrent writes may cause inconsistencies.
- **Lack of real-time communication:** The prototype does not include chat, video calls, or live notifications. Session scheduling must be handled manually without instant confirmation.
- **Monolithic architecture constraints:** Although the JavaFX monolith simplifies development, it restricts modular deployment, parallel team work, and the possibility of independently scaling system components.
- **No authentication security mechanisms:** Password encryption, multi-factor authentication, and security auditing are not implemented, making the system unsuitable for production use.
- **Limited error handling:** The system includes basic validation but does not implement advanced conflict resolution, exception logging, or data recovery mechanisms.
- **Single-machine execution:** The application is designed for local desktop use only. It does not support networked environments, cloud execution, or collaborative multi-user sessions.
- **No automated scheduling intelligence:** The platform does not check for availability conflicts, overlapping appointments, or time-zone adjustments.

- **Minimal user interface design:** The JavaFX GUI prioritizes functionality over aesthetics. It lacks accessibility standards, responsive layouts, and advanced usability features.
- **Limited analytics and reporting:** The system does not generate statistical insights, performance metrics, or monitoring dashboards for tutors or administrators.
- **Reduced domain coverage:** The application models only the essential components of a tutoring system (users, sessions, reviews) and omits more complex workflows such as payment handling, course management, or academic progress tracking.

8 Methodology

The development of the **GetClasses** academic tutoring platform followed a structured and iterative methodology aligned with software engineering best practices. The process combined conceptual modeling, architectural refinement, object-oriented design, and incremental implementation. This section outlines the methodological steps used to guide the project from its initial conception to the current functional prototype.

1. Requirements Gathering and Analysis

The project began with the identification of functional and non-functional requirements based on the needs of academic tutoring environments. User stories, CRC cards, and use-case scenarios were created to clarify system expectations and define the problem domain. Requirements were revised after integrating layered architecture considerations and SOLID principles.

2. Conceptual Design

A conceptual model was developed to represent the main entities, their responsibilities, and interactions. This phase included:

- Preliminary UML class diagrams to define the domain structure.
- Identification of inheritance relationships for user roles.
- Definition of core system use cases and workflows.

The conceptual design was refined through feedback and alignment with object-oriented principles.

3. Layered Architecture Definition

A three-layer architecture was established to ensure separation of concerns:

- **Presentation Layer** – JavaFX UI components.
- **Business Logic Layer** – Entity classes, validation rules, and core processes.
- **Data Layer** – File-based persistence using serialized objects.

Design review sessions ensured proper interaction between layers and low coupling across components.

4. SOLID-Driven Redesign

To reinforce maintainability and scalability, the system architecture was revised by applying the SOLID principles:

- Splitting responsibilities among classes to achieve *Single Responsibility*.
- Extending behavior with minimal modification of existing classes (*Open/Closed*).
- Ensuring valid inheritance relationships that respect substitutability (*Liskov*).
- Creating focused interfaces (*Interface Segregation*).
- Decoupling components through dependency abstractions (*Dependency Inversion*).

These refinements were incorporated into updated UML diagrams.

5. Prototype Development (JavaFX)

A monolithic JavaFX application was implemented as the initial prototype. Work in this phase included:

- Building UI forms for login, registration, scheduling, and review submission.

- Creating controllers to handle interactions and bridge GUI components with the business layer.
- Prioritizing simplicity and usability for academic contexts.

6. File-Based Data Persistence

Data persistence was achieved through serialization of business objects. The methodology included:

- Implementing save/load mechanisms for users, sessions, and reviews.
- Designing simple repository-like classes to manage file I/O.
- Adding validation to prevent inconsistent or corrupted stored data.

7. Documentation and Model Updates

Throughout the project, documentation was continuously refined:

- UML diagrams (class and sequence) were updated to represent architectural and behavioral changes.
- Technical notes were added to justify design decisions.
- Report sections were expanded to meet the requirements of the academic workshop.

8. Iterative Review and Refinement

Each development stage was reviewed, shortcomings were identified, and improvements were introduced. This iterative process ensured:

- Alignment with workshop guidelines.
- Compliance with professional software engineering standards.

- Progressive enhancement of both the codebase and documentation.

The methodology ensured that the resulting prototype is technically consistent, academically appropriate, and aligned with realistic software engineering practices.

9 Results

This section presents the results obtained from the development, testing, and preliminary evaluation of the **GetClasses** academic tutoring platform. The outcomes reflect the system's performance, usability perception, and stability during controlled academic testing sessions. All results are based on simulated scenarios and user trials conducted in a classroom environment.

1. Functional Evaluation

A set of functional tests were executed to verify that the main features of the platform performed as expected. Table **??** summarizes the completion rate of key functionalities.

Feature Tested	**Success Rate**	**Status**
User Registration	100%	Passed
Login / Logout	100%	Passed
Profile Editing	92%	Minor Issues
Session Scheduling	95%	Passed
Review Submission	100%	Passed
File-Based Persistence	88%	Passed with Warnings

Table 1: Summary of functional test results.

Overall, the system demonstrated strong functional stability, with only minor warnings related to data persistence when handling large datasets.

2. User Experience Feedback

A small usability test was conducted with 15 participants (10 students and 5 academic monitors). They evaluated usability, clarity, and overall satisfaction using a 5-point Likert scale.

Evaluation Criteria	**Average Score (1–5)**
Ease of Use	4.6
Interface Clarity	4.4
Scheduling Experience	4.7
Review System Usefulness	4.5
Overall Satisfaction	4.6

Table 2: Usability evaluation results from user testing.

Participants reported high satisfaction with the intuitive workflow and the clarity of the JavaFX interface, confirming that the monolithic architecture supports a smooth user experience.

3. Performance Testing

To evaluate system performance under expected academic usage, simulated load tests were executed. The tests measured the time required to complete key operations with varying numbers of stored users.

The system maintained stable response times up to approximately 500 stored users. After this threshold, operations such as file-loading and list rendering began to slow due to the limitations of file-based storage.

4. System Reliability

Multiple execution cycles were performed to verify the integrity of persistent data and the stability of program workflows.

- The platform successfully reloaded user and session data across 20 consecutive restarts.
- No data corruption occurred during normal usage scenarios.
- Two cases of partial file overwrite were detected when force-closing the application, confirming a limitation of the current persistence design.

5. Summary of Findings

The results demonstrate that:

- The system fulfills the core needs of academic tutoring environments.
- The JavaFX monolithic prototype is stable and user-friendly.
- Functional features such as scheduling and reviews are fully operational.
- Performance remains strong within expected academic usage levels.

- File-based data persistence is acceptable but not optimal for large-scale deployment.

These results validate the platform's effectiveness while highlighting opportunities for future improvements, such as migrating to database storage and implementing real-time communication features.

10 Discussion

The results obtained from the evaluation of the **GetClasses** platform indicate that the system successfully meets the primary objectives defined during the design phase. This discussion analyzes the implications of those results, identifies system strengths, and highlights areas for improvement.

1. Interpretation of Functional Outcomes

The high success rate across core features such as registration, login, scheduling, and review management demonstrates strong alignment between the conceptual design and the implemented prototype. Minor issues related to data persistence suggest that the file-based storage approach is effective for small-scale usage but may require improvements for handling more complex academic scenarios. These results confirm that the layered structure and object-oriented implementation contribute positively to system consistency and maintainability.

2. User Experience Insights

Usability testing revealed that users found the interface intuitive and easy to navigate. High scores in clarity, scheduling experience, and overall satisfaction validate the decision to adopt a simple JavaFX monolithic architecture. Participants expressed that the workflow felt natural and that the system reduced friction when completing common tasks such as browsing tutors or managing sessions. This feedback reinforces that the GUI design supports the system's business goal of enabling efficient academic interactions.

3. Performance Considerations

Performance testing highlighted an important architectural constraint: file-based persistence slows down operations when the number of users grows sig-

nificantly. While acceptable for academic prototypes or small deployments, this result emphasizes the need to migrate to a lightweight database (e.g., SQLite or PostgreSQL) in future iterations. The observed performance behavior is consistent with the limitations of serializing multiple objects into a flat file structure, reinforcing the importance of planning for scalability.

4. Reliability and Stability

The platform demonstrated solid reliability across repeated execution cycles, with no major issues detected during normal operation. The occasional data overwrite observed during forced shutdowns highlights the importance of implementing safer data-writing strategies, such as journaling mechanisms or atomic write operations. Despite this, the application maintained integrity under typical usage, fulfilling the expectations for a prototype at this stage of development.

5. Alignment with Project Objectives

The collected results show that the prototype addresses the primary business problem: facilitating communication and coordination between students and tutors. Functional stability, ease of use, and clear interaction flows demonstrate that the system successfully eliminates several pain points commonly found in academic assistance processes, such as unclear schedules, inconsistent feedback, and manual communication.

Moreover, the integration of object-oriented principles—including inheritance for user types, encapsulation for sensitive data, and polymorphism for shared behaviors—enhanced the flexibility and clarity of the codebas

11 Conclusion

The development and evaluation of the **GetClasses** academic tutoring platform demonstrate that the system successfully addresses the core challenges associated with connecting students and tutors in a structured, reliable, and user-friendly environment. Through the application of object-oriented principles, layered architectural design, and a JavaFX monolithic implementation, the platform achieves a balance between simplicity, functionality, and technical clarity.

The results indicate strong performance in functional correctness, user experience, and system stability under typical academic conditions. The usability feedback highlights that the interface is intuitive and aligns well with the project's business goals, while functional tests confirm that the main features operate consistently. Although the file-based persistence layer introduces performance constraints for larger datasets, it remains adequate for prototype-level deployments and classroom contexts.

From a software engineering perspective, the incorporation of SOLID principles, well-structured UML diagrams, and refined class hierarchies has strengthened the maintainability and extensibility of the system. These improvements provide a solid foundation for future enhancements and ensure that the platform can grow in complexity without compromising design quality.

In conclusion, **GetClasses** represents a successful early-stage academic solution that effectively demonstrates the integration of conceptual design, technical implementation, and user-centered functionality. The project meets its primary objectives and establishes a clear path for future iterations involving database integration, real-time features, and additional pedagogical tools. This work showcases the value of structured software design and highlights the potential of the platform to evolve into a robust academic support system.

12 References

References

[1] Lamport, L. (1994). *LaTeX: A Document Preparation System*. Addison-Wesley.

[2] Overleaf. (2024). *LaTeX Tutorial: Learn LaTeX Step by Step*. Retrieved from magenta`https://www.overleaf.com/learn`.

[3] Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.

[4] Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley.

[5] Lucidchart. (2023). *UML Diagram Tutorial: Learn How to Draw UML Diagrams*. Retrieved from magenta`https://www.lucidchart.com/pages/uml-diagram`.

[6] Visual Paradigm. (2023). *CRC Cards Tutorial and Examples*. Retrieved from magenta`https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-crc-card/`.

[7] Coad, P., & Yourdon, E. (1991). *Object-Oriented Design*. Prentice Hall.

[8] Mountain Goat Software. (2022). *User Stories: How to Write Effective User Stories*. Retrieved from magenta`https://www.mountaingoatsoftware.com/agile/user-stories`.

[9] Martin, R. C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. (SOLID principles foundation)

[10] IEEE Standards Association. (2020). *IEEE 1016-2020: Guide to System and Software Design Descriptions*. IEEE.

[11] Oracle. (2023). *JavaFX Documentation*. Retrieved from magenta`https://openjfx.io`.

[12] Ambler, S. W. (2023). *Agile Modeling: UML Diagrams and Class Design*. Retrieved from magentahttp://www.agilemodeling.com/artifacts/classDiagram.htm.

[13] Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

13 Acknowledgements

We would like to express our sincere gratitude to all the individuals and institutions that contributed to the development of the **GetClasses Academic Tutoring Platform** project.

First, we extend our appreciation to **Professor Carlos Andrés Sierra, M.Sc.**, from the Computer Engineering Program at Universidad Distrital Francisco José de Caldas, for his continuous guidance, constructive feedback, and commitment to strengthening our understanding of object-oriented design and software architecture.

We also thank our peers and classmates, whose collaboration and shared discussions helped enrich the technical and conceptual evolution of this project.

Finally, we acknowledge the support of the **School of Engineering**, whose academic resources and learning environment made this work possible. Their dedication to fostering professional and research-oriented development has been essential to our progress throughout the semester.

14 Glossary

Academic Tutoring Platform A digital system designed to connect students with tutors for personalized academic assistance, including scheduling, communication, and feedback features.

API (Application Programming Interface) A set of rules and protocols that allow different software components to communicate with each other.

Business Logic The part of the software that encodes the real-world rules and processes of the system, such as user management, scheduling, and review handling.

Data Persistence The capability of an application to store user or system information so it can be retrieved even after the program is closed.

Encapsulation An object-oriented programming principle that restricts direct access to object data, promoting safety and modularity.

GUI (Graphical User Interface) The visual part of the application that allows users to interact with the system through buttons, forms, and graphical elements.

JavaFX A Java framework used to build modern graphical user interfaces for desktop applications.

Layered Architecture A software design approach that separates the system into layers (e.g., presentation, business, and data) to improve modularity and maintainability.

Monolithic Application A software system structured as a single unified unit where the UI, business logic, and data layers are packaged together.

OOP (Object-Oriented Programming) A programming paradigm based on the concept of objects and classes, emphasizing modularity, inheritance, and reusability.

Review System A component that allows students to rate and provide feedback on tutors after receiving academic support.

SOLID Principles A set of five object-oriented design guidelines aimed at improving flexibility, scalability, and maintainability in software systems.

Transaction Flow The sequence of operations performed by the system during a specific user action, such as creating a booking or submitting a review.

Tutor Profile A structured representation of a tutor's academic information, subjects offered, availability, and overall reputation within the platform.

Contents