TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Valentin Djomin 221543IAPM

Creating a Web Environment for the Self-Practicing of Tasks for Creating UML Class and Package Diagrams

Master's thesis

Supervisor: Erki Eessaar
PhD

Co-supervisor: Priit Järv
PhD

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Valentin Djomin 221543IAPM

UML klassi- ja paketiskeemide loomise ülesannete iseseisva lahendamise veebikeskkonna loomine

Magistritöö

Supervisor: Erki Eessaar
PhD

Co-supervisor: Priit Järv
PhD

Tallinn 2025

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Valentin Djomin

Date: 12.05.2025

Abstract

This thesis focuses on the development of a web-based application aimed at improving the process of teaching and learning Unified Modeling Language (UML) diagrams in higher education. UML is a widely adopted standard for software modeling and plays an important role in both education and software projects. However, existing UML tools are often too complex for beginners, which makes learning more difficult or lack essential features for effective learning and automated assessment. The goal of this thesis is to create a web-based software, UML Solver, that allows students to practice building UML class and package diagrams based on predefined tasks, while receiving immediate feedback through an automated validation system. The tasks are not built-in, i.e., the lecturer can specify an unlimited number of tasks.

The application is developed using PHP 8 and JavaScript. The database of the tool is made by using PostgreSQL. Diagrams are stored as JSON objects in the database (in columns with JSONB type). It is designed to be user-friendly and extensible, with the possibility to support additional UML diagram types in the future. The system includes a built-in validation mechanism that automatically checks student-created diagrams against the reference diagram created by the lecturer and provides meaningful feedback to support learning.

Key features of the software include an interactive diagram editor, configurable validation settings, and environment that make both self-practice and formal assessment possible. The development process follows a research-driven and iterative approach, integrating pedagogical needs and technical constraints.

The result is a flexible and accessible educational tool that addresses current limitations in UML learning environments. It improves student engagement and understanding of UML syntax, semantics as well as modeled subjects and makes it easier for lecturers to assess the skills and knowledge of students.

This thesis is written in English and is 87 pages long, including 7 chapters, 30 figures and 5 tables.

Annotatsioon

UML klassi- ja paketiskeemide loomise ülesannete iseseisva lahendamise veebikeskkonna loomine

Lõputöö eesmärk on luua veebipõhine rakendus UML-diagrammide õpetamiseks ja õppimiseks kõrgkoolis. UML on populaarne ja standardiseeritud visuaalne modelleerimiskeel, mis mängib olulist rolli nii hariduses kui ka tarkvaraarenduste projektidess. Olemasolevad modelleerimisvahendid on sageli algajatele liiga keerulised, mis pärsib õppimist või puuduvad neil vajalikud funktsioonid õppimise ja hindamise toetamiseks.

Töös loodud tarkvara - UML Lahendaja - võimaldab üliõpilastel vastavalt etteantud ülesannetele koostada klassi- ja paketidiagramme ning saada tänu sisseehitatud automaatsele kontrollile kohest tagasisidet. Tagasiside andmiseks võrreldaks üliõpilase loodud diagrammi õppejõu koostatud diagrammiga, mis on andmebaasis ülesande juures salvestatud. Võrdlemisel arvestatakse diagrammi struktuuriga (millised on elemendid, milliste teiste elementidega ja mis viisil on need seotud), kuid mitte diagrammi kujundusega. Võrdlusmehhanism on süsteemi sisse ehitatud, sõltub diagrammi tüübist ning uute diagrammi elementide või diagrammi tüüpide lisamisel vajab täiendamist. Samas ülesanded ei ole süsteemi sisse ehitatud, mis tähendab, et õppejõud saab lisada piiramatu arvu ülesandeid. Selle toetamiseks on loodud õppejõule ka tarkvara võrdlusdiagrammide loomiseks ja JSON kujul esitatud diagrammide vaatamiseks. Rakendus on loodud PHP 8 ja JavaScripti abil. Andmebaasi loomiseks kasutatakse PostgreSQLi. Nii üliõpilaste loodud diagrammid kui õppejõu koostatud võrdlusdiagrammid salvestatakse andmebaasis JSON tüüpi objektidena (veergudes, mis on JSONB tüüpi).

Platvormi peamised omadused on interaktiivne redaktor ja võimaluste loomine selle vahendi kasutamiseks nii iseseisvaks harjutamiseks kui ka formaalseks hindamiseks. Arendusprotsess järgib teaduspõhist ja iteratiivset lähenemist. Rakenduse loomisel peeti silmas, et seda oleks võimalik tulevikus laiendada teiste UMLi diagrammitüüpide toega.

Lõpptulemuseks on kasutajasõbralik ja ligipääsetav tarkvara, mis aitab tudengitel õppida paremini mõistma UML-i süntaksi, semantikat ning selle abil koostatud mudeleid ja lihtsustab õppejõududel hindamist.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 87 leheküljel, 7 peatükki, 30 joonist, 5 tabelit.

List of Abbreviations and Terms

AJAX	*Asynchronous JavaScript and XML.* A web development technique that allows sending and receiving data from a server asynchronously, enabling page content updates without full reloads.
API	*Application Programming Interface.* A defined set of functions and protocols that allow different software components or systems to communicate and exchange data.
BPMN	*Business Process Model and Notation.* A standardized graphical notation for modeling and documenting business workflows and processes in a clear and understandable way.
CASE tool	*Computer Aided System Engineering* tool. Modeling software where one can create models, validate models, and generate new artifacts (different types of models, documentation, code, tests) based on these.
CRUD	*Create, Read, Update, Delete.* The four fundamental operations used in databases and web applications to manage persistent data.
CSRF Token	*Cross-Site Request Forgery Token.* A unique, secret value generated by the server to protect against unauthorized requests by verifying that actions come from an authenticated user.
CSS	*Cascading Style Sheets.* A stylesheet language used to describe the look, formatting, and layout of HTML elements on web pages.
EA	*Enterprise Architect.* A professional tool for designing, visualizing, and documenting software architecture using UML and other modeling languages.
HTTP	*HyperText Transfer Protocol.* The foundation of data communication on the web, used to transfer HTML pages and other resources between clients and servers.
HTTPS	*HyperText Transfer Protocol Secure.* An extension of HTTP that uses encryption (typically via SSL/TLS) to ensure secure communication between web clients and servers.
JS	*JavaScript.* A dynamic programming language that runs in browsers and is used to create interactive and dynamic content on web pages.
JSON	*JavaScript Object Notation.* A lightweight, human-readable format for structuring and exchanging data, commonly used in web APIs.

LLM	*Large Language Model.* A type of machine learning model trained on massive text datasets to understand, generate, and interact using human language.
LTS	*Long-Term Support.* A software release type that receives security updates and maintenance for an extended, predefined period.
MDA	*Model-Driven Architecture.* A development methodology that focuses on creating system functionality through models that can be transformed into executable code.
MVC	*Model-View-Controller.* A software architectural pattern that separates application logic into three interconnected components: model, view, controller.
NPS	*Net Promoter Score.* A metric used to measure customer satisfaction and loyalty.
OO software	*Object-Oriented* software. A type of software designed using the principles of Object-Oriented Programming, where the system is built around objects that represent real-world entities, encapsulating data and behavior.
OOP	*Object-Oriented Programming.* A programming paradigm based on the concept of "objects". It emphasizes principles like encapsulation, inheritance, and polymorphism to create modular, reusable, and maintainable software.
PHP	*Hypertext Preprocessor.* A widely-used open-source scripting language designed for server-side web development and dynamic page generation.
PNG	*Portable Network Graphics.* A raster image format that supports lossless compression and transparency, often used for web graphics.
REST	*Representational State Transfer.* An architectural style for designing stateless web services that use standard HTTP methods for operations.
RSA	*Rational Software Architect.* A professional tool developed by IBM for software modeling and architectural design.
RUP	*Rational Unified Process.* A structured software development process framework that emphasizes iterative development and well-defined roles and deliverables.
SSL	*Secure Sockets Layer.* A cryptographic protocol that was originally developed to secure data transmitted over the internet.
SVG	*Scalable Vector Graphics.* An XML-based vector image format that can scale to any size without losing quality, ideal for web graphics and diagrams.

TLS	*Transport Layer Security.* A modern, more secure replacement for SSL used to encrypt and protect internet communication.
UML	*Unified Modeling Language.* A standardized modeling language used to specify, visualize, and document the design of software systems.
URL	*Uniform Resource Locator.* The standardized address used to locate and access resources on the internet.
XMI	*XML Metadata Interchange.* A standard format for exchanging metadata information via XML. It is commonly used to represent UML models in a structured, machine-readable way.
XML	*eXtensible Markup Language.* A markup language used to encode documents and data in a structured, machine- and human-readable format.

Table of Contents

List of Figures

List of Tables

1 Introduction

This chapter presents the background of the work, states the problem and research objectives, and lays out the structure of the thesis.

1.1 Background

The Unified Modeling Language (UML) was developed in the mid-1990s as a response to the growing complexity of software systems and the need for a standardized way to represent software designs. Before UML, many different modeling languages already existed (and to be fair, still exist), which caused confusion and inconsistency in software documentation and communication. To solve this, the Object Management Group (OMG) introduced UML as a unified standard in 1997. Since then, UML has become the most widely used modeling language in software engineering.

UML is a general purpose visual language that can be used to model many different domains, including domains that have nothing to do with IT. However, UML diagrams are mostly used to provide a way to visualize the structure and behavior of information and software systems. They allow developers, architects, and stakeholders to understand requirements, how the system is designed, how its components interact, and how data flows throughout the system. The visual nature of UML makes it easier to communicate ideas, detect potential design flaws early, and ensure consistency between requirements and implementation. Over the years, UML has become an important part of the software development lifecycle, especially in large-scale and complex projects where documentation and planning are critical. UML is used in a variety of IT domains including software engineering, systems engineering, business process modeling, and database design [1]. According to recent industry surveys, around 88% of software development specialists use some form of UML in their workflows [2]. It is especially common in enterprise environments, embedded systems development, and model-driven engineering. Popular software development methodologies like Rational Unified Process (RUP) and Model-Driven Architecture (MDA) are based on UML diagrams [3]. However, in modern software development, even teams that follow agile

methods often use simple UML diagrams to help plan and discuss the system architecture.

UML 2.4 specifies 14 different types of diagrams [4], each serving a different purpose. These diagram types fall into two categories - diagram types to specify the static structure of the system and diagram types to specify the behaviour of the system. For instance, class diagrams describe the static structure of a system, use case diagrams show functional requirements from a user's perspective, and sequence diagrams illustrate object interactions over time. The ability to create, interpret, and validate these diagrams is considered an important skill for software engineers. Understanding UML is also frequently a requirement in academic programs, job descriptions, and technical interviews.

In the context of higher education, teaching UML has become a common part of computer science and software engineering curricula. However, many students struggle with mastering diagram syntax and semantics without practical tools that offer guidance and feedback. As a result, there is a growing need for interactive platforms that help students practice creating UML diagrams and receive immediate validation of their work. Such systems not only improve learning outcomes but also support lecturers by automating the assessment process.

1.2 Problem Statement

In the field of information technology and software engineering, educational platforms have evolved significantly, providing students with automated tools to enhance their learning experience. This thesis argues that teaching and assessing the creation of visual models for software and information systems is an area where automation can be effectively applied. Models are simplifications of the world that help us to better understand the modeled domain. Models capture knowledge about the domain and models could be used for the efficient communication about the subject matter. Models could be a basis for manual code creation, or at the age of Large Language Models (LLMs) could be used as the direct input for the LLMs to generate implementations (code, tests).

One of the most popular modeling languages is UML (Unified Modeling Language). UML is a general purpose visual modeling language that allows us to create different types of diagrams. Being general purpose means that this language could be used in many domains and for many tasks. The most often used diagram type is the class diagram that is used for specifying the static structure of the system [2]. This type of diagram can, for instance, be used for creating concept maps, models of website structure, domain models, which are the input for finding software classes, entity-relationship diagrams, database design models, and software design models. The survey [2] also showed that in various contexts where UML diagrams are used to model software architecture, class diagrams are frequently chosen by practitioners as the primary tool for describing functional structures, data structures, concurrency structures, and even software code structures. Similarly, package diagrams are heavily favored for modeling software module structures. This importance highlights how widely used and valued class diagrams and package diagrams are among software professionals for showing and organizing complex software designs. Both class diagrams and package diagrams are static structure diagrams [5].

Many UML tools are designed for professionals and can be too complex for students. Educational tools like StudentUML are easier to use but still have several limitations [6]. StudentUML does not have a web version, which makes it less accessible. It lacks features for lecturers to give feedback directly on diagrams, and it doesn't allow viewing of student past attempts, which could help in learning from previous mistakes.

Additionally, the user can export diagrams to different image formats, but it cannot export diagrams in XMI format, which is a standard format used for transferring models between different modeling tools [6]. Moreover, StudentUML does not support exporting models in PlantUML textual format, which is much more compact, simpler, and better human readable compared to XMI. It further limits its use with other tools and platforms for diagram sharing.

This thesis aims to develop a web-based application that enables students to practice creating UML class and package diagrams, incorporating an automated validation system to assess their correctness. We will call the software UML Solver. The application will be built entirely from scratch using PHP 8 and JavaScript, incorporating open-source libraries for diagram creation and visualization. Although the first

implementation of the tool only supports class diagrams and package diagrams, it will be designed in a manner that the support for other UML diagram types could later be added.

The research focuses on developing and evaluating a web-based system that automates UML class and package diagram assessment. It builds upon existing methodologies related to UML notation, comparison algorithms, feature-matching techniques, and the impact of web-based educational tools on learning effectiveness. UML class and package diagrams will serve as the primary focus due to their fundamental role in the analysis of software systems and design of object-oriented software systems.

1.3 Research Objectives

The primary objective of this research is to develop and evaluate a web-based application that allows for the interactive creation and automated validation of UML class and package diagrams, which are crucial in software system analysis and design. The research aims to explore the effectiveness of automated systems in validating these diagrams and to define optimal approaches for comparing diagrams created by students with reference diagrams provided by lecturers. The tool will implement an algorithmic validation approach of submitted diagrams. This will involve the use of feature-matching algorithm to assess the structural and syntactical correctness of the diagrams.

The ultimate goal is to provide immediate and meaningful feedback to students, improving their understanding and skills in UML diagramming while also addressing the gap in current educational tools regarding validation capabilities.

Depending on the configurations made by the lecturer, the tool could be used for self-practicing as well as for grading and collecting points. With the help of this tool a lecturer should be able to familiarize students with UML syntax and semantics and possibly introduce to students some common modeling (in this case analysis and data modeling) patterns.

1.4 Structure of the Thesis

The thesis is structured into several chapters, each focusing on different aspects of the project. It begins with an Introduction that includes the background, statement of the problem, research objectives, and the overall structure of the thesis. This sets the stage for understanding the need for the research and what it aims to achieve.

In the Methodology chapter, the research object and process are described as well as used tools are shortly mentioned.

The Related Works chapter reviews academic literature and previous research focused on the teaching of UML, software modeling tools, and validation techniques in educational environments. It highlights existing approaches to modeling education, discusses challenges faced in diagram-based learning, and outlines prior attempts to automate the assessment of UML diagrams. Subsections explore methods for teaching and assessing UML knowledge, as well as the criteria for evaluating model quality, which are directly relevant to the validation logic implemented in this thesis.

The Existing Tools chapter provides a comprehensive overview of current technologies relevant to UML modeling, comparison, and practice. It begins with a short discussion of UML diagram types, followed by an in-depth review of widely used UML diagramming tools such as Enterprise Architect, StarUML, UMLet, yEd Live, Creately, and Lucidchart. The section also explores UML comparison tools like EMF Compare, Visual Paradigm, and UMLDiff, evaluating their capabilities for analyzing model differences. Finally, it examines code practicing platforms such as LeetCode and HackerRank, highlighting how they support interactive learning. This analysis forms a basis for identifying functional gaps and areas for innovation that the proposed UML Solver system aims to address.

The System Development chapter outlines the technical foundation and implementation process of the UML Solver. It begins with an overview of system requirements, distinguishing between functional and non-functional needs such as performance, maintainability, and browser compatibility. The chapter then presents the chosen technology stack and tools that support development. The architecture of the application is analyzed through the separation of server-side and client-side components. Following that, the user interface is discussed in terms of both student and lecturer perspectives,

each tailored to their respective roles. Finally, the chapter explores the diagram validation subsystem, covering both rule-based algorithmic checks and LLM-based validation. Although LLM-based validation was explored, the system outlines clear reasons why this method is currently not suitable for automated assessment in this context.

The Analysis and Results chapter presents an evaluation of the UML Solver system based on real user feedback and development reflections. It begins with an analysis of student survey responses, demonstrating the system's usability, clarity of instructions, and effectiveness in improving UML skills. The discussion summarizes how well the tool meets its intended goals and addresses the specific needs of learners. Identified limitations are acknowledged, along with their impact on the overall system experience. Reflections on the development process highlight strengths, difficulties, and insights gained throughout the project. The chapter concludes with suggestions for future improvements and directions for ongoing development.

Finally, the Summary captures the main findings and results of the thesis, providing a concise wrap-up of the research and development process. This allows readers to quickly grasp the contributions of the thesis and its impact on the field of UML diagramming platforms.

2 Methodology

The research follows the Design Science methodology, which emphasizes the design and creation of an artificial artifact and validates its effectiveness through proper evaluation [7]. The approach ensures that the developed artifact (in this case a novel software system) is both necessary and practically useful, aligning with best practices in the field and addressing real-world requirements.

2.1 The Object

During academic journey, the author had the opportunity to closely work with UML diagrams studying the subject IDU1550 (Software Architecture and Design). This course provided a solid foundation in understanding key concepts such as software reliability, interoperability, and the principles of architecture and design. The course highlighted the connection between design, code, and testing, introduced agile documentation practices, and emphasized the role of UML diagrams in clearly expressing architectural ideas [8]. The learning process included both theoretical understanding and practical exercises, where students were expected to create various UML diagrams and explain their use in different phases of software development. The author used a StarUML to create all required UML diagrams, as he was working on a Linux-based system and faced technical difficulties installing Enterprise Architect through Wine. StarUML has support for Linux and provides the necessary functionality. It was fully sufficient to meet the course requirements. All diagrams were built based on the assignment description by the lecturer. The evaluation process took place during practical sessions, where each student presented and defended their diagrams in an oral format, explaining the structure, purpose, and correctness of their solutions.

Personal experience of the author with UML diagrams extends beyond academic studies. In his professional work, he has frequently used various types of UML diagrams to model the architecture of web-based systems. One of the most valuable applications was the use of package diagrams to represent the structure of Gradle modules in complex enterprise Java applications. Since his projects often involved

multiple microservices, each with its own internal architecture, it became essential to document these structures accurately. These diagrams served as a powerful tool for knowledge sharing within development teams and were particularly useful for onboarding new team members. Visualizing the relationships between packages helped team members gain a faster and deeper understanding of how different parts of the system were organized and connected. In addition to package diagrams, the author regularly used class diagrams to support the design and development of systems. These diagrams allowed developers to better understand the structure of objects, their attributes, methods, and the relationships between them. This made it easier to implement solutions aligned with the design principles discussed during planning. Moreover, the author had practical experience creating sequence diagrams to illustrate the flow of logic across services in a distributed microservice environment. These diagrams helped clarify the order of operations and interactions, especially when tracing complex business logic that spanned several components of the system.

Based on this background and practical need, the vision of UML Solver is focused on helping university students learn and practice creating UML diagrams. The first release, which is planned to be the result of this work, covers class diagrams and package diagrams. The platform is aimed at providing an educational environment where students can improve their understanding of UML by drawing diagrams and receiving automated feedback based on predefined correct solutions provided by the lecturer. This tool will allow students to explore the logical structure of systems through visual modeling, aiding both their conceptual knowledge and technical skills. Its web-based nature means students do not need to install additional software and can access the application from any location with an internet connection. Moreover, it means that the system is available 24/7, thus allowing students to select the most suitable time for them for practicing.

The software does not aim to replace professional modeling environments because it has different goals. The first version of the software does not aim to support the entire scope of UML. The software is specifically tailored for educational purposes, designed to address the most common types of diagrams used in introductory and intermediate-level software engineering courses. The result of the software is not only the ability to create diagrams but also to validate them against lecturer-defined models,

giving students immediate insights into mistakes and encouraging self-improvement. The software is intended to be used (at least) in the database courses. There are many resources that present analysis- and data modeling patterns that encapsulate the best practices in modeling and designing a database for a particular domain [9]-[14]. Moreover, there are many object-oriented design patterns that present suitable structures of software classes [15]. If a task for a student is to create a diagram that corresponds to a pattern, then it not only introduces and familiarizes UML but also introduces to the learners these patterns.

From a validation perspective, the tool will control the structural correctness of diagrams by comparing the student's created version with the lecturer's reference. At this stage of development, the system includes restrictions in the user interface that prevent common errors from being made, such as connecting incompatible elements or omitting required fields. Whether it is a flaw (some mistakes from which one can learn cannot happen) or a feature (this kind of behavior is also in better modeling tools) is debatable. When an error is detected, the user is notified through a clear and informative message, which contributes to the learning process by guiding students toward the correct approach. However, the current version of the software does not validate pragmatic aspects, such as design style, color, font, or layout aesthetics. These features, which fall under the domain of visual presentation rather than logical correctness, are excluded from the current scope of validation. The focus remains on helping students understand and apply the structural rules of UML diagrams, ensuring they can express architectural and design ideas effectively using standardized notation.

The object of this thesis is the development of an educational software UML Solver that supports the creation and validation of UML diagrams for university students. It draws from academic experience, real-world practices, and a focused educational goal. The platform will guide students in constructing correct diagrams, provide meaningful feedback, and serve as a practical supplement to traditional learning methods in software architecture and design education.

The expected outcomes of the research include a fully functional web-based system for students, enabling UML class and package diagram construction and automated validation. In addition, the result includes a web-based user interface for lecturers for constructing reference diagrams and viewing diagrams submitted by students.

Although existing web-based UML modeling tools offer diagramming functionality, few provide automated validation, particularly with AI integration (e.g., Lucid.app). This research primarily implements an algorithmic validation model but also explores LLM capabilities. In practice, while LLMs can provide quite accurate responses, it yields inconsistent results when the same question is posed multiple times, and the answers may not always be correct. The rule based algorithmic approach makes the system a valuable tool for software engineering education. The developed system has the potential to be extended to other types of UML diagrams and other visual modeling languages, serving as a foundation for future research in automated assessment technologies for various educational fields. The software offers syntax and semantics validation of models, with the possibility of expanding to pragmatic evaluation in the future using LLM.

The software intends to repeat the success of SQL Solver that was developed in Tallinn University of Technology in 2024 and has been successfully employed in database courses [16]. The tool is able to automatically assess solutions to SQL tasks (i.e., SQL statements) submitted by students. The tool allows an unlimited number of tasks and over the last year more than 400 tasks have been added to it. UML Solver uses the ideas of user interface and database structure of SQL Solver as the basis for its own user interface and database structure, allowing the author to concentrate on the functionality that is specific to this tool. The software will be developed using PostgreSQL, PHP8, and JavaScript/jQuery in accordance with non-functional requirements, and will include functionality for supervisors to define reference UML diagrams.

2.2 The Development Process

The development process for the UML diagram practicing platform was methodically organized and closely matched the guidelines provided by the thesis supervisors. Firstly, preliminary research was conducted on existing UML diagramming tools, automated validation techniques, and coding-practicing tools.

Next, a prototype of the application user interface was created using Figma [17]. This prototype served as a visual draft for discussions about the system's functionalities and design, ensuring that both the student and supervisors had a clear understanding of the

project's direction from the beginning. Selected views from the prototype are presented in Appendix 2, including the UML diagram types overview page in Figure 1 and the class diagram task solving interface in Figure 2.

During these discussions, the supervisors proposed an initial structure for the database that was intended to support the application's functionalities efficiently. As the project progressed, the database schema was changed based on further analysis and feedback to ensure it could handle data more effectively. The final revised version of the database structure is composed of several interconnected schemas, which are presented in Appendix 3. These include the task collections schema Figure 3, feedback schema Figure 4, language classificators schema Figure 5, problem of task attempts schema Figure 6, and the task schema Figure 7, all designed in DataGrip.

To manage the development process effectively, a Kanban board on KanbanFlow was used [18]. This tool proved invaluable for tracking progress and organizing tasks related to the project. All feature requests and suggestions for improvements from the supervisors were logged into the Kanban board. This approach ensured transparency in task management, allowed for clear prioritization, and helped maintain a steady development pace. It also made it easier to monitor the overall project status and identify potential bottlenecks. An example of the Kanban board setup is shown in Appendix 4 Figure 8.

This approach not only helped in maintaining a clear overview of the project status at any given time but also facilitated the prioritization and scheduling of tasks. Weekly calls in the MS Teams were an integral part of the workflow, during which the work completed over the week was reviewed. These sessions provided an opportunity to demonstrate new functionalities, discuss the status of ongoing tasks, and receive direct feedback from the supervisors. During these meetings, we also updated the Kanban board with new tasks and planned the goals for the next week.

This structured weekly review and planning cycle ensured that the project remained on track and aligned with the supervisors expectations. The iterative nature of this process, combined with the agile management approach provided by KanbanFlow, allowed for flexibility in development and responsiveness to changes in project requirements.

Regular updates and the clear visualization of progress on the Kanban board helped maintain a lively and effective development environment, significantly enhancing the final product's quality. This organized yet adaptable approach to project management made sure the development of UML Solver was well-managed and aligned with the academic supervisors educational and functional specifications.

The system was tested with students to measure its usability [19]. User feedback was collected through web-based surveys to evaluate usability and system performance. The accuracy of the system in automatically validating diagrams was evaluated by solving a set of tasks and comparing the results produced by the system with the expected results. The evaluation was done by the author and by the supervisors. To do that a set of tasks was created that cover all the model element types that the software currently supports. The evaluators provided correct answers to the tasks to make sure that these are accepted by the system and also provided incorrect answers (incomplete, the use of wrong types of model elements, the creation of wrong model elements) to make sure that these are rejected by the system.

Based on received feedback from these evaluations some changes were made to the software.

2.3 Tools and Technologies Used

Throughout the development of the platform, a set of supporting tools and technologies was used to organize the workflow in writing, testing and debugging the code, and to ensure the quality and maintainability of the system. These tools were essential in facilitating the development process, improving productivity, and supporting collaboration with supervisors. In this section, each tool is described along with its specific role and contribution to the project:

- IntelliJ IDEA is used as an integrated development environment (IDE). It is chosen for local development due to its robust features that support both front-end and back-end development. IntelliJ IDEA provides comprehensive tools for code writing, editing, and debugging, enhancing developer productivity and making it easier to manage large codebases.

- Figma: Used for designing the user interface. Figma is a web-based tool that enables designers to create interactive and visually appealing UI prototypes. In the initial stages of development, a prototype was implemented in Figma, which was instrumental in defining the system requirements. Figma's collaborative features make it easy for designers and developers to work together and iterate on the UI design, ensuring that the final product is both functional and user-friendly. This early prototyping in Figma clarified what needed to be implemented, making the development objectives more understandable and straightforward.
- Git + GitLab: Git is employed as the version control system to manage the application's codebase, while GitLab serves as the online platform for repository hosting. This setup is particularly useful for a thesis project as it makes it convenient to track changes and update the codebase efficiently, facilitating smooth progress and collaboration. The final archived source code will be sent to the supervisors.
- Enterprise Architect: This tool is used to conceptualize and visualize how UML diagrams are implemented within the software. Enterprise Architect supports the design and modeling of software and information systems architecture, making it invaluable for planning and documenting the application structure. Additionally, using this tool has enabled the identification of necessary elements and functionalities required for constructing UML class diagrams as well as package diagrams. It also plays an important role in creating models for the application's architecture and implementation.
- Postman: Postman is utilized to test the web API's endpoints, ensuring that they behave as expected and handle various types of requests correctly. It allows developers to quickly verify the functionality of the RESTful services and debug them when necessary, improving the quality and reliability of the application.
- Ubuntu 22.04.5 LTS: The application is developed and tested on Ubuntu, a widely-used Linux distribution known for its stability and support. Using Ubuntu as the development platform ensures that the application is stable and performs well in a Linux server environment, which is commonly used in production. The final version of the application will be deployed in a similar environment, ensuring compatibility and optimal performance in real-world settings.

- KanbanFlow is a lightweight task management tool based on the Kanban methodology. It allows users to visualize tasks, manage priorities, and monitor progress in a structured and intuitive way. In the context of this research, KanbanFlow was used to track the development process, organize tasks, and record ideas for future improvements. It helped maintain a clear overview of the project status and supported time management throughout the development cycle.
- DataGrip is a professional database management tool developed by JetBrains. It provides an intuitive interface for working with SQL databases, enabling users to browse data, run queries, and visualize relationships between tables. DataGrip was used to directly manage the PostgreSQL database to make CRUD operations against database schema. It also served as a visual reference for understanding the relationships between tables and constraints.
- Microsoft Teams is a collaboration platform designed for communication and teamwork, offering video meetings, chat, and file sharing in one environment. Within this research, Microsoft Teams was used for weekly meetings with supervisors, which took place every Monday. It also hosted a dedicated group chat where supervisors provided feedback, answered questions, and shared relevant academic materials, supporting continuous communication and supervision.
- Google Docs is a cloud-based word processing application that allows multiple users to edit documents in real time and leave comments for collaborative writing and reviewing. In the context of the thesis project, Google Docs was the primary tool for writing the thesis text. Supervisors were able to follow updates, suggest edits, and provide structured feedback directly in the document. This ensured consistency, correctness, and progress in the written part of the work.
- Google Forms was used to conduct a student survey after the application launch in order to gather feedback on the usability and effectiveness of the system. This allowed for structured data collection and analysis of user experience to inform evaluation and further development.
- ChatGPT is a LLM that can assist with writing, coding, debugging, and language correction tasks across various domains. ChatGPT was used to support programming in JavaScript and PHP, which are not author’s primary

programming languages. It helped resolve technical issues more efficiently and served as a grammar and clarity checker for the English text of the thesis.

This section has provided an overview of the tools that supported the development process and helped ensure effective implementation and collaboration. Detailed information about the technology stack of the system is presented in the section The Technology Stack.

3 Related Works

This chapter refers to scientific research that is related to the topic of the current thesis. The ideas from these papers were used as an input for designing the web-based UML practicing tool.

3.1 Teaching and Assessing UML Knowledge

The integration of automated assessment systems into educational environments has gained considerable attention in recent years, particularly in the domain of object-oriented software engineering. UML remains a core element in the instruction of systems analysis and design courses, serving as a universal visual modeling language used to represent the structure and behavior of software systems. Due to the increase in student numbers and the demand for personalized feedback, several solutions have been proposed to automate the evaluation of UML diagrams produced by students, thereby improving efficiency and consistency in the grading process.

One of the recent directions explored in research focuses on systems that compare student diagrams to a predefined lecturer's solution. The study presented in "Assessing Students' UML Class Diagrams: A New Automated Solution" [20] describes an approach that relies on transforming both tutor and student diagrams into XMI format, parsing them into structured elements, and matching them using a developed algorithm. This system enables scoring based on identified mismatches and generates detailed feedback, which not only assists tutors but also supports self-assessment by students. The solution is scalable and well-suited for remote and large-group learning environments, emphasizing automated feedback and time efficiency in grading. However, this solution requires using an open-source modeling environment to construct the diagram, which is then converted to XMI for validation.

A complementary approach is presented in "A Tool to Automate Student UML Diagram Evaluation" [21]. This research describes a Java-based standalone tool capable of evaluating several types of UML diagrams - such as class, use case, activity, sequence,

and state machine diagrams - by comparing the XMI representations of student and reference models. The tool performs both strict and relaxed comparisons, and outputs scores along with detailed feedback files for students and lecturers. The study highlights the tool's effectiveness in reducing the time required for grading and increasing consistency, especially in courses with large enrollment numbers.

In the context of educational software tailored for UML learning, the tool described in "StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design" [22] addresses the challenges of using professional-grade CASE tools in academic environments. The study outlines how general-purpose UML tools, while powerful, are often too complex for beginners due to their feature-rich interfaces and strict adherence to UML syntax. StudentUML was developed specifically to meet educational needs, focusing on simplicity, ease of use, and consistency checking. It allows students to construct diagrams incrementally, ensures logical correctness, and helps them internalize modeling concepts. The tool supports project-level management and distinguishes between analysis and design diagrams, which is essential for teaching proper software development methodology.

In addition, the study concludes that many tools available on the market, including MinimUML [23], UMLet, and Ideogramic UML, lack comprehensive support for model validation, project consistency, and feedback generation. While these tools aim for simplicity, they often fail to guide students in creating semantically correct models or in understanding the distinctions between different types of UML diagrams. This highlights the need for tools like StudentUML, which combine ease of use with educational rigor.

The practical application and evaluation of StudentUML in a university-level course is further discussed in "Learning and Practicing Systems Analysis and Design with StudentUML" [24]. This research presents a real-world case of integrating the tool into lab sessions of a Systems Analysis and Design course. According to the study, StudentUML enhanced students' understanding of UML concepts, reduced confusion caused by complex interfaces, and provided immediate, valuable feedback. The tool's validation capabilities and project consistency checks were found to be particularly useful for beginners, reinforcing correct modeling practices.

Another solution targeted at educational environments is described in “QuickUML: A Tool to Support Iterative Design and Code Development” [25]. This tool supports drawing UML class diagrams, generating Java code, and reverse engineering diagrams from existing source code. While limited in scope compared to other systems, QuickUML is praised for its minimalistic interface and ability to help students understand object-oriented design through iterative modeling and coding cycles. The research emphasizes that introducing such tools early in education helps promote design thinking and aligns well with the principles of incremental software development.

Another relevant study describes the DIAGRAM environment by Py, Auxepaules, and Alonso [26]. This tool aims to teach students the basics of object-oriented modeling using UML Class Diagrams. DIAGRAM uses a diagnostic module. This module compares the student's diagram to a reference diagram provided by the teacher. The comparison is done using a graph-matching algorithm. The tool identifies different types of errors, such as missing or extra elements. It then provides feedback at different levels (e.g., notifying, questioning, suggesting) to help the student. Like UML Solver, DIAGRAM provides automated feedback by comparing diagrams. However, there are several differences between the two tools. DIAGRAM seems to be a standalone desktop application (built with Java). In contrast, UML Solver is a web-based tool that does not need installation. Also, DIAGRAM only supports Class diagrams. UML Solver supports both Class and Package diagrams and is designed to be extendable to other diagram types in the future. The comparison methods are different: DIAGRAM uses graph matching, while UML Solver uses a rule-based comparison of JSON data. The specific ways feedback is given also differ. DIAGRAM's description does not explicitly mention features like persistent storage of attempts, detailed student statistics, or export formats like PlantUML, which are part of UML Solver.

Web service UMLGrader also gives automated feedback on UML class diagrams by comparing them to a standard solution. It checks for common errors and was initially for diagrams from IBM Rational Rose [27]. UMLGrader differs from UML Solver because it only handles class diagrams and is tied to a specific tool. UML Solver is web-based, uses its own JSON format for diagrams made in the tool, supports both class

and package diagrams, and focuses on structural correctness in its rule-based comparison, not just name matching. UML Solver also tracks attempts and statistics.

The MinimUML tool [23] offers a simple way to learn UML diagramming for beginners. It provides basic UML features for class diagrams in a simple desktop app, focusing on design exploration. While minimUML is easy to use, it does not automatically check student work against a correct example, which is a key feature of UML Solver. MinimUML helps with creating diagrams, while UML Solver also helps by validating them.

The examined research collectively demonstrates that while professional UML tools offer full-featured environments, their complexity can hinder learning. Educational tools that provide real-time validation, manageable interfaces, and targeted feedback are more effective for academic purposes. Automated assessment systems must go beyond simple structural comparison. They must facilitate learning by allowing students to understand and correct their mistakes. Feedback, both formative and summative, plays a crucial role in this process.

Existing tools for UML education either simplify diagram creation (like minimUML), offer general feedback (like DIAGRAM), or provide assessment for specific tool formats (like UMLGrader). However, there is a need for a tool like UML Solver. UML Solver is a web-based platform that doesn't require installation. It supports key UML diagrams (class and package) with an interactive editor and, most importantly, automatically checks student diagrams against teacher-defined examples using rules. This checking uses a standard JSON format. Features like saving attempts, showing progress, and exporting diagrams make UML Solver a complete learning tool. It aims to solve common problems in UML education by being easy to use while providing strong, immediate feedback.

In conclusion, the reviewed works provide a foundation for the development of the proposed platform, which aims to combine the strengths of automatic evaluation and pedagogical effectiveness. Students will construct their own solutions, which will be automatically evaluated based on structural and semantic alignment with the reference model. The system will generate immediate, detailed feedback, thus contributing to a

more interactive, scalable, and effective learning environment for software design education. The system developed in this thesis differs from existing solutions by being fully web-based and integrating both an interactive diagram area and schema validation logic. Additionally, as all data is stored in the cloud, lecturers can easily create and update the database with new information.

3.2 Model Quality

Good model quality means that the model is built correctly, shows the right meaning, and can be used easily in real situations. When analyzing models created by drawing UML diagrams, it is important to consider three main aspects: syntax, semantics, and pragmatics [28]. These aspects define how well a model is constructed, how accurately it represents the intended meaning, and how useful it is in practice. Understanding these three elements is essential for evaluating the quality of models in both software development and educational settings. They provide a clear framework for checking whether a model is correct, meaningful, and suitable for its purpose.

Syntactic correctness is the only syntactic goal. A syntactically correct model expresses statements only with symbols that are defined in the language and provides all the required constructs and information to follow the grammar of the language. Thus, each UML diagram must adhere to certain rules about what types of elements can be used, what elements can be connected and what association types could be used to connect what types of elements. For instance, in the context of relationships like association, aggregation, and composition, it is possible to connect a class to itself to show different roles or responsibilities within the same entity. However, in the case of relationships like implementation and generalization, such self-connections are not allowed. These syntactic rules ensure that the diagrams are structurally consistent and understandable. Syntactic rules in case of visual models also prescribe how the elements should look like on diagrams.

Semantics is about the meaning of the elements in the model. There are two semantic goals - validity and completeness. Validity means that all the statements made by the model should be correct and relevant. Completeness means that the model should contain all the statements about the domain that are correct and relevant. Statements

should not have any collisions in the logic and should be certain. What model is complete depends on the task for which the model is used.

Pragmatics, the third aspect, means that the model should be comprehensible, i.e., all interested parties should be able to understand it. Thus, models should be presented in a manner that makes them as easily understandable as possible. In case of diagrams it means, for instance, that diagrams should avoid long lines, crossing lines, font size should be appropriate, coloring of elements is consistent and avoids too bright colors. Moreover modeling conventions should be followed, e.g., in the context of generalizations in a class diagram a more general concept (class) is placed above a more specific concept (class).

4 Existing Tools

The tool that is produced as the result of the thesis combines the aspects of UML diagramming tools, coding-practicing environments that give automated feedback, and tools that allow us to find the difference between textual artifacts. It is used to practice creating UML diagrams and checking these against the reference provided by the lecturer. Thus, the overview of existing software should cover all these aspects. Investigating the tools allowed the author to collect ideas for making UML Solver. We provide a compact comparison of the tools in the form of comparison tables as well as present a short section about the main features of each tool. Moreover, because the current thesis is about making an educational software for learning UML, we firstly provide a short overview of relevant aspects of UML.

4.1 UML

The Unified Modeling Language (UML) is an industry-standard modeling language widely adopted in both academic and professional settings for designing and visualizing software systems. Maintained by the Object Management Group (OMG), UML is comprehensive and structured, with its latest specification (version 2.5.1 as of April 2025) spanning over 790 pages [29]. This extensive scope reflects UML's flexibility in modeling complex systems across different domains, encompassing both static and dynamic aspects of software architecture. Due to its standardized nature and visual clarity, UML is especially useful in educational contexts, where it aids students in understanding the architecture, components, and interactions within object-oriented systems.

To meet the functional and educational goals of the software under development, it was decided to implement support for two key UML diagram types: **class diagrams** and **package diagrams**. These diagrams are fundamental in the structural modeling of software systems. Class diagrams describe the blueprint of object-oriented components, while package diagrams provide an overview of the modular organization of a system. Class diagrams can be used to model requirements to a database (i.e., entity-relationship

diagrams), database data structures (e.g., structure of tables), and concepts/relationships in the real world that would be the basis for determining what the classes should be in an Object-Oriented software (OO software).

By modeling class and package diagrams, it becomes possible to represent both the internal details and the architectural organization of a system. Class diagrams focus on internal class structure and interactions, while package diagrams abstract these into higher-level organizational units. These two diagram types complement each other and form a complete picture of the system's static structure. Their inclusion ensures that learners can explore both micro and macro perspectives of information system- or software architecture, facilitating better comprehension, clearer analysis, and improved system analysis and design capabilities.

Through this structured approach based on the UML 2.5.1 specification, the model delivers an educationally effective and technically sound representation of software systems, providing an essential foundation for teaching principles of object-oriented modeling and system architecture.

4.1.1 Class Diagrams

Class diagrams are arguably the most widely used form of UML diagrams, particularly in the context of object-oriented development. They provide a static structural representation of a system, showcasing its classes, attributes, operations, and the relationships between different entities. In object-oriented programming (OOP) , each class acts as a template for creating objects. On a UML diagram **a class** is typically visually composed of three sections: its name, a list of attributes, and a set of operations. Attributes denote the properties or fields of a class, such as variables holding specific data, while operations define the behaviors or methods that objects of the class can perform. Each attribute and operation is associated with a visibility modifier:

- "+" for public,
- "-" for private,
- "#" for protected.

Each visibility modifier controls accessibility and encapsulation, reinforcing principles of object-oriented design.

Beyond regular classes, UML allows the definition of abstract classes. **An abstract class** cannot be instantiated directly; instead, it serves as a base from which other classes inherit. The main characteristic of an abstract class is the presence of abstract operations, e.g., method declarations without implementations. These operations must be concretely implemented in the derived subclasses. Abstract classes provide a mechanism to define general behaviors while deferring specific implementations, encouraging code reuse and polymorphic behavior in system design.

Another critical structural element in UML class diagrams is **the interface**. An interface represents a contract that other classes must fulfill. It defines a set of operations without implementations, ensuring that any class implementing the interface provides concrete definitions for all declared methods. Interfaces are essential for achieving modularity and decoupling in software systems, allowing for more flexible and interchangeable components.

In addition to structural elements, class diagrams also capture a variety of relationships that describe the nature of interaction between classes. These include association, aggregation, composition, implementation, and generalization.

Association denotes a basic link between two classes, indicating that objects of one class are connected and may interact with objects of another class. This connection can be bidirectional or unidirectional. Associations can also be self-referential, meaning a class can be linked to itself, which is useful in modeling hierarchical relationships like employee-supervisor structures. Multiplicity can be defined at each end of an association to specify how many instances of a class may be involved in the relationship.

Aggregation represents a "whole-part" relationship where the part can exist independently of the whole. For example, a university may consist of multiple departments, but those departments can exist independently as organizational units. Aggregation is visually represented with an empty diamond at the aggregate (whole) end of the association.

Composition is a stronger form of aggregation, where the lifecycle of the part is strictly tied to the whole. If the whole is destroyed, its parts are also destroyed. This is typically used when modeling scenarios such as a building and its rooms, where rooms cannot

exist independently without the building. A solid diamond at the composite (whole) end is used to represent composition.

Implementation relationship describes how a class implements an interface. It indicates that the class agrees to fulfill the contract defined by the interface by providing concrete implementations of its methods. This relationship is represented using a dashed line with a hollow arrow pointing to the interface.

Generalization reflects inheritance between a subclass and a superclass. It implies that the subclass inherits all properties and operations of the superclass and may also override them or introduce new ones. This relationship is visualized with a solid line and a hollow triangle pointing toward the superclass.

4.1.2 Package Diagrams

In contrast to class diagrams, package diagrams provide a high-level view of the system by organizing model elements into packages. This modular representation is especially useful when dealing with large-scale systems, as it improves clarity and maintainability. **Packages** in UML are used to group related classes, interfaces, and even other packages under a common namespace. Each package is represented as a folder-like symbol, typically labeled with its name. The primary role of a package is to encapsulate functionality and promote organized code structures, helping developers and students understand the boundaries and dependencies of various components.

Package diagrams also include relationships that define how different packages interact. The most commonly used relationship is **the usage** dependency. This indicates that one package relies on another to function correctly, without implying ownership or inclusion. A usage relationship is often used to model situations where one component makes use of types or operations defined in another module. It allows the modeling of dependencies across different parts of the system and supports modular development practices.

4.2 UML Diagramming Software

The landscape of UML diagramming tools varies significantly, covering both offline and online platforms that provide a variety of functions and access models. Many of

these tools provide both free and paid versions, where the free versions might come with a trial period or have significant functional limitations. For our system, having an online capability is essential, allowing students to work not only from the university but also from home and other remote locations. This section will shortly analyze some tools, focusing particularly on features relevant to creating UML class diagrams and UML package diagrams, noting any unique characteristics of each tool.

Some modeling tools (i.e., tools that are not only meant for drawing diagrams) have functionality to check models against predefined rules. On the other hand one can see the absence of built-in functionality for diagram validation against a reference model. This is not a limitation of these tools because this is not a part of intended functionality of the tools.

However, the lack of the functionality points to a significant gap between the capabilities of available UML tools and the specific needs of educational institutions. This necessitates custom development or adapting open-source tools, creating a significant opportunity for educational technology innovation in platforms with integrated, advanced validation capabilities.

Many existing UML tools are closed-source, which makes it difficult to add features like automatic validation or comparison with a reference model. While some tools, such as Enterprise Architect or StarUML, offer strong modeling capabilities, they do not support direct model comparison. Simpler online tools like UMLet or yEd Live are easier to use but also lack built-in validation. As a result, there is a clear gap between the needs of educational institutions and the functionality of current tools.

4.2.1 Enterprise Architect

Enterprise Architect (EA) is a robust offline modeling tool that supports a comprehensive array of modeling languages including UML, BPMN, and ArchiMate [30]. Primarily designed for enterprise architects and software developers, EA is utilized for detailed software design, business process modeling, and system integration. It includes both a paid version and a 30-day free trial, which provides full functionality temporarily. The paid version is essential for ongoing large-scale projects due to its advanced features like database modeling, code generation, and reverse engineering capabilities. The software is intended for complex projects where detailed

documentation and strict adherence to standards are crucial. EA provides advanced validation functionalities, though automated validation against a user-defined standard requires custom scripts or specialized setups or use a third party extension like Model Expert [31].

There are few studies about the popularity of UML modeling tools. We have found two studies according to which EA is the most popular UML modeling tool [2], [32]. This is the reason why EA is used as a basis for designing the user interface of UML Solver.

There are many third party extensions to EA [33]-[34]. Thus, hypothetically it would be possible to create an extension that compares a model against a reference. However, such a tool will require installing additional software (by a potentially inexperienced user), EA is Windows software (i.e., making it difficult to use by Linux and Mac users), and such tool cannot be used to assess students (because students will have to have a reference model).

4.2.2 StarUML

StarUML is an agile offline UML tool aimed at software engineers and educational environments focusing on quick and efficient modeling [35]. It supports essential UML diagrams and offers extensions for additional functionality. StarUML provides a free version with basic features and a paid version that includes more advanced functionalities, such as model validation, diagram themes, and export options (e.g., PDF and HTML). This tool is designed for simplicity and speed, making it ideal for students and professionals who need to produce UML diagrams quickly without the overhead of more comprehensive tools. While it includes model validation features, it does not natively support comparison of models to find the difference without further customization (external plugins).

4.2.3 UMLet

UMLet is an open-source, web-based UML tool designed for fast diagram creation [36]-[37]. It is free to use, with no paid version, focusing instead on providing a straightforward platform for users to quickly draw UML diagrams. UMLet is particularly used in educational settings where students need to learn the basics of UML efficiently. Its design philosophy is about simplicity and speed, allowing for rapid

diagramming with minimal learning curve. UMLet is ideal for educational workshops or classes that require a tool for illustrating UML concepts without the complexity of more feature-rich environments. The main disadvantage, it does not support automated validation.

4.2.4 yEd Live

yEd Live is a versatile graph and diagramming online tool that offers both free and paid versions [38]. The free version allows basic diagramming suitable for personal or educational use, while the paid version provides enhanced features like high-quality exports, advanced layout algorithms, and more extensive customization options. Originally designed to cater to both casual and professional users, yEd Live supports a broad range of diagram types beyond UML, including network diagrams, flowcharts, and more. This tool is used for both educational purposes and corporate settings where visual representation of data and systems is required. yEd Live does not include specific features for UML diagram validation or comparing models with each other to find the differences.

4.2.5 Creately

Creately is a user-friendly diagramming and design online tool that facilitates collaboration and simplicity in creating diagrams [39]. It offers a limited free version primarily for trial purposes and individual users, while the paid versions cater to teams and enterprises with features like real-time collaboration, extensive shape libraries, and full access to its desktop version. Creately was designed to enhance teamwork on visual content, making it suitable for educational groups, business teams, and remote collaborations. It is widely used in schools, universities, and businesses where collaborative creation of diagrams is essential. Creately does not specifically offer functionality for comparing models but does provide tools for consistency checking within diagrams.

4.2.6 Lucidchart

Lucidchart, known for its clean interface and extensive diagramming capabilities, offers both a basic free version and advanced paid subscriptions [40]. The free version is suitable for personal use with some limitations on features and the number of documents, whereas the paid versions offer advanced features like team collaboration,

revision history, and integration with other tools (e.g., Google Drive, Slack). Lucid.app is designed to support a wide range of users from students to professionals across various industries, facilitating complex diagramming needs including network diagrams, process maps, org charts, and UML diagrams. It is particularly valued for its ability to support collaborative work environments and integrate seamlessly with various online platforms. Each tool is tailored to meet specific needs, ranging from simple educational purposes to complex enterprise requirements. The choice between free and paid versions typically depends on the user's need for advanced features, support, and scalability. These tools are not only used for educational purposes but also in professional settings where detailed modeling, documentation, and collaboration are important. Lucid.app integrates AI features that simplify collaboration and automate the diagram creation process. These AI tools offer suggestions for diagram improvements, assist in data analysis, and automatically generate visualizations based on textual descriptions. Lucid.app does not natively support automated comparison of two models but is highly adaptable for collaborative and detailed diagramming.

4.2.7 Evaluation of UML Diagramming Tools

Before beginning the implementation of the application, it was important to evaluate existing UML diagramming tools. This evaluation served two main purposes. Firstly, it helped to define a concrete set of functional and non-functional requirements for the system. By studying existing tools, the author was able to identify commonly used features, interface solutions, and workflow patterns that users expect in UML diagramming environments. This made it possible to ensure that the developed system would offer a competitive and relevant feature set.

Secondly, the evaluation provided valuable insight into user needs in the domain of diagram creation. Users of UML tools often have different expectations depending on their goals. For example, quick sketching of ideas or formal documentation. By analyzing how current tools address these use cases, the author could better understand what features are most valuable in practical usage. In particular, attention was paid to usability, support for different diagram types, export options, and the balance between flexibility and simplicity. This analysis helped shape the vision of the application and guided design decisions throughout development. A comparison of existing UML diagramming software is presented in Table 1 and Table 2.

4.3 UML Comparison Tools

As software modeling has become a key aspect of object-oriented design education, the need for specialized tools to compare UML diagrams has increased. In modern educational and industrial environments, students and developers often produce multiple versions of UML diagrams for the same problem. This creates a practical need for comparing these diagrams in order to identify structural and semantic differences. Tools designed for UML comparison help automate this process, making it easier to review changes, validate correctness, and assess diagram quality.

UML comparison tools serve various purposes. In education, they support automatic grading and feedback by comparing student diagrams to reference models provided by lecturers. In collaborative development, such tools help teams track changes in system design over time. A typical UML Diff tool analyzes diagrams at different levels: some focus on purely structural elements such as classes, relationships, and attributes, while others also consider naming, and diagram semantics. Advanced tools even evaluate the impact of changes between versions, helping users understand not just what changed, but why the change matters.

Most UML comparison tools operate by converting diagrams into an intermediate format such as XMI (XML Metadata Interchange), which allows structured parsing and element-by-element comparison. The resulting differences can be displayed visually or reported in text form. Visual feedback is especially helpful in educational settings, as it allows learners to directly see where their diagrams deviate from correct solutions. Although the general goal of these tools is the same, their capabilities, integration options, and complexity may vary significantly. Some tools are built into professional CASE environments, while others are stand-alone applications or research prototypes. Below is a short review of the most relevant and widely known UML Diff tools available today.

Every UML comparison tool has certain limitations that affect flexibility or accessibility. Some tools work only inside specific modeling environments like Eclipse or IBM RSA, which makes them hard to use in other systems. Commercial tools, such as Visual Paradigm and IBM RSA, require paid licenses. They are often large, complex, and expensive, which makes them less practical for simple tasks or student use. Free or

academic tools are easier to use but often have limited features. They may not show visual differences or support deep semantic comparison. These limitations show that building a good and flexible UML comparison tool is a complex task.

4.3.1 EMF Compare

EMF Compare is a powerful and widely used tool built on the Eclipse Modeling Framework (EMF) [41]. It supports model comparison for a variety of formats, including UML diagrams defined via Ecore and XMI. This tool offers detailed structural and semantic comparison features. Users can visualize model differences within the Eclipse IDE, making it suitable for software development and academic environments. EMF Compare is especially effective when used in combination with other modeling tools like Papyrus or Sirius. It supports both two-way and three-way comparisons and integrates with version control systems. It is important to note that EMF Compare is not a standalone application. It requires installation and use within the Eclipse IDE environment. This dependency means that users unfamiliar with Eclipse may face a learning curve, and the tool is better suited for technically proficient users or those already working with Eclipse-based modeling workflows.

4.3.2 Visual Paradigm

Visual Paradigm includes an integrated versioning and diagram comparison feature within its Teamwork Server module [42]. It supports visual side-by-side comparison of UML diagrams and highlights differences in model elements, including class structures, relationships, and attributes. This feature is useful in collaborative settings where multiple users work on the same project. It provides clear change tracking and historical comparisons. Visual Paradigm is a professional-grade tool, and its comparison features are intuitive and well-documented. However, all advanced features, including diagram comparison and version diff tools, are only available in the paid versions. The free Community Edition offers limited functionality and does not support Visual Diff for diagram comparison or comparing a diagram in different revisions. Additionally, in the Community Edition, all printed or exported diagrams contain a visible watermark, which is unsuitable for formal academic and professional use.

4.3.3 IBM Rational Software Architect

IBM Rational Software Architect (RSA) provides extensive support for modeling and model management [43]. It includes built-in tools for comparing UML diagrams, detecting structural and semantic changes, and managing multiple model versions. The comparison results are displayed visually, making it easier for users to identify and understand changes between models. RSA is aimed at enterprise environments and integrates well with IBM's ecosystem, including version control systems and automated development pipelines. However, it is a paid commercial product and could be considered heavy for educational use.

4.3.4 Enterprise Architect

Enterprise Architect supports model comparison using its “Baseline” functionality [44]. This feature allows users to capture a snapshot of a UML package at a certain point in time and later compare it with the current version. Differences are shown through a visual interface, including changes in classes, associations, and attributes. Comparison is limited to diagrams created and stored within a single project on the same user’s computer. It does not support comparing diagrams across different projects or between different users. While this method is somewhat limited compared to dedicated diff tools, it is still effective for small-to-medium projects and individual work sessions.

4.3.5 UMLDiff

UMLDiff is a research-based algorithm that implements semantic and structural comparison algorithms for UML diagrams [45]. It has been proposed in academic literature as a way to improve understanding of diagram evolution. UMLDiff analyzes class diagrams by comparing not only syntax but also the roles and relationships of elements within a model. Though not widely used in commercial products, this tool plays an important role in research and educational experiments related to automatic grading and model consistency checking.

4.3.6 DiffMerge

DiffMerge is a general-purpose text comparison tool that can be used to compare XMI files, which represent UML diagrams [46]. However, it does not provide any understanding of UML structures or model semantics. The tool simply highlights textual differences line by line, without interpreting elements like classes, relationships,

or attributes in a meaningful way. It lacks any built-in support for UML or diagram-specific visualization, which makes it unsuitable for users who need visual feedback or structure-aware comparison. Due to these limitations, DiffMerge may only be used in basic or experimental scenarios where a quick textual comparison is sufficient, such as debugging or testing simple student submissions in a prototype environment.

4.3.7 Evaluation of UML Comparison Tools

In evaluating existing UML diagram comparison tools, two key questions were considered.

The first was whether these tools address the specific tasks set in this thesis. It was observed that while some tools provide basic validation capabilities, they do not fully support the goals defined in this work. Most existing software is limited to general-purpose syntax checks and lacks the ability to perform domain-specific validation or semantic analysis. These limitations prevent them from being used effectively in educational environments where specific modeling constraints must be enforced. As a result, they cannot fully satisfy the functional requirements identified in the early stages of this project.

The second question was focused on how the solution developed in the thesis differs from the available alternatives. The solution developed as part of this thesis introduces key advantages that are not present in the reviewed tools. The main distinction lies in the flexibility and extensibility of the validation system. The developed application allows the creation and editing of custom validation rules that take into account the context of the diagram (rule based validation). This includes not only checking for structural correctness but also evaluating the semantic relationships between elements.

The comparative characteristics of UML diagram comparison tools are presented in Table 3.

4.4 Code Practicing Software

Although current platforms do not specifically target UML diagram construction, many coding-practice systems offer valuable insights into effective teaching and skill

evaluation techniques. These platforms engage users through structured exercises, immediate feedback, and supportive guidance - principles that are highly applicable to the design of a UML learning environment. In such systems, the goal is not limited to validating correctness but extends to actively supporting learning through hints, detailed feedback, and progress tracking.

By examining how popular platforms implement features such as task variety, real-time validation, personalized suggestions, and user motivation strategies, we can identify best practices that can enhance both the usability and pedagogical value of a UML-focused tool. The comparison below highlights how each platform addresses these educational aspects and offers a foundation for designing interactive and effective UML training experiences.

4.4.1 LeetCode

LeetCode is a popular platform focused on algorithmic problem-solving [47]. It presents users with coding challenges and offers an immediate evaluation of their solutions. LeetCode provides test cases and identifies edge cases where the user's code fails. The platform highlights the incorrect output and offers the ability to compare it with the expected result. These features promote self-directed learning and help students focus on specific issues in their solutions. For our project, LeetCode is relevant in terms of how feedback is structured - precise, focused, and directly tied to user input. This style of feedback will inform how UML validation messages are presented to students in the platform.

4.4.2 HackerRank

HackerRank offers a broad set of challenges across various domains, including algorithms, databases, artificial intelligence, and more [48]. What makes HackerRank notable is its structured learning paths, which combine theory with interactive tasks. The platform provides a smooth learning curve, combining difficulty scaling with immediate feedback on progress. Its user interface clearly separates task description, code editor, and output validation, creating a clean and focused workspace. The platform also supports hints and explanations that are gradually revealed. These features are valuable for our UML learning platform, where structured progression, clean task presentation, and optional hints can improve the learning experience.

4.4.3 CodeSignal

CodeSignal is another modern platform used by both learners and employers to evaluate programming skills [49]. Its strength lies in creating real-world-like coding environments and producing clear scoring metrics based on performance. CodeSignal tracks accuracy, speed, and efficiency, and provides results in a structured report format. For our case, its relevance lies in how performance feedback is presented in a user-friendly and motivating way. CodeSignal's feature of tracking the time taken to solve a task can be especially useful, as it encourages efficiency and helps monitor student progress over time.

4.4.4 CodeChef

CodeChef is an open platform that hosts programming contests and practice problems for users at all skill levels [50]. One of its most educational features is its discussion section, where users share solutions, ask questions, and learn from one another. It promotes a learning community that is not only supported by automated validation but also by peer interaction. The concept of discussion is particularly interesting, as it provides students with the opportunity to ask questions and receive feedback from the teacher on specific assignments.

4.4.5 Evaluation of Code Practicing Tools

As part of the preliminary work for the development of the UML Solver application, an evaluation of existing code practicing platforms was carried out. The main purpose of this analysis was to understand the user experience offered by such systems. Particular attention was paid to how these platforms handle task validation, deliver feedback to users, and manage timing constraints during exercises. These aspects are critical in learning environments, where immediate and clear interaction with the system plays a key role in user motivation and understanding.

The analysis focused on the way systems present validation results, whether they provide instant feedback or delayed evaluation, and how they notify users about errors or successful submissions. Additionally, the presence of features such as time tracking, task deadlines, and interface responsiveness was examined. These elements were important for shaping the presentation layer of the UML Solver application. By studying existing solutions, it was possible to identify which interaction patterns are

intuitive and effective for educational use cases. The insights gained through this comparison directly influenced the design of user feedback, submission status messages, and interface responsiveness in the developed system.

The comparison of software platforms for code practice and learning is presented in Table 4.

5 System Development

This chapter provides an overview of the development process of the UML Solver platform, a web application designed to help students build UML diagrams and receive automated validation based on reference models created by lecturers. The system aims to support learning and streamline evaluation in academic environments.

The chapter outlines key requirements that shaped the project, describes the technologies used, and explains the general architecture of the application. Special attention is given to how the platform handles diagram validation, ensuring both correctness and feedback for users. The development process followed established software engineering practices to ensure that the system is functional, reliable, and easy to use.

5.1 System Requirements

This section outlines the specific requirements for the UML Solver platform. It covers both functional and non-functional aspects that define how the system should operate and perform.

5.1.1 Functional Requirements

This section presents the functional requirements of the UML Solver system. These requirements define the core features and actions that the platform must support in order to fulfill its educational purpose. They describe how students interact with tasks, create diagrams, receive validation, view statistics, and communicate with lecturers. Additionally, they cover the tools available for lecturers to create and manage reference models. The listed functions ensure the platform operates effectively as a learning and evaluation tool within the academic environment.

Student

1. Can log in by using his/her username and password in Maurus environment.
    - Login function returns student number if he/she can enter and NULL otherwise.

2. Can switch the language between Estonian and English. Switching the language changes not only the user interface language but also the presented content because each task is associated with exactly one natural language.
3. Can see a list of active tasks (name, diagram type, difficulty, state of solving (solved, unsolved, untried))
    - solved - at least one correct attempt (it is irrelevant if there are incorrect attempts after the correct attempt),
    - unsolved - at least one attempt but no correct attempts,
    - untried - no attempts.
4. Can sort (ascending or descending) and filter the list of active tasks based on all the fields.
    - By default the tasks are sorted by name.
5. Can see ordered sets of tasks (name, number of tasks in it, number of solved tasks) that have state active.
    - The sets are sorted by name.
6. Can select a set and see:
    - all the tasks in it (including the sequence number of the task in the ordered set),
    - the description of the set,
    - the number of solved tasks in the set.
7. Can select a task and see:
    - name,
    - diagram type (name + if exists, then diagram type description and documentation; the latter could contain references, i.e, URLs to outside resources),
    - difficulty,
    - state of solving,
    - description.
8. Can see hints (one-by-one or all together) based on the system-level configuration that determines how many attempts one must make to see hints (-1 would mean that hints are never shown).
9. Can see a clock that shows how much time in seconds he/she has already spent at the task page.

10. If a student has selected a set and then selected a task from the set, then at the task page he/she can move to the next or to the previous task in the set.
11. Can solve a task by creating a class diagram or a package diagram.
    - In case of class diagrams at least the following elements should be supported:
        - class,
        - attribute,
            - visibility,
            - type,
            - multiplicity
        - operation,
        - undirected association:
            - name,
            - multiplicities,
            - roles
        - aggregation:
            - name,
            - multiplicities,
            - roles
        - composition:
            - name,
            - multiplicities,
            - roles,
        - generalization,
        - abstract class,
        - interface
    - In case of package diagrams at least the following elements should be supported:
        - package,
        - usage relationship (with possibility to give name).
12. Can submit the answer for automatic assessment.
13. Can see the automatic feedback of the system.
    - Task is solved if there is 100% conformance to the reference model provided by the lecturer with the following exceptions:

        - placement of elements on a diagram is irrelevant,
        - order of attributes/operations in a class is irrelevant,
        - the names are case insensitive, e.g., “Client”, “client”, and “cLiEnT” would be considered equal,
            - On the other hand, “Client” and “Customer” are not considered to be the same thing because the terminology is determined by the task.
        - spaces before and after names are removed before the evaluation,
        - within the class name a single _ and a single space are considered equal.
14. Can give feedback (write a question/remark) to a lecturer regarding a specific task. Feedback is personalized, i.e., not visible to other students.
15. Can see answers to the feedback.
    - From a task page feedback related to this particular task.
    - From a general menu all feedback (time, task name, answering time). Clicking on it opens the actual answer.
16. Can see past task attempts of a particular task (time, state of solving, submitted solution).
17. Can select a past task attempt to continue working with the diagram that was produced as a result of this.
18. Can export the created diagram in PlantUML format.
19. Can save the screenshot of a diagram.
20. Can see the history of all task attempts (time, name, diagram type, difficulty, state of solving).
21. Can sort (ascending or descending) the history based on all the fields.
22. Can see statistics of his/her task attempts:
    - number of task attempts,
    - number of successful task attempts + percentage from total attempts,
    - number of failed task attempts + percentage from total attempts,
    - total spent time,
    - total spent time to get the right answer,
    - for each difficulty level:
        - number of tasks,
        - number of solved tasks,

        - percentage of solved tasks.
    - for each diagram type:
        - number of tasks,
        - number of solved tasks,
        - percentage of solved tasks.
    - for each set of tasks:
        - number of tasks,
        - number of solved tasks,
        - percentage of solved tasks.

23. In case of statistics only the tasks that have state “active” or “visible for statistics” are considered.
24. Can delete history, i.e., make it anonymous. It means that feedback and task attempts of the student are not associated with him/her any more. Whether this is possible depends on the system-level configuration (1 - possible; 0 - not possible).
25. The system should log all task attempts, including time, student-ID, submitted diagram, and given feedback.
26. The number of task attempts that a student can make should be unlimited. However, all the attempts should be logged.
27. Can let the system select a random task.

Lecturer

1. Can log in by using his/her username and password in Maurus environment (Login function returns TRUE if he/she can enter and FALSE otherwise).
2. Can construct a reference model and see/copy its representation in JSON format.
3. Can give a model in JSON format as an input and see its visual representation.
4. Can save a model in PlantUML format.
5. Can save the screenshot of a diagram.

The functional requirements knowingly lack the management of grading. The system should allow students to solve the tasks. It should collect detailed information about task attempts. Grading or receiving points depends on a particular course and keeping it outside the system makes both the system and the grading more flexible. In the system the student knows what he/she has done and the course determines how it translates to points or to a grade.

5.1.2 Non-Functional Requirements

This section defines the non-functional requirements of the UML Solver system. These requirements describe the quality attributes that ensure the platform is secure, accessible, maintainable, and performs efficiently under expected workloads. They include standards for the database, user interface, technology compatibility, language support, and security practices. While they do not define specific functionalities, these constraints are essential for delivering a stable, user-friendly, and scalable system suited for academic use.

Tools

- Database Management System: PostgreSQL.
    - It must be assumed that always the newest version of it will be used.
- Application: PHP 8, CSS, JavaScript, jQuery, Bootstrap.

Database

- All declarative constraints that can be enforced at the database level (PRIMARY KEY, UNIQUE, NOT NULL, FOREIGN KEY, CHECK) should be enforced at the database level regardless of whether a corresponding validation takes place at the application level as well.

User interface

- The general look and feel of the user interface should resemble SQL Solver [16], to deploy ideas that have proven themselves in practice, and leave more time to design the automated assessment module of the system.
- The user interface should also reuse ideas from the Enterprise Architect CASE tool in order to be familiar to students.
- The user interface should follow the requirements of the European Standard for ICT Accessibility EN 301 549 [51].
- It should work with all major web-browsers (Chrome, Firefox, Edge, Opera, Safari).

Multiple language translations

- Translation files of user interface elements should be separate files.
    - It should also be possible to translate the feedback to a task attempt.
- The software should be developed in a manner that it would be easy to add additional languages to Estonian and English.

Security

- The application should use the database as a user with minimal possible amount of privileges, i.e., not as a superuser.
- In the PostgreSQL database PUBLIC should be stripped from all the default privileges.
- SQL injection and cross-site scripting attacks should be prevented.

5.2 The Technology Stack

This section describes the technologies used in building the application, including tools for both front-end and back-end development.

5.2.1 Used Tools

The development of UML Solver utilizes a carefully chosen set of technologies, each playing a critical role in creating an efficient, scalable, and user-friendly system. This section provides an overview of each component of the technology stack:

- PostgreSQL (ver. 17 at the time of creating the thesis) is chosen for its reliability and robust feature set, serving as the database management system for the application. It is a popular open-source software that offers extensive capabilities for handling complex queries and ensuring data integrity [52]. The use of PostgreSQL in this project is essential for managing complex data efficiently and securely. Additionally, it meets a non-functional requirement specified by the supervisor due to PostgreSQL's cost-effectiveness as a free tool, its large and active community, and continuous updates. In addition, PostgreSQL has a widespread use in many other projects at the Tallinn University of Technology. It ensures compatibility and support within the existing technological ecosystem.
- PHP (version 8) is chosen as the primary server-side scripting language for its robust support and notable performance improvements compared to earlier versions of PHP. The preference for PHP was guided by the supervisor's recommendation, emphasizing its ease of maintenance and the potential for the future enhancements. The supervisor's prior experience with PHP and its widespread adoption in other projects at the Tallinn University of Technology

were key factors in this choice. This experience ensures a consistent approach across projects and allows for efficient use of existing knowledge and resources.

- JavaScript with jQuery. JavaScript, supplemented with jQuery, is employed to enhance the client-side functionality of the application. This combination allows for creating a dynamic and responsive user interface by handling events, performing AJAX calls to the server, and manipulating the DOM. jQuery simplifies many common tasks in JavaScript, making the code more manageable and less prone to errors.
- JointJS – the JavaScript library is utilized to provide advanced diagramming capabilities. JointJS allows for the easy creation and manipulation of the graphical elements that make up UML diagrams, offering extensive customization options to fit the specific needs of the application. Compared to other libraries, JointJS was chosen due to its flexibility, open-source availability, active community, and strong support for UML-like structures. Structure could be serialized to JSON format. GoJS was not selected because it is a commercial library, which means limited functionality and visible watermarks without a paid license. bpmn-js is a suitable library for creating diagrams, but It is focused on BPMN format and has limitations in serialization and deserialization (exports to BPMN format or SVG picture). Mermaid was excluded due to its complexity of markdown-inspired text definitions and styling limitations.
- Bootstrap is used to improve the front-end framework of the application. Bootstrap is well-known for its responsive design templates and reusable components, which include buttons, forms, and other user interface elements. This framework makes it easier to develop a visually appealing and consistent interface across different screen sizes and devices. By using Bootstrap, the application ensures a high level of usability and accessibility, making it easier for users to interact with the system. Bootstrap's grid system and responsive design capabilities are especially helpful in creating a layout that adjusts dynamically, providing an optimal viewing experience both on desktop and mobile environments. Using Bootstrap not only helps keep the user interface clean and modern but also speeds up the front-end development process by using its extensive pre-built components.
- Docker is used to containerize the application in the development environment, ensuring that it runs consistently across local development environments.

Docker Compose allows for defining and running multi-container Docker applications. In this project, Docker is particularly useful for managing the application's infrastructure, such as the PostgreSQL database, in a way that is both scalable and isolated from the host system.

This comprehensive technology stack provides a robust foundation for developing and maintaining the UML Solver application. Each technology was selected for its specific benefits, ensuring that the final product is capable of meeting the complex requirements of educational and professional environments where UML diagramming is essential.

5.3 Web Application Architecture

The web application utilizes an architecture designed to provide a reliable and responsive user experience while efficiently handling complex data interactions for UML diagramming. It is structured according to Martin Fowler's three-tier architectural pattern [53], which splits the logic into clear layers for presentation, business logic, and data management. This structured approach enhances the system's maintainability and allows for simpler modifications in the codebase as development needs evolve.

The server-side implementation employs PHP to manage both static page rendering and dynamic API requests effectively. Routing mechanisms distinguish between these functions: requests to the root or specific endpoints trigger page rendering, while requests prefixed with /api are directed to the API handlers. This clear separation simplifies the organization of the codebase, facilitating easier code management and enhancing the readability and supportability of the system.

The architecture is designed to improve maintainability, making it easier for developers to introduce modifications or enhancements. By separating different functionalities into distinct sections, the system not only becomes more manageable but also improves other critical aspects such as code readability and the ease of ongoing maintenance.

An overview of this architectural structure is illustrated in Appendix 6 Figure 9, which presents the UML Solver Three-Tier Architecture Diagram.

5.3.1 Server-side Structure

The server-side structure of the web application is built to handle a variety of operations, from rendering web pages to processing API requests, effectively ensuring that the system is robust and responsive. This part of the application is crucial for the security, efficiency, and scalability of the UML Solver.

The PHP server is configured to distinguish between content rendering and API service requests based on the URL structure. Standard requests to the root or specific page endpoints are processed for page rendering. The server checks if the requested page exists; if it does not, it returns a 404 error and redirects the user to a custom 404 error page. This ensures a smooth user experience by managing invalid routes gracefully.

To complement this structure, all REST API endpoints used in the UML Solver application are documented in Appendix 7 Figure 10, while all available web page routes are outlined in Appendix 8 Figure 11.

The API interaction is used in the web-application. When the server receives a request that starts with /api, it recognizes this as a call to the application's API. These requests are handled by a designated controller optimized for API interactions. The API plays a critical role in the application's functionality, handling several essential operations:

- Authentication. Verifying the identity of the user to ensure that responses are secure and personalized.
- Validation. Checking the integrity and correctness of the data received from the front end. This step is vital to prevent issues related to data format and content, which can affect database operations and business logic execution.
- Business Logic. Once the request data is validated, the server processes it through the business logic layer. This layer is where the application’s core functionalities are implemented, including data manipulation and logic processing that form the backbone of features of the UML Solver.
- Database Interaction. The server interacts with the database through the database layer, which abstracts the data storage and retrieval mechanisms. This layer facilitates communication with the database, ensuring that data queries and updates are performed efficiently and securely. The use of prepared statements

and parameterized queries helps prevent SQL injection attacks and ensures data integrity.

This multi-layered server-side structure supports not only the current functionalities but also provides a foundation for future enhancements. The separation of concerns among different layers increases the system's maintainability and allows for easier updates and scalability. Each layer can be modified or improved independently based on future requirements, ensuring that the application remains robust and adaptable as new features are developed or existing ones are refined.

Additionally, as the application faces increased load, particularly during concurrent validation of UML diagrams by students, the architecture is designed to enhance scalability. This scalability allows the system to distribute the load across multiple nodes effectively. As a result, the application can handle higher traffic and processing demands without degrading performance, ensuring that it continues to operate smoothly and efficiently even under stress. This capability is particularly important in educational environments where many students might be using the system simultaneously, requiring the application to manage large volumes of requests and data interactions in parallel.

5.3.2 Client-side Structure

The client-side structure of the web application is designed to deliver an interactive and user-friendly interface, crucial for both students and lecturers. It utilizes a combination of HTML, CSS, JavaScript, jQuery, and Bootstrap to create a responsive and intuitive environment. HTML structures the content, CSS and Bootstrap handle the styling and responsive design, ensuring the interface is attractive and functional across different devices and screen sizes. JavaScript, enhanced by jQuery, facilitates dynamic interactions and behaviors, streamlining operations such as DOM manipulation, event handling, and AJAX calls for efficient communication with the server-side REST API. The AJAX interactions are important for the seamless operation of the application, particularly when students submit UML diagrams for validation or when data needs to be fetched without refreshing the webpage. This setup allows for the smooth and continuous interaction with the server, where data can be sent and received asynchronously, enhancing the user experience by maintaining a dynamic and responsive interface.

Real-time interactivity is a key feature of the client-side implementation, with JavaScript and jQuery providing immediate feedback on the user's actions, such as validating UML diagram elements directly in the browser. This instant feedback is vital for educational tools, as it helps students correct errors and learn more effectively through immediate responses. Additionally, the use of Bootstrap enhances the client-side architecture by providing a robust framework for developing responsive layouts and components that are visually appealing and easy to use. This integration ensures that the application not only functions well but also looks good and is accessible on a variety of devices.

The client-side also includes a routing mechanism that efficiently manages the display of different views and states of the application without server-side page loads. This client-side routing helps in reducing the load on the server and speeds up the user interaction with the application, making the system more scalable and efficient. The combination of technologies creates a powerful and effective user interface that supports the educational goals of the UML diagram application. To enhance the functionality and interactivity of UML Solver, the application uses JointJS, a powerful open-source library (Mozilla Public License 2.0) that facilitates the creation and management of diagrams. JointJS is integrated into the client-side architecture to enable robust diagramming capabilities directly within the web browser. To fulfill the detailed needs of comprehensive UML diagrams, all diagram elements and their relationships were designed and coded from scratch based on the Enterprise Architect elements design. These components are defined in a dedicated JavaScript file - *uml_elements.js*. This setup ensures that each element is confirmed to UML standards and is fully functional for educational purposes. The features of each element and relationship have a custom design that aligns with best practices in diagramming tool design, enhancing the visual clarity and usability of the diagrams. The aesthetics and functionality of these designs are important as they contribute significantly to the learning experience.

For diagram validation and management, JointJS provides a built-in mechanism for serializing diagrams into JSON format. This serialization captures all aspects of the diagram, including element designs and positioning. The JSON format is particularly advantageous as it allows diagrams to be saved and later retrieved, maintaining their original layout and design. This functionality is important for scenarios where a student

would like to review a previous attempt or a lecturer needs to review a student's work. The ability to serialize and deserialize diagrams ensures that users can seamlessly validate and save their progress and reopen their diagrams without any loss of data.

This integration of JointJS into the client-side structure enriches the application's capabilities, making it a powerful tool for creating, managing, and validating UML diagrams in an educational setting. By combining HTML, CSS, JavaScript, jQuery, Bootstrap, and JointJS, the application not only provides a dynamic and user-friendly interface but also a technically robust platform for UML diagramming that meets educational needs effectively.

5.4 User Interface

This section provides an overview of the web application's user interface, which includes various pages available to end users. The screenshots are presented in the Appendix 9. The UML Solver is built with PHP and serves both as a web page renderer and a RESTful API provider. The available pages with business logic related to diagram construction and validation include:

- /tasks
- /tasks/class-diagram/{id}
- /tasks/package-diagram/{id}
- /attempts
- /task-attempts/{id}
- /statistics
- /feedback
- /lecturer/class-diagram
- /lecturer/package-diagram

The following subsections describe the user interface structure and functionalities provided for the two main user roles: students and lecturers. Particular focus is placed on pages that implement business logic related to diagram construction and validation.

5.4.1 Student-Oriented Interface

Access to the student interface requires authentication using valid credentials from the in-house Maurus system. The authentication process is handled by invoking a

predefined function in the external database service, where the student's username and password are submitted. If the credentials are valid and access is permitted, the function returns a unique student identifier. This identifier is stored in the session and is used to associate user-specific data with actions performed within the system, such as diagram submissions and feedback entries.

Following successful authentication, a CSRF token is generated by the application with a validity period of three hours. This token is attached to all authorized API requests, and the system performs token validation on each request to secure restricted endpoints.

The authentication function used is externally provided and was made available to the system developers. After logging in, students gain access to protected pages, including those for viewing available tasks, solving diagram-related exercises, reviewing attempt history, analyzing personal statistics, and submitting feedback.

Tasks Page (/tasks)

The tasks page presents students with a comprehensive overview of all available UML diagram exercises Figure 12. Tasks are grouped into collections, which are displayed on the left side of the interface. Each collection includes a progress indicator that shows how many tasks have been completed out of the total available in that group.

In the main section, tasks are listed in a table format with several filtering and sorting options. Students can search tasks by name, filter by diagram type (e.g., class diagram, package diagram), by difficulty level (e.g., beginner, advanced) or by state of solving (e.g., not started, solved). The table displays key information for each task, including:

- Name - the title of the task;
- Diagram Type - the type of diagram that is practiced with the task;
- Difficulty Level - the intended skill level;
- Status - the current solving status, such as “Not started,” “Started but not solved,” or “Solved.”

There are also two action buttons: Reset filters clears all applied filters, and Select random task chooses a random task for the student to attempt.

This interface helps students navigate and manage their progress through the diagram-solving exercises.

Class Diagram Task Page (/tasks/class-diagram/{id})

This page provides an interactive environment for solving UML class diagram tasks Figure 13. The workspace displays the initial state of a diagram, which may include predefined elements and relationships. In many cases, the workspace may be empty, requiring the student to construct the entire solution from scratch. The task description is presented on the left, containing a set of textual constraints that the diagram must fulfill. These statements define relationships between classes, cardinality, and other structural rules. Students also see a short description of the diagram type and references to the documentation about the diagram type.

The central canvas allows users to build or modify the class diagram visually.

Available actions are grouped into the following categories:

- Elements of the class diagram:
  - Class - insert a standard class;
  - Abstract Class - insert an abstract class;
  - Interface - insert an interface element.
- Relationships of the class diagram:
  - Association - creates a general link (association) between elements;
  - Aggregation - defines a "whole-part" relationship where there is no existential dependency between the whole and its parts;
  - Composition - represents strong ownership with existential dependency between the whole and its parts;
  - Realization - indicates implementation of an interface;
  - Generalization - define inheritance between classes.
- Features
  - Check Solution - validate the diagram against the reference model;
  - Load Attempt - restore a previously saved attempt;
  - Export to PlantUML - generate a textual PlantUML representation of diagram;
  - Save Screenshot - download a visual snapshot of the current diagram;
  - Clear Workspace - remove all elements from the canvas.

Additionally, a student can ask a question from the lecturer about the specific task. After submitting the diagram for validation, a results panel slides out from the right. It displays the following information: attempt number, time used, status of the attempt (e.g., Solved/Unsolved), hints and validation errors Figure 14.

Package Diagram Task Page (/tasks/package-diagram/{id})

It is similar to the class diagram task page, except that it has different elements (Package) and Relationships (Usage) Figure 15.

Attempts Page (/attempts)

The attempts page provides an overview of all task-solving attempts made by the student Figure 16. It is presented in a tabular format and supports filtering by diagram type, difficulty level, solving state as well as keyword-based search.

Each row in the table contains the following information:

- Task Name - the name of the task attempted.
- Diagram Type - the UML diagram type.
- Difficulty Level - the intended complexity of the task.
- Result - whether the attempt was Solved or Unsolved.
- Time of Attempt - a timestamp showing when the solution was submitted.

This page allows students to track their activity, review their progress over time, and identify which tasks remain unsolved. It serves as a personal task history log, supporting reflective learning and time management.

Task Attempt History Page (/task-attempts/{id})

This page allows students to review their previous solution attempts for a specific task Figure 17. The central canvas displays the result of the selected attempt, including the submitted UML diagram. On the left side, the original task description is shown, outlining the requirements the student was expected to fulfill.

The right panel provides detailed information about the selected attempt:

- Task Status - indicates whether the overall task is solved.
- Attempt Result - shows whether the specific attempt was successful.
- Attempt Number, Time Used, and Submission Timestamp.
- Feedback - displays system-generated feedback if available.

Below this, a chronological list of all previous attempts is shown. Students can select any attempt to view its content and status, enabling reflective analysis of their progress and mistakes. This page supports self-assessment and encourages iterative improvement in diagram construction.

Statistics Page (/statistics)

The Statistics page provides a personalized summary of the student's performance across all completed attempts Figure 18. The top section displays general metrics, including:

- Total number of attempts
- Number and percentage of successful and unsuccessful attempts
- Total time spent solving tasks (in minutes)
- Average time taken to reach a correct solution

Below the general overview, the page presents detailed breakdowns:

- Difficulty Level Statistics - shows the number and percentage of solved tasks by difficulty category (e.g., beginner, advanced).
- Diagram Type Statistics - aggregates performance by diagram type, such as class diagrams and package diagrams.
- Collection Statistics - summarizes performance for each task collection used in the system.

This page allows students to reflect on their progress, identify areas for improvement, and monitor their activity within the diagram-solving environment.

Feedback Page (/feedback)

The Feedback page allows students to see all the feedback that they have received based on their questions Figure 19. On the left side, a list of tasks with available feedback is displayed. Selecting a task opens a conversation view in the center of the screen, where messages between the lecturer and the student are shown in chronological order. Students can read comments, submit replies, and continue the discussion if clarification is needed. Each message includes a timestamp to track the feedback process.

On the right side, a task summary is presented, which includes:

- Task Status (e.g., Solved/Unsolved/Untried)
- Total Attempts
- Task Name and Description (including class structure requirements, attributes, and constraints).

This feature promotes individual guidance and supports formative feedback, helping students understand their mistakes and improve their solutions based on direct input.

5.4.2 Lecturer-Oriented Interface

Access to the lecturer interface is granted through authentication using the Maurus system, similarly to student login. In this case, the application invokes a separate external function, which verifies whether the user has lecturer privileges. Upon successful verification, boolean value TRUE is returned and the logged in role stored in the session. This identifier not only supports request personalization but also serves as the basis for role distinction within the system. A CSRF token with a three-hour validity is also issued, and it is required for all authenticated API interactions.

The role flag assigned during login determines both the availability of interface pages and access to protected backend endpoints. While students are directed toward learning tasks, the lecturer interface is designed primarily for supporting the preparation of tasks, and occasional manual checking of those tasks.

Lecturer-specific tools allow users to visually construct reference diagrams and define initial diagram states directly in the browser. These diagrams can be exported or imported as JSON objects. Additionally, lecturers can review student-submitted diagrams to assist with manual evaluation when needed. Built-in PlantUML export functionality further supports rapid diagram to task description conversion using LLM, making task authoring more efficient. Lecturer interface is not connected with a database. It is assumed that the lecturer has his/her own software for the task management where among other things serialized diagrams can be saved and read.

Lecturer Class Diagram Editor Page (/lecturer/class-diagram)

This page provides lecturers with a browser-based environment for manually constructing UML class diagrams. It is primarily intended to support the creation of reference solutions and initial diagram states for use in student tasks. The interface replicates the student diagram editor but includes additional features relevant for task preparation and review Figure 20. Available tools are grouped as follows: Elements (Class, Abstract Class, Interface), Relationships (Association, Aggregation, Composition, Realization, Generalization). As for the features, they are different compared with student class-diagram page:

- Export to PlantUML – generate a textual representation for documentation or task descriptions.

- Export JSON – serialize the current diagram for saving or reuse.
- Import JSON – load a previously saved diagram from a JSON structure.
- Save Screenshot – download a visual capture of the diagram.
- Clear Workspace – remove all content from the canvas.

The editor also supports keyboard shortcuts to streamline editing. This page is especially useful for quickly creating, previewing, and modifying UML diagrams in preparation for task configuration or manual student evaluation.

Lecturer Package Diagram Editor (/lecturer/package-diagram)

It is similar to the class diagram editor page except that it has different elements (Package) and Relationships (Usage) Figure 21.

5.5 Diagram Validation

In the user interface, built with the JointJS library, students create UML class or package diagrams that represent their solution to a given modeling task. These solutions are then subject to automated evaluation to provide feedback and guide the learning process. Thus, diagram validation is one of the core features of the application. Although the internal validation logic remains hidden from the student, the output of this process plays a central role in the learning experience. The student interacts directly with the results of validation, using this feedback to assess their understanding and improve their modeling skills. Each validation outcome reflects the comparison between the student's attempt and the reference diagram created by the lecturer. Based on the identified differences and reported issues, students are expected to analyze their mistakes and adjust their diagrams accordingly. As such, validation acts not only as a technical mechanism but also as a learning tool that supports comprehension of UML semantics and structure.

This section outlines two approaches considered during development: validation using LLMs validation and a rule-based algorithmic validation method. In the early stages of the project, attempts were made to apply LLM-based validation to compare and evaluate diagrams. However, after testing and analysis, it was decided to rely entirely on the algorithmic approach. The reasons for this decision, as well as the results of LLM validation experiments, are discussed in the corresponding subsection.

5.5.1 LLM Based Validation

At the early stages of system design, one of the explored directions for diagram validation was the use of LLMs. The idea was to leverage the reasoning capabilities of LLMs to compare a student-created UML diagram with a predefined reference diagram prepared by the lecturer.

The proposed validation workflow involved serializing both the student's attempt and the lecturer's reference diagram into a structured JSON format. This format captures all relevant diagram elements, such as classes, attributes, relationships, and positions. A predefined prompt template was constructed, which included the serialized content of both diagrams along with instructions for comparison. The intention was to have the LLM generate a validation report describing the differences and evaluating the correctness of the student's solution based on expected modeling rules:

“ *# Role*

Thorough and detail-oriented university lecturer, who validates UML diagrams made by students.

 # Guidelines

Compare the following two UML diagrams provided in JSON format: one is a student attempt, and the other is the reference solution. Identify key differences in classes, attributes, and relationships. Evaluate whether the student diagram correctly implements the structure and semantics of the reference.

 # Format

Provide a clear and concise validation report.”

A manually defined set of validation criteria was established in collaboration with a lecturer. These criteria described typical modeling mistakes and semantic violations that should be detected during the comparison. For example, the presence or absence of classes, incorrect associations, missing inheritance relations, or naming mismatches were among the expected validation targets. The goal was for the LLM to analyze the two diagrams and provide a human-readable explanation of whether the student’s solution met the expectations, including a description of specific errors. The use of a locally installed LLM gives total control over the software and avoids costs associated with cloud-based services.

Several LLMs were tested locally for this task:

- mistral-7b-instruct (Q4_K_M quant method),
- phi-2 (Q4_K_M quant method),
- tinyllama (Q4_K_M quant method),
- deepseek-coder-6.7b-instruct (Q4_K_M quant method).

All models were run on a local machine to simulate realistic performance in a constrained environment, similar to what might be encountered in institutional or offline use cases. During these experiments, it was observed that some models occasionally produced coherent and useful feedback. However, there were significant limitations that ultimately led to the decision to abandon LLM-based validation as the primary mechanism.

One of the main issues encountered was the non-deterministic nature of LLM outputs. Sending the same input prompt to the model multiple times could yield slightly or even significantly different responses. This made it difficult to ensure consistency and reliability in validation, which are critical in educational systems. Students relying on LLM feedback might receive varied explanations for identical submissions, which could lead to confusion or mistrust in the system.

Another challenge was related to the clarity and precision of the output. In many cases, the generated validation feedback lacked logical structure or actionable insights. Some responses were vague or overly generic, making it difficult to determine whether the student's solution was correct or not. Additionally, LLMs occasionally misinterpreted the structure or intent of the diagrams, leading to incorrect conclusions in the generated validation report.

Performance was also a factor. Unlike rule-based validation, which can be executed almost instantly, LLMs require additional time to process the input and generate a response. Even with smaller models, this delay posed a risk of degrading the user experience, especially when scaled to multiple concurrent users in a classroom environment.

It is important to note that while the LLM-based approach demonstrated some potential especially in generating natural language descriptions and comparisons. It lacked the robustness, reliability, and transparency needed for precise validation. Furthermore, the inability to directly enforce or explain custom validation rules limited its applicability in contexts where strict grading and consistent feedback are essential. However, LLMs could be well-suited for evaluating pragmatic quality, e.g., naming consistency and placement of elements on diagrams. They may also be useful for assisting in basic syntactic validation by identifying obvious structural errors or suggesting corrections in a human-readable format.

Based on these findings, the decision was made to transition fully to a deterministic, rule-based algorithmic validation approach. The LLM validation experiments nevertheless provided valuable insights into the challenges of using AI models for semantic comparison and confirmed the need for a more predictable and structured evaluation method in the context of educational UML modeling.

5.5.2 Rule-based Algorithmic Validation

To ensure consistent and reliable feedback for students, the system adopts a rule-based algorithmic approach to diagram validation. This method was chosen over probabilistic or AI-based alternatives to guarantee determinism, transparency, and full control over the validation logic. The goal of this validation approach is to identify and classify mistakes in UML diagrams submitted by students, comparing them to a reference diagram created by the lecturer. The process is fully automated, precisely defined, and provides students with structured and localized feedback, supporting their learning and helping them improve their modeling skills over time.

The core validation algorithm can be conceptualized as a deterministic, rule-based comparator that operates in multiple phases. Its primary input is a student-submitted UML diagram serialized in JSON format, paired with a task identifier used to retrieve the corresponding reference model. The purpose of the algorithm is to perform an exact syntactic and semantic comparison between the student's submission and the reference solution, thereby identifying all deviations that violate the expected model. The process is structured as follows:

1. **Normalization Phase**. Both the student and reference diagrams are parsed and internally normalized. This ensures consistent formatting and prepares data for

precise comparison. Normalization includes unifying naming conventions (e.g., case-insensitive comparison) and extracting key model elements such as classes, packages, relationships, attributes, and methods.

2. **Structural Integrity Phase.** The algorithm verifies that the overall count and structure of diagram components match the reference. Functions involved:
    - *checkNodesForDuplicate()*,
    - *checkRelationshipsForDuplicate()*,
    - *checkPackagesForDuplicate()*,
    - *checkTotalCounts()*.
3. **Element-wise Comparison Phase.** In this phase, the algorithm performs a systematic comparison across multiple dimensions. Functions involved:
    - *compareNodes()*,
    - *compareRelationships()*,
    - *comparePackages()*,
    - *compareAttributesAndMethods()*.
4. **Error Aggregation Phase.** All mismatches are categorized into predefined error types and returned in a structured format for further interpretation. Unlike probabilistic or heuristic-based systems, this algorithm guarantees deterministic output: the same input will always produce the same result. It is exhaustive, non-interruptible, and strictly aligned with the reference model, providing a full diagnostic overview to support student learning.

The validation process begins when a student opens a task on the /task/{taskId} page. Each task is predefined and stored in the PostgreSQL database in the task table. The task definition includes the diagram type (e.g., class or package), a difficulty level, the description of the assignment, and optionally an initial diagram state. If an initial state is provided, it is automatically deserialized and rendered in the browser using the JointJS library. This serves as a starting point for the student's work. Otherwise, the student begins with a blank canvas and builds the diagram from scratch based on the instructions provided.

At any point during his/her work the student may initiate validation. At this point, the diagram constructed in the browser is serialized into a JSON format. This JSON contains all relevant diagram data (classes or packages, relationships, attributes, and

methods) as well as the visual layout of elements. The diagram is then submitted to the backend for validation. Each attempt is stored in the *Task_attempt* table, along with metadata such as the task ID, timestamp, time spent, student ID, and the full serialized solution Figure 6.

On the backend, the validation logic retrieves the lecturer's reference diagram for the corresponding task. This reference is also stored in the database as part of the *Task* table in the *solution* field. The type of validation that will be applied is automatically selected based on the diagram type associated with the task. For example, tasks involving class diagrams invoke the class diagram validator, while package diagram tasks trigger a different, specialized validator.

The validation procedure is broken into multiple precise steps, each addressing different structural and semantic aspects of the diagram. For class diagrams, the following core steps are executed:

- *checkNodesForDuplicate()* identifies duplicate class definitions. This step verifies that each class appears only once in the diagram and that no two classes share the same name, which could otherwise cause ambiguity or logical errors in interpretation.
- *checkRelationshipsForDuplicate()* detects repeated associations or generalizations between the same classes. Redundant relationships of the same type and direction are not allowed, as they clutter the diagram and do not provide additional meaning.
- *checkTotalCounts()* ensures the total number of key elements in the diagram (classes, relationships, attributes, and methods) matches the expected values defined in the reference model. This helps detect models that are structurally complete but may contain extra or missing components not caught in earlier checks.
- *compareNodes()* compares the presence and naming of class elements. This includes checking whether all required classes from the reference diagram exist in the student's solution and whether their names match. It also detects extra classes that are not part of the expected model.
- *compareRelationships()* checks the correctness of all relationships between classes. This includes verifying the type of relationship (e.g., association,

aggregation, generalization), the source and target classes, and additional properties such as role names, multiplicities, and labels.

- *compareAttributesAndMethods()* verifies the correctness of all class attributes and methods. It checks for the presence of required fields and operations, validates data types, multiplicities, and visibility modifiers (e.g., public, private), and ensures that method signatures match the reference solution.

Similarly, for package diagrams, the validation follows a parallel structure with functions tailored to the semantics of packages and their relations:

- *checkPackagesForDuplicate()* checks for duplicate package definitions based on name and identity. Each package in a diagram should be uniquely named and defined only once. Duplicate packages often indicate redundant or unintended modeling actions and must be resolved before the diagram can be considered valid.
- *checkRelationshipsForDuplicate()* identifies any repeated relationships between packages. A valid package diagram should not contain multiple identical links between the same pair of packages with the same type and label. This check helps enforce the clarity and minimalism of the model, avoiding redundant or ambiguous associations.
- *comparePackages()* compares the list of packages in the student's diagram to those in the reference solution. It detects both missing and extra packages. This step is critical to ensure that all required structural components of the model are present and that no unintended elements were added.
- *compareRelationships()* verifies that the relationships between packages match the expected set in both direction, type, and labeling. It ensures that dependencies, nesting, and usage relationships are all correctly represented. The validator also checks for semantic correctness in the role names and relationship labels, when applicable.

Each validation step may produce one or more errors, all of which are strictly typed and linked to a comprehensive taxonomy of error types. This taxonomy is maintained in the *Problem_type* table, and each error type is associated with multilingual translations stored in the *Problem_type_translation* table. These translations ensure that students

receive feedback in their selected language (Estonian or English) without loss of specificity or meaning. Detailed explanations for errors are stored separately in i18n localization files (est.php - for Estonian language, eng.php - for English language) and rendered on the frontend after validation.

Errors detected during validation are collected, grouped by category, and returned to the frontend. Each error includes a type identifier, a brief message summarizing the issue, and a more detailed description. All this data is prepared based on the selected UI language before being returned to the user. Errors are displayed as a list besides the diagram area. While the system does not currently support visual highlighting of erroneous elements within the diagram itself, the structured error list allows students to read, interpret, and correct their models accordingly.

Validation results are also saved in the database for historical reference. Each submission is recorded in the *Task_attempt* table with *submitted_solution* field (containing the serialized JSON), a text about errors, and a boolean flag *is_result_correct* indicating whether the solution was fully correct. Students can view the history of all their attempts across all tasks on a dedicated page. Additionally, the system allows a student to reload any previous attempt and use it as a starting point for a new solution.
To avoid overwhelming the student with excessive feedback, the number of visible error messages is configurable. By default, all errors are displayed, but lecturers can adjust the *max_number_of_result_errors* parameter in the system's configuration file. This allows showing only the most important issues, which may help students focus on key problems first before diving into more detailed refinements.

Alongside formal validation, the system includes a configurable hint mechanism. Lecturers can prepare hints in advance and store them in the hint table. These hints are tied to specific tasks and are revealed to the student only after a certain number of incorrect attempts. This behavior is controlled by the *min_count_to_see_hints* setting. For example, setting this value to 5 means that a student must submit five incorrect solutions before receiving a contextual tip. This approach is intended to encourage self-directed problem solving while still offering support if progress stalls.

By relying on a deterministic, rule-based validation framework, the system ensures consistency, fairness, and transparency in automated evaluation. Unlike probabilistic or AI-driven systems, every rule is explicitly implemented and returns reproducible results. This makes the system particularly well-suited for educational contexts, where clear and repeatable feedback is crucial for effective learning. The combination of structured validation logic, typed error definitions, localized explanations, and gradual guidance through hints provides a complete ecosystem for UML diagram practice, analysis, and iterative improvement. All defined error types with descriptions are presented in Table 5.

6 Analysis and Results

This chapter analyzes the results that were achieved as the result of writing the thesis and reflects the work that was done.

6.1 Analysis of Students Survey Responses

To evaluate the practical effectiveness and usability of the UML Solver system, a survey was conducted. It collected 17 answers. 12 answers were given by students of "Databases I" course (i.e., the target audience of the tool) who used the application as part of their coursework. The responses provide insight into the overall user experience, including the clarity of instructions, effectiveness of feedback, and perceived educational value. The students firstly had to solve six tasks in UML Solver. These tasks covered both class and package diagrams and used all the model element types that would be used in the tasks in "Databases I" course (they did not cover methods, interfaces, and realize relationships). Most of the tasks listed a set of statements and asked students to create a diagram based on these (see an example from Figure 22). Based on these students had to draw a diagram. Firstly, the lecturer (the supervisor) had to construct reference diagrams (see an example from Figure 23) and thus also test the interface that is meant for the lecturers. Students had to solve all six tasks and answer the questions to get some extra points in the "Databases I" course. Thus, they had to use the tool quite thoroughly. In average they made 32 attempts to validate their models and spent 45 minutes with all the tasks. The findings are summarized below.

The first question addressed the usability of the diagram editor, specifically how easy it was to create and edit diagram elements such as classes, packages, and relationships Figure 24. The results show that the vast majority of users rated the experience positively, with 52.9% selecting 4 and 29.4% selecting 5 on a 5-point Linear scale. Notably, no respondents chose ratings below 3, indicating general ease of use and user satisfaction with the diagramming interface. This suggests that the system's core interaction design is effective and requires only minor refinements, if any.

The second question examined the clarity of task descriptions Figure 25. Here too, the responses were largely positive, with 76.5% of students rating the descriptions either 4 or 5. Only one respondent gave a score of 2, which might reflect an isolated issue or a user-specific misunderstanding. The majority view confirms that task instructions are generally well-structured and accessible, though some marginal improvements in wording or contextual support could further enhance comprehension.

An essential feature of the UML Solver system is its automated feedback mechanism. The third question assessed whether this feedback was clear and understandable Figure 26. Responses were overwhelmingly favorable, with 64.7% giving the maximum score of 5, and 23.5% selecting 4. No respondents rated this aspect below 3. This indicates that the system successfully communicates feedback in a manner that supports student learning, reinforcing correct modeling practices and guiding users toward improvement.

Next, students were asked whether their UML diagramming skills had improved as a result of using the system Figure 27. The responses showed a positive trend, with 64.7% choosing 4 or 5. However, 29.4% selected a neutral score of 3, and one student chose 2, suggesting that while most users experienced some level of learning gain, the system's pedagogical impact could vary depending on the user's engagement level or prior knowledge.

Finally, participants were asked how likely they would be to recommend UML Solver to other students Figure 28. The results were highly encouraging, with 88.2% giving a score of 7 or above, and 41.2% selecting the maximum score of 10. This high level of endorsement underscores the system's practical value and its perceived benefit in a learning environment. The last question allowed us to calculate the Net Promoter Score (NPS) [54]-[55]. NPS is calculated as the percentage of Promoters (ratings 9–10) minus the percentage of Detractors (ratings 0–6). In this survey, 58.8% of respondents were Promoters, and 0% were Detractors, resulting in an NPS of 59 (58.8% − 0% = **~59%**). The NPS value is **59%** and it is considered a strong score, showing that the majority of users are not only satisfied but also willing to recommend the system to others. In practical terms, this means the tool meets user expectations and performs well in its educational context.

In conclusion, the survey responses validate that UML Solver is an effective and educational suitable tool for practicing UML modeling. Users found the interface intuitive, the instructions clear, and the automated feedback mechanism helpful. It was reflected with answers on "Positive Feedback Question" Figure 30. Moreover, the majority of respondents reported an improvement in their skills and indicated a strong willingness to recommend the tool to others. Survey's results clearly demonstrate that the tool fulfills its intended purpose and effectively addresses the specific needs within its niche. It fully meets the previously defined functional requirements, confirming its practical value and reliability in real use cases. These findings support the tool's continued development and integration into educational contexts. The feedback also pointed to the minor mistakes or usability issues with answers on "Negative Feedback Question" Figure 29 that the author has already solved or plans to solve immediately.

6.2 Discussion

The implementation of checking syntax and semantics in our platform highlights its capabilities and limitations. While the existing modeling tools excel in facilitating the creation, editing, and visualization of UML diagrams, their limitation lies in the automated validation of these diagrams against predefined standards or reference models. This feature is especially important in educational settings where the accuracy of diagrams and adherence to specific diagramming conventions are critical for effective learning and assessment. However, commercial UML tools do not include built-in functionalities for such in-depth validation with reference models. Although the scientific literature describes tools with goals similar to UML Solver (see section 3.1), none of these offer a set of functionalities as complete as UML Solver provides.

6.3 Limitations

The developed system has certain limitations that were defined by the scope of the current thesis. One of the key limitations is the support for only two types of UML diagrams - class diagrams and package diagrams. While these are among the most commonly used diagram types in software modeling and education, the UML specification includes other diagram types such as activity diagrams, use case diagrams, sequence diagrams, and state machine diagrams. These were not included in the current

implementation due to time constraints and the focus on delivering a functional and stable foundation.

In addition, the UML specification itself is extensive and includes a wide range of structural and behavioral features. Some of these features are rarely used in practice and are more relevant in highly specific modeling scenarios. For example, class diagrams support advanced constructs such as association classes, which represent associations that also have attributes or operations. These advanced features were not implemented in this project, as the primary objective was to focus on widely adopted elements that are most relevant for typical use cases in learning environments. Based on these priorities, a specific subset of UML functionality was selected, and the system requirements were defined accordingly.

6.4 Reflection of the Work Done

This section presents a reflective analysis of the development process, highlighting the strengths, challenges, and key lessons learned throughout the project. The goal is to critically assess the quality of the implemented work, identify areas of success, and recognize aspects that could have been handled more effectively. By doing so, it becomes possible to draw meaningful conclusions about the author's professional development and to outline practical improvements for similar projects in the future.

The reflection is structured into three sections. The first section, Things that Went Well, focuses on the practices and decisions that positively influenced the project. It covers elements such as communication, technology choices, implementation techniques, and user feedback. The second section, Things that Went Poorly, describes specific difficulties encountered during the project. It highlights the technical and organizational aspects that limited efficiency or caused complications during development and deployment. The third section, Things to Do Differently if Repeating the Work, provides an evaluation of what should be improved if the project were to be done again. This includes practical recommendations based on the challenges faced and emphasizes the importance of early planning, automation, and architectural foresight.

Together, these sections provide a comprehensive and balanced view of the work done and support a deeper understanding of the development process from both a technical and organizational perspective.

6.4.1 Things that Went Well

Several aspects of the project were executed successfully and contributed positively to the overall outcome. One of the key strengths was the regular and structured communication with the supervisors. Weekly meetings were held to review the progress made by the author and to plan the tasks for the upcoming week. This iterative approach allowed for constant monitoring and ensured that the project remained on track. The communication itself was efficient and responsive. The supervisors provided timely feedback and support, which helped maintain alignment throughout the development process. This synchronization not only minimized misunderstandings but also served as a strong motivational factor for the author, encouraging consistent and high-quality work.

The choice of technologies was another element that went well. The technology stack was defined at the beginning of the project, which allowed for stable development without the need for major changes. Although the stack was predefined, the author was responsible for designing the system architecture. This architectural plan was reviewed and approved by the supervisors before implementation, ensuring both technical feasibility and alignment with project goals.

From a functional perspective, the application successfully met all of the initial system requirements. The system was completed in time and demonstrated its practical applicability within the context of a database course. This confirmed that the implemented functionality was both relevant and usable in a real academic environment.

In terms of implementation quality, the author focused on creating reusable components, which was especially important due to the need to support both class diagrams and package diagrams. Shared elements of the system were abstracted and reused effectively. As a result, the codebase remains flexible and can be easily extended to support additional diagram types in future development phases.

Lastly, the application received feedback from a wide group of users. The responses were collected, analyzed, and used to evaluate user satisfaction. Survey statistics showed a generally positive assessment of the web application. This feedback confirmed that the system was not only functional but also well-received by its target audience.

6.4.2 Things that Went Poorly

One of the problems that emerged during the project was related to the localization of the user interface. In the early stages of development, the design was created in English. The author did not have prior experience with multilingual interfaces and did not anticipate the challenges that would follow. When the Estonian translation was added later, it became clear that the layout did not adapt well to text variations. Phrases in different languages can have different lengths, and this has a direct impact on the visual structure of the application.

In many cases, buttons, labels, and other elements became misaligned or visually distorted. Some text fields overflowed or were cut off entirely. These problems affected both the usability and the appearance of the system. As a result, several parts of the interface had to be redesigned after localization was introduced. This reactive redesign required extra development time and effort. The situation clearly demonstrated the importance of planning for localization from the very beginning. If the layout had been built with flexibility in mind, such as by using dynamic sizing and content-aware containers, many of these issues could have been avoided.

6.4.3 Things to Do Differently if Repeating the Work

One of the key lessons learned during the development and deployment of the application was the underestimated complexity of localization. At first, it seemed like a simple task. However, in practice, adding multilingual support to a web application takes at least one week. In most cases, two weeks is a more realistic estimate. Each language version must be carefully tested. It is important to check if all content is visible and correctly displayed. User interface elements should remain properly aligned. Translated strings can be longer or shorter depending on the language. This affects the layout and positioning of page elements. The author had no prior experience with

localization. As a result, the number of required adjustments was unexpected. These adjustments were necessary to maintain a consistent and functional design across all supported languages.

The deployment process also had its difficulties. These could have been avoided through automation. The production server contained sensitive user data. Because of this, the author did not have direct access. The initial setup was difficult. This was due to differences between the development and production environments. Configuration inconsistencies appeared as a result. A GitLab CI/CD pipeline could have helped. It would have made the deployment process faster, safer, and more reliable. In this project, the code was sent as an archive. It was deployed manually by the supervisor. This approach caused delays and created risks of human error.

Another issue came from the hosting environment. The application was deployed on a server that already hosted another PHP-based system (SQL Solver [16]). Both systems used the same Apache server. This caused session collisions and affected the platform's stability. Fortunately, the issue was easy to fix once it was identified. However, it was not discussed at the beginning of the project. This led to avoidable complications. During development, the application was accessed through http://localhost:8000. In production, the URL changed. This difference required extra adjustments. In particular, routing and asset linking had to be modified. Proper planning at the start could have prevented these problems.

6.5 Further Work

At the time of writing the thesis LLMs already showed a great potential in processing both images and text, reasoning based on these, and generating all kinds of things based on these. However, small-scale tests on checking diagrams with LLMs revealed inconsistent results (e.g., the same LLM can yield different outcomes for the same diagram) and the possibility of both false positives (identifying non-existent mistakes) and false negatives (missing actual mistakes). Thus, it was decided that in the first version of the software the LLMs will not be utilized as assessment assistants. However, we envision that in the future LLMs could be used to assess the answers. This approach will leverage LLMs to enhance the validation process by comparing textual

representations of student-generated diagrams against those provided by lecturers. This would have at least two advantages:

1. It would be possible to evaluate the pragmatics aspect of models, i.e., their comprehensibility.
2. It would simplify the enhancement of the system. Current algorithmic approach of evaluating models means that addition of each new model element type or diagram type would require the changes and extensions in the algorithm. However, if the system uses a LLM for the assessment, then after changing the diagram editor it should be possible to change prompts that the system sends to a LLM. Ideally, it could be done without changing program code at all.

If the system would support both algorithmic and AI based assessment, then the system should incorporate flexible validation settings, allowing lecturers to configure and toggle between validation methods. This adaptability will cater to different learning environments and teaching methodologies, thereby enhancing the educational impact of the tool. Furthermore, logging all the task attempts of students, including types of mistakes, makes it possible after some time (to collect a reasonable amount of task attempts) to analyze these answers to find things that are difficult for students in terms of UML. This information can be used to adjust teaching strategies.

Currently the system does not support Uni-ID digital identity of the university to log in because this functionality was not considered to have the highest priority. Future work could include adding the support of Uni-ID because it will reduce the number of passwords that a student has to remember.

In addition to supporting additional model element types of class and package diagrams and supporting additional types of diagrams, another possible functionality would be highlighting incorrect model elements.

Feedback of potential users pointed to the small functionality changes (like implementing undo functionality, creating elements with copy/paste method, changing the relationship type without deleting and recreating it) that improve user experience but might be quite complicated to implement.

7 Summary

As a result of the work, a web-based software called UML Solver was developed using PHP-8 and JavaScript, integrating JavaScript diagramming library JointJS for UML diagram creation. UML Solver features interactive tools that enable students to create UML class and package diagrams based on tasks defined by the lecturer while incorporating automated validation. In the tool an algorithmic approach for validation was implemented although AI-based evaluation employing LLMs to analyze textual representations of diagrams was also considered. PostgreSQL is used for database storage, maintaining records of diagram structures, user interactions, and validation results. Reference diagrams as well as diagrams created by students are saved in columns with JSONB type. Creating the interface for managing tasks and analyzing answers was outside the scope of the work but it has already been implemented by the supervisor of the work, making the system fully functional.

The choice of PHP-8 and pure JavaScript ensure compatibility with a broad range of web environments, provides scalability, and maintains a lightweight yet robust infrastructure for real-time interaction. By avoiding reliance on heavy front-end frameworks, the system remains adaptable and efficient in handling diagram rendering and validation tasks. The use of these languages makes the system as easily maintainable and portable as possible for its future maintainer. Frontend behaviour was created using plain JavaScript and jQuery without additional frameworks.

UML Solver includes a user interface that ensures a seamless experience for students. Feedback mechanisms were embedded to assist learners in identifying and correcting errors. Additionally, progress analytics and statistics views for students were incorporated, allowing students to track their progress. The web-based nature of the system ensures accessibility, allowing students to practice UML diagramming from any location with an internet connection through the browser. The system was validated by a set of interested persons, most of whom were students who were learning the course “Databases I” at the time (i.e., the target audience of the tool) and received in general positive feedback.

References

[1] R. S. Pressman and B. R. Maxim, *Software engineering: a practitioner's approach*, 8. ed. in McGraw-Hill series in computer science. New York, NY: McGraw-Hill Education, 2015.

[2] M. Ozkaya and F. Erata, "A survey on the practical use of UML for different software architecture viewpoints", Information and Software Technology, vol. 121, p. 106275, 2020.

[3] M. Fowler, Ed., *UML distilled: a brief guide to the standard object modeling language*, 3. ed, 16. printing. in The Addison-Wesley object technology series. Boston, Mass.: Addison-Wesley, 2010.

[4] "Intro to UML 2.5 diagram types and templates," Nulab. Accessed: Apr. 20, 2025. [Online]. Available: https://nulab.com/learn/software-development/intro-uml-diagram-types-templates/

[5] "How to create static diagrams in Unified Modeling Language." Accessed: Apr. 20, 2025. [Online]. Available: https://www.redhat.com/en/blog/static-UML-diagramming

[6] E. Ramollari and D. Dranidis, "StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design".

[7] "Võimalikud uurimistöö meetodid. Disainiteadus." Accessed: Apr. 20, 2025. [Online]. Available: https://maurus.ttu.ee/loputoo.php

[8] "Software Architecture and Design - IDU1550." Accessed: Apr. 20, 2025. [Online]. Available: https://ois2.taltech.ee/uusois/aine/IDU1550

[9] *The Data Model Resource Book, Volume 1*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/the-data-model/9780471380238/

[10] *The Data Model Resource Book, Vol. 2: A Library of Data Models for Specific Industries*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/the-data-model/9780471353485/

[11] *The Data Model Resource Book, Volume 3: Universal Patterns for Data Modeling*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/the-data-model/9780470178454/

[12] *Data Model Patterns*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/data-model-patterns/9780133488654/

[13] *Patterns of Data Modeling*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/patterns-of-data/9781439819906/

[14] *Analysis Patterns: Reusable Object Models*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/analysis-patterns-reusable/9780134271453/

[15] *Design Patterns: Elements of Reusable Object-Oriented Software*. Accessed: Apr. 20, 2025. [Online]. Available: https://learning.oreilly.com/library/view/design-patterns-elements/0201633612/

[16] L. Demidov, V. Suprun, and E. Eessaar, "Creating a Web Environment for the Self-Practicing of SQL Tasks," Jun. 2024, Accessed: Apr. 20, 2025. [Online]. Available: https://digikogu.taltech.ee/et/item/b5d1ec7f-c49d-4299-bddc-b1c84e77e10d

[17] "Figma: The Collaborative Interface Design Tool," Figma. Accessed: Apr. 20, 2025. [Online]. Available: https://www.figma.com/

[18] "KanbanFlow - Lean project management. Simplified." Accessed: Apr. 20, 2025. [Online]. Available: https://kanbanflow.com/

[19] "Lõputöö soovitused - 5.6.2 Kasutatavuse hindamine." Accessed: Apr. 20, 2025. [Online]. Available: https://staff.ttu.ee/~erki.eessaar/loputood_soovitused.html#5_6_2

[20] R. Jebli, J. E. Bouhdidi, and M. Y. Chkouri, "Assessing Students' UML Class Diagrams: a New Automated Solution," in *2023 7th IEEE Congress on Information Science and Technology (CiSt)*, Dec. 2023, pp. 431–435. doi: 10.1109/CiSt56084.2023.10409936.

[21] S. Modi, H. A. Taher, and H. Mahmud, "A Tool to Automate Student UML diagram Evaluation," *Acad. J. Nawroz Univ.*, vol. 10, no. 2, pp. 189–198, Jun. 2021, doi: 10.25007/ajnu.v10n2a1035.

[22] E. Ramollari and D. Dranidis, "StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design".

[23] S. A. Turner, M. A. Pérez-Quiñones, and S. H. Edwards, 'minimUML: A minimalist approach to UML diagramming for early computer science education', Journal on Educational Resources in Computing, vol. 5, no. 4, p. 1, Dec. 2005.

[24] D. Dranidis, I. Stamatopoulou, and M. Ntika, "Learning and Practicing Systems Analysis and Design with StudentUML," in *Proceedings of the 7th Balkan Conference on Informatics Conference*, in BCI '15. New York, NY, USA: Association for Computing Machinery, Sep. 2015, pp. 1–8. doi: 10.1145/2801081.2801104.

[25] C. Alphonce and P. Ventura, "QuickUML: a tool to support iterative design and code development," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, in OOPSLA '03. New York, NY, USA: Association for Computing Machinery, Oct. 2003, pp. 80–81. doi: 10.1145/949344.949359.

[26] D. Py, L. Auxepaules, and M. Alonso, 'Diagram, a Learning Environment for Initiation to Object-Oriented Modeling with UML Class Diagrams', Journal of Interactive Learning Research, vol. 24, no. 4, pp. 425–446, 2013.

[27] R. W. Hasker, 'UMLGrader: an automated class diagram grader', J. Comput. Sci. Coll., vol. 27, no. 1, pp. 47–54, Oct. 2011.

[28] O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in

conceptual modeling," *IEEE Softw.*, vol. 11, no. 2, pp. 42–49, Mar. 1994, doi: 10.1109/52.268955.

[29] "The Unified Modeling Language Specification Version 2.5.1." Accessed: Apr. 20, 2025. [Online]. Available: https://www.omg.org/spec/UML/

[30] "Enterprise Architect 12." Accessed: Apr. 20, 2025. [Online]. Available: https://sparxsystems.com/products/ea/12/

[31] "Model Expert Homepage," eaTeamWorks. Accessed: Apr. 20, 2025. [Online]. Available: https://www.eateamworks.com/modelexpert/

[32] D. Carr and S. Else, "State of enterprise architecture survey: Results and findings," *Enterp. Archit. Prof. J.*, vol. 13, no. 05, 2018.

[33] "Third Party Extensions for Enterprise Architect | Sparx Systems." Accessed: Apr. 20, 2025. [Online]. Available: https://sparxsystems.com/products/3rdparty.html

[34] A. Valtna and E. Eessaar, "A Software to Generate Human Language Sentences from Enterprise Architect CASE Tool\s UML Class Diagrams," Jan. 2022, Accessed: Apr. 20, 2025. [Online]. Available: https://digikogu.taltech.ee/et/item/6f8ed2e4-5adb-4944-9249-3860eab1ae1c

[35] "StarUML." Accessed: Apr. 20, 2025. [Online]. Available: https://staruml.io/

[36] "UMLetino - Free Online UML Tool for Fast UML Diagrams." Accessed: Apr. 20, 2025. [Online]. Available: https://www.umletino.com/umletino.html

[37] UMLet, *UMLet - github*. (Apr. 15, 2025). JavaScript. Accessed: Apr. 20, 2025. [Online]. Available: https://github.com/umlet/umlet

[38] "yEd Live," yWorks, the diagramming company. Accessed: Apr. 20, 2025. [Online]. Available: http://www.yworks.com/yfh

[39] "Creately | Visual Collaboration & Diagramming Platform," Creately. Accessed: Apr. 20, 2025. [Online]. Available: https://creately.com/

[40] "Lucid visual collaboration suite." Accessed: Apr. 20, 2025. [Online]. Available: https://lucid.app/

[41] "EMF Compare | Overview." Accessed: May 10, 2025. [Online]. Available: https://eclipse.dev/emf/compare/overview.html

[42] "Ideal Modeling & Diagramming Tool for Agile Team Collaboration." Accessed: May 10, 2025. [Online]. Available: https://www.visual-paradigm.com/

[43] "Rational Software Architect Designer | IBM." Accessed: May 10, 2025. [Online]. Available: https://www.ibm.com/products/rational-software-architect-designer

[44] "Baselines in Enterprise Architect | Sparx Systems." Accessed: May 10, 2025. [Online]. Available: https://sparxsystems.com/resources/baseline/

[45] Z. Xing and E. Stroulia, "UMLDiff: an algorithm for object-oriented design differencing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, in ASE '05. New York, NY, USA: Association for Computing Machinery, Nov. 2005, pp. 54–65. doi: 10.1145/1101908.1101919.

[46] "SourceGear | DiffMerge." Accessed: Apr. 20, 2025. [Online]. Available:

https://www.sourcegear.com/diffmerge/

[47] “LeetCode - The World’s Leading Online Programming Learning Platform.” Accessed: May 10, 2025. [Online]. Available: https://leetcode.com/

[48] “HackerRank - Online Coding Tests and Technical Interviews.” Accessed: May 10, 2025. [Online]. Available: https://www.hackerrank.com/

[49] “CodeSignal - Discover and Develop In-Demand Skills,” CodeSignal. Accessed: May 10, 2025. [Online]. Available: https://codesignal.com/

[50] “CodeChef | CodeChef: Practical coding for everyone.” Accessed: May 10, 2025. [Online]. Available: https://www.codechef.com/

[51] S. Dahmen-Lhuissier, “EN 301 549 V3 the harmonized European Standard for ICT Accessibility,” ETSI. Accessed: Apr. 20, 2025. [Online]. Available: https://www.etsi.org/human-factors-accessibility/en-301-549-v3-the-harmonized-european-standard-for-ict-accessibility

[52] “DB-Engines Ranking,” DB-Engines. Accessed: Apr. 20, 2025. [Online]. Available: https://db-engines.com/en/ranking

[53] M. Fowler, *Patterns of enterprise application architecture*, Nineteenth printing. in The Addison-Wesley Signature Series. Boston San Francisco New York Toronto Montreal London Munich Paris Madrid Capetown: Addison-Wesley, 2013.

[54] “Beyond the NPS: Measuring Perceived Usability with the SUS, NASA-TLX, and the Single Ease Question After Tasks and Usability Tests,” Nielsen Norman Group. Accessed: May 11, 2025. [Online]. Available: https://www.nngroup.com/articles/measuring-perceived-usability/

[55] “Net Promoter Score: What a Customer-Relations Metric Can Tell You About Your User Experience,” Nielsen Norman Group. Accessed: May 11, 2025. [Online]. Available: https://www.nngroup.com/articles/nps-ux/

Appendix 1 – Non-Exclusive Licence for Reproduction and Publication of a Graduation Thesis[1]

I Valentin Djomin

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Creating a Web Environment for the Self-Practicing of Tasks for Creating UML Class and Package Diagrams”, supervised by Erki Eessaar and Priit Järv.
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

12.05.2025

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Figma Prototype Views

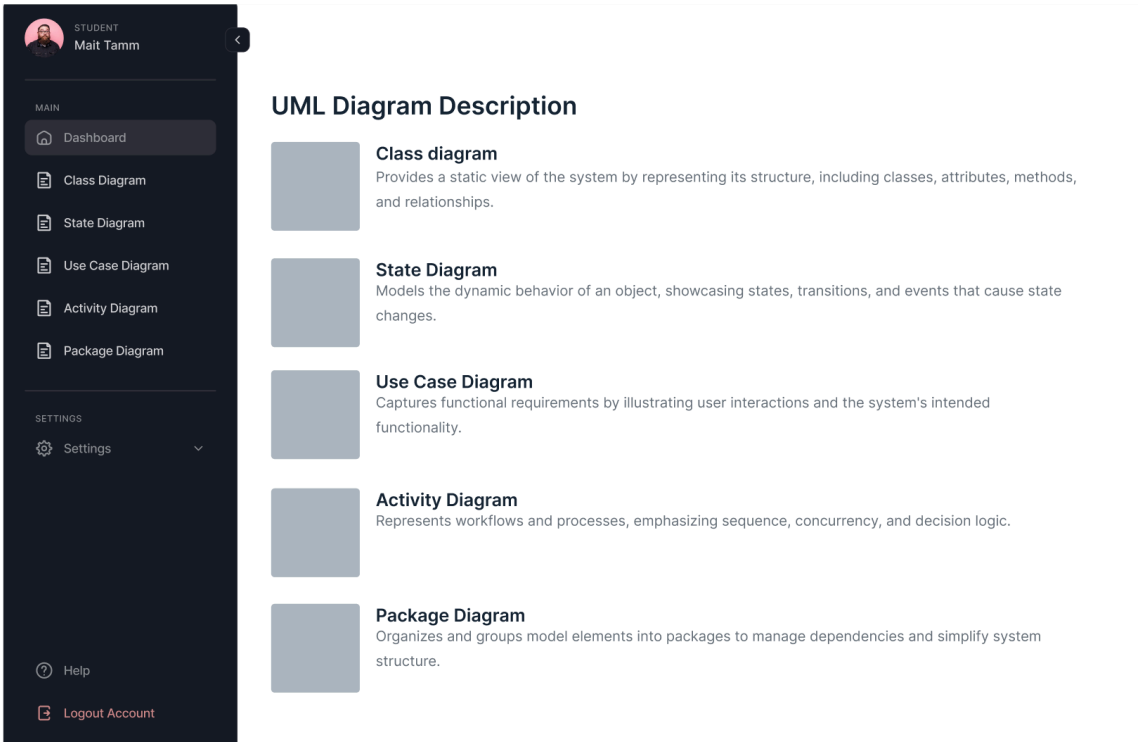

Figure 1. UML Diagram Types Overview Page.

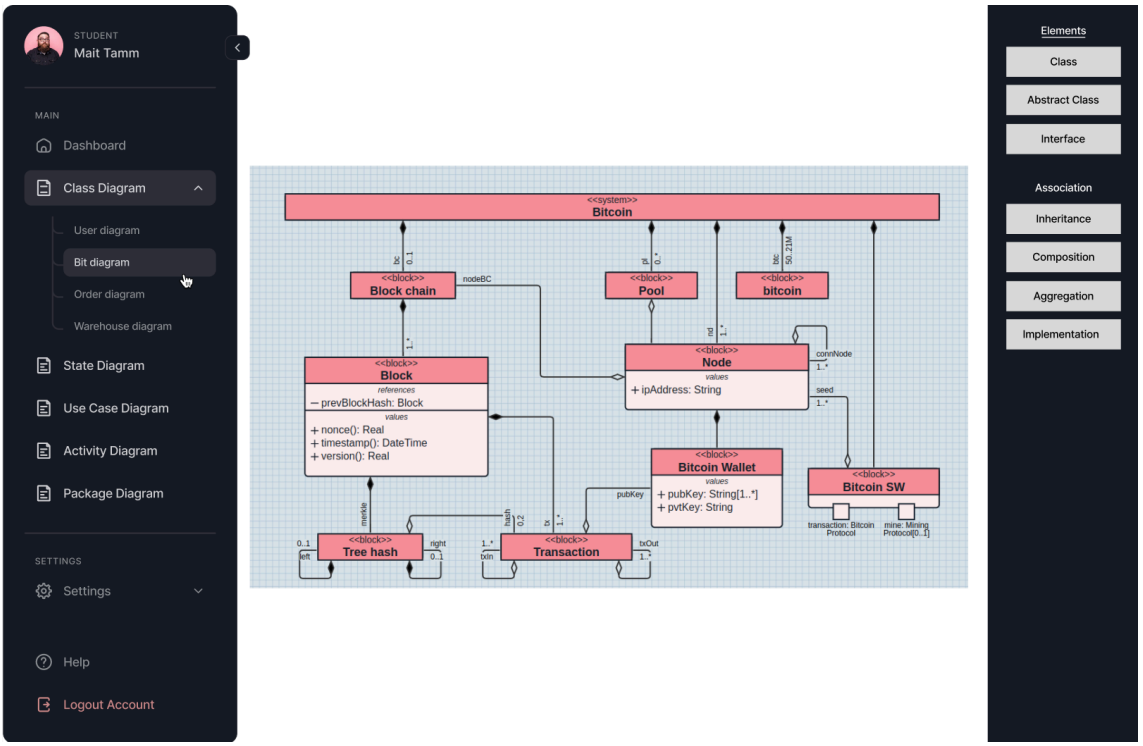

Figure 2. Class Diagram Task Solving Interface.

Appendix 3 – Database Structure

Figure 3. Task Collections Database Schema Designed in DataGrip.

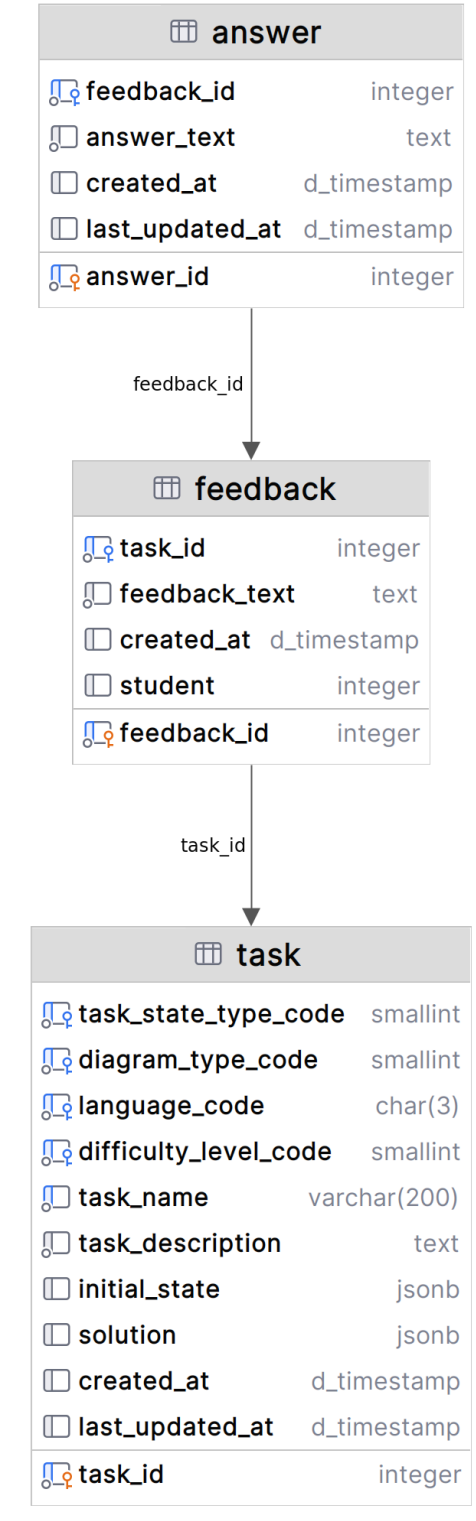

Figure 4. Feedback Database Schema Designed in DataGrip.

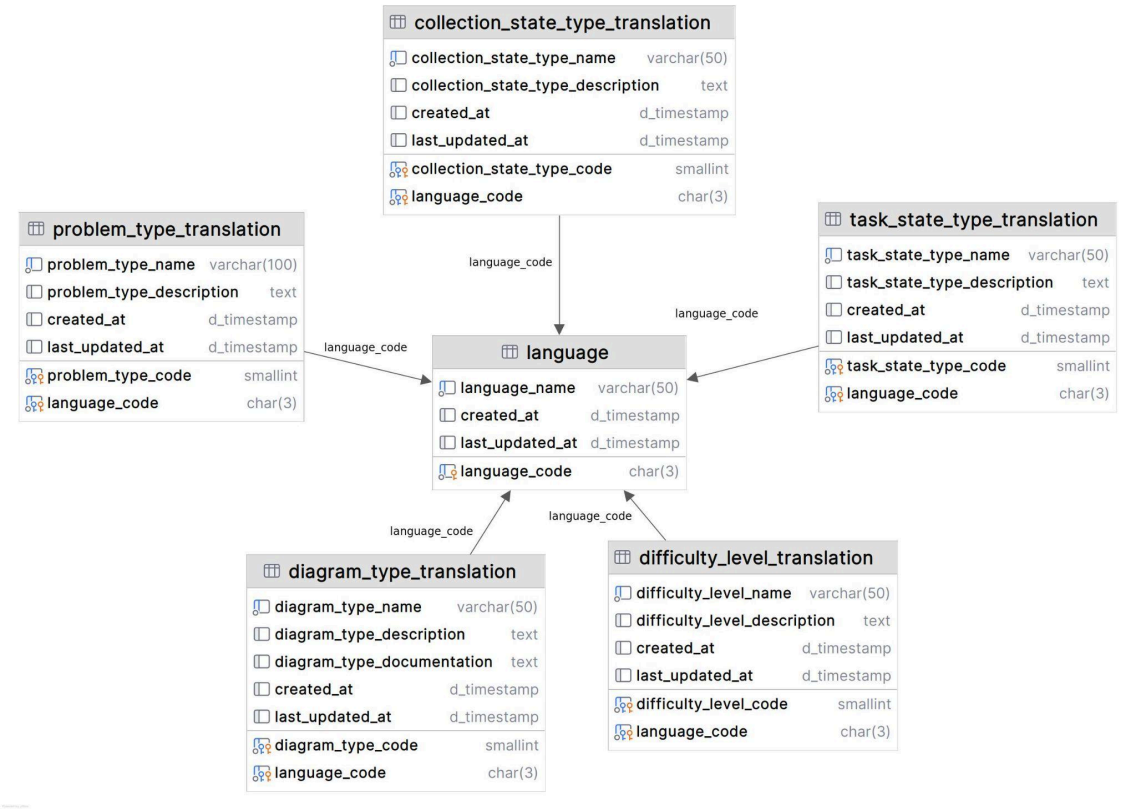

Figure 5. Language Classificators Database Schema Designed in DataGrip.

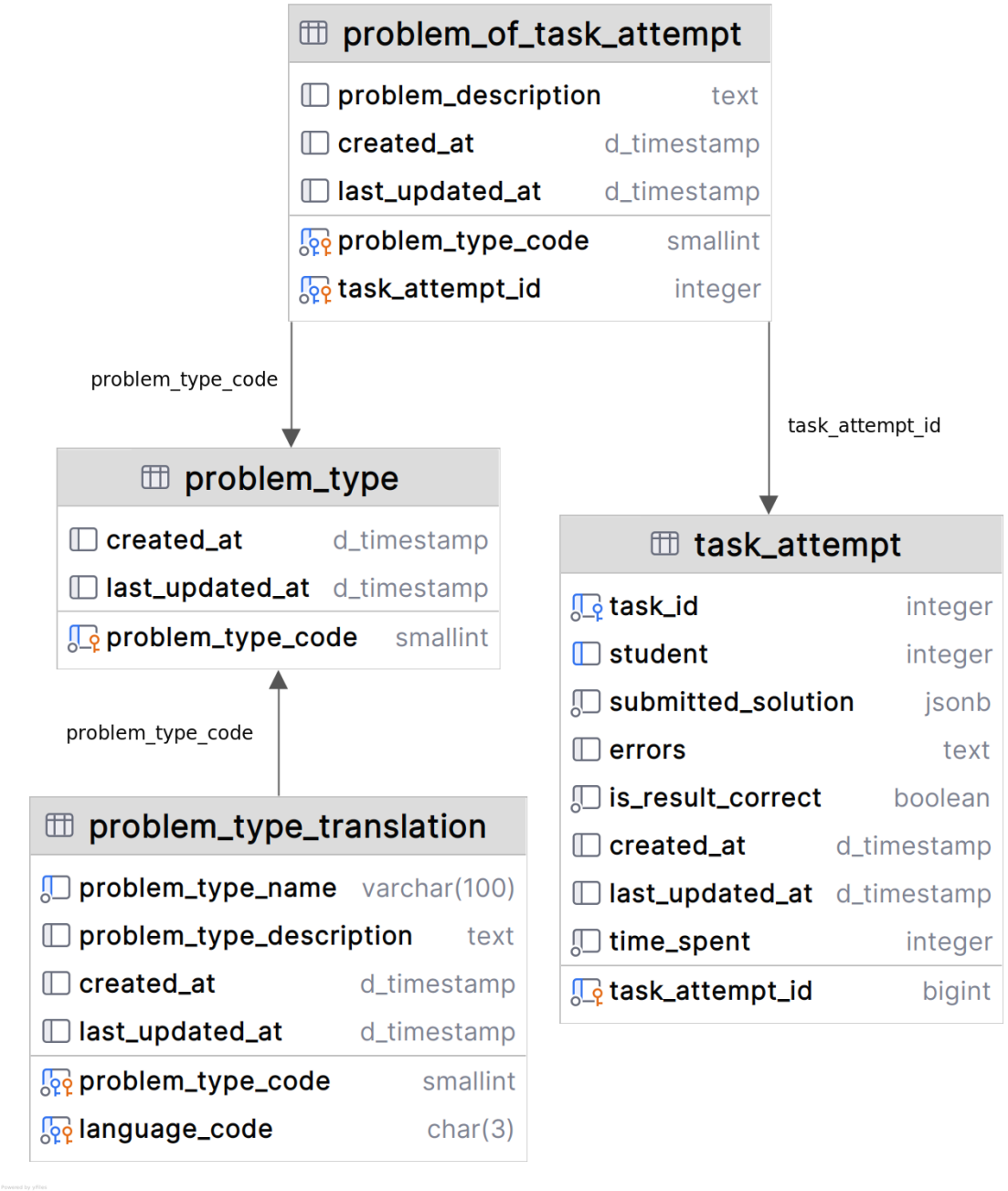

Figure 6. Problem of Task Attempts Database Schema Designed in DataGrip.

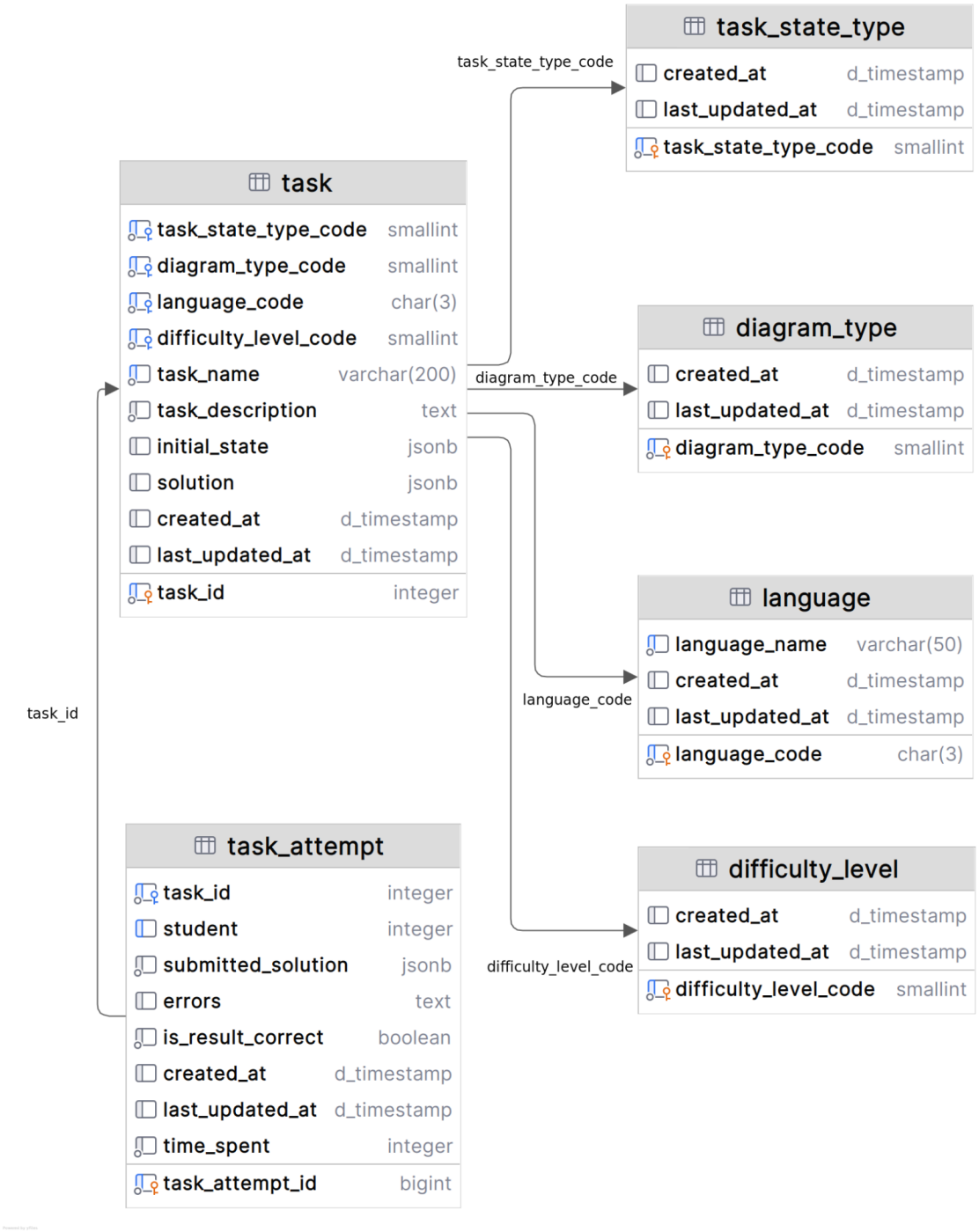

Figure 7. Task Database Schema Designed in DataGrip.

Appendix 4 – Kanban Board

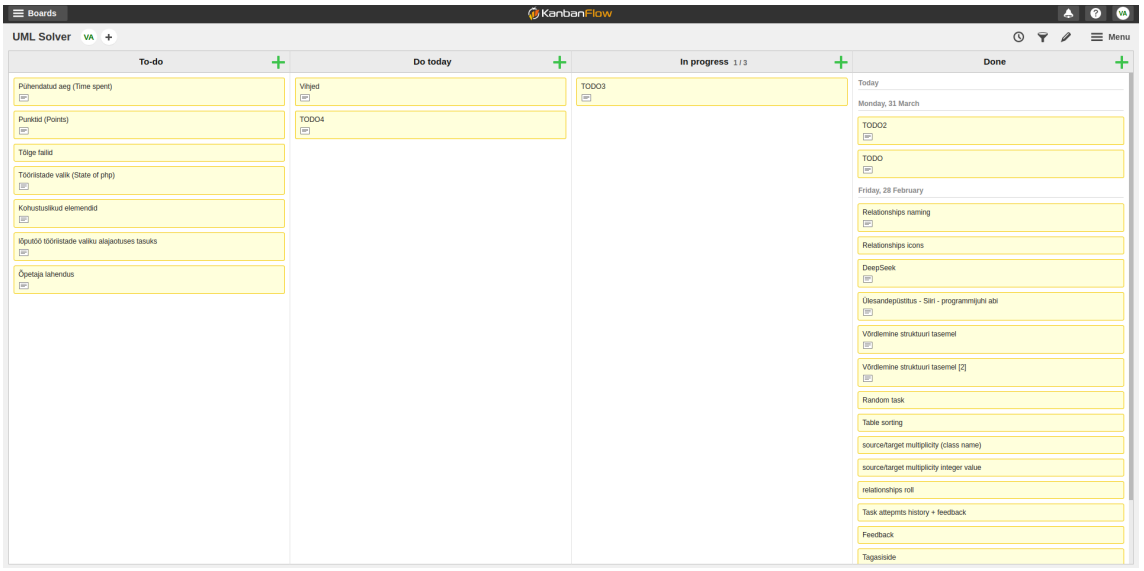

Figure 8. Kanban Board Displaying Task Management.

Appendix 5 – Existing Tools

Table 1. Comparative Overview of UML Tools (Part 1).

	Enterprise Architect 12	**StarUML**	**UMLet 15.1**
Main Use	Enterprise modeling, large systems	Education, software development	Education, sketches
Platform	Desktop (Windows)	Desktop (Windows, Linux, MasOS)	Web
Free Version	Limited	Limited	Yes
Paid Version	Yes	Yes	No
License Type	Commercial	Commercial	Open-source
Supported Languages	UML, BPMN, ArchiMate, SysML, etc.	UML 2.x	UML 2.x (limited subset)
Third-party Extensions	Yes (many available)	Yes	No
Built-in Extension Mechanism	Yes (scripting and API)	Yes (JavaScript-based plugins)	No
Model Validation Support	Yes (rules via scripting/plugins)	Yes	No
Validation Against Predefined Rules	Possible (via setup/extensions)	Limited	No
Model Comparison (Built-in)	Limited (requires extension)	No	No
Integrated AI Features	No	No	No

Table 2. Comparative Overview of UML Tools (Part 2).

Feature	yEd Live	Creately	Lucidchart
Main Use	General-purpose visualization	Collaborative modeling	Professional collaboration
Platform	Web	Web/Desktop	Web
Free Version	Limited	Limited	Limited
Paid Version	Yes	Yes	Yes
License Type	Freemium (limited functionality for free, premium for full access)	Freemium (limited functionality for free, premium for full access)	Freemium (limited functionality for free, premium for full access)
Supported Languages	Generic diagrams (not UML-focused)	UML, BPMN.	UML, flowcharts, BPMN.
Third-party Extensions	No	No	Yes (via integrations)
Built-in Extension Mechanism	No	No	No
Model Validation Support	No	Partial (consistency checks only)	Yes (AI-based suggestions only)
Validation Against Predefined Rules	No	No	No
Model Comparison (Built-in)	No	No	No
Integrated AI Features	No	No	Yes (auto suggestions, improvements, layout)

Table 3. Comparative Characteristics of UML Diagram Comparison Tools.

	EMF Compare	**Visual Paradigm**	**IBM RSA**	**Enterprise Architect**	**UMLDiff**	**DiffMerge**
Environment	Eclipse IDE only	Visual Paradigm environment	IBM Rational Software Architect	EA	Research. academic prototypes	Generic text diff tool
License	Free	Paid	Paid	Paid	Limited	Free
Visual UML Comparison	Yes	Yes	Yes	Limited (via Baseline)	Yes (prototype)	No (XMI-level only)
Structural Comparison	Yes	Yes	Yes	Yes	Yes	Partial
Semantic Comparison	Yes	Yes	Yes	Partial	Yes	No

Table 4. Comparison of Software Platforms for Code Practice and Learning.

	LeetCode	HackerRank	CodeSignal	CodeChef
Target Audience	Intermediate to advanced developers	Students, job seekers, educators	Job seekers, universities, employers	Beginner to competitive programmers
Task Types	Algorithms, data structures, coding interviews	Coding challenges, SQL, AI, databases	Coding tasks, assessments, games	Programming problems, contests
Feedback Style	Instant feedback with test case breakdowns	Immediate test results, detailed scoring	Score reports, real-time test validation	Basic validation and leaderboard feedback
Validation Approach	Automatic test execution and scoring	Auto-evaluation with custom test cases	Live execution with system tests	Test case execution and runtime results
Hint System	Premium users get hints and solutions	Built-in hints and solution discussions	Hints available in guided modes	No structured hints, discussion forums only
Progress Tracking	Progress dashboard and rankings	Skill score and certificate tracking	Personal dashboard and readiness scores	User profiles and contest history
Competitive Mode	Yes (Contests, Rankings, Leaderboards)	Yes (Contests, Hackathons)	Yes (Arcade, Coding Battles)	Yes (monthly contests, ranking system)
Free/Paid Access	Freemium (some tasks free, premium for full access)	Free for individuals, paid for employers	Freemium (pro features for companies)	Free (educational and open source model)
Integration Potential	Good (APIs, widely used in technical prep)	High (used in education and recruiting)	Moderate to High (enterprise & academia use)	Limited (mostly self-contained platform)

Appendix 6 – Three Tier Architecture

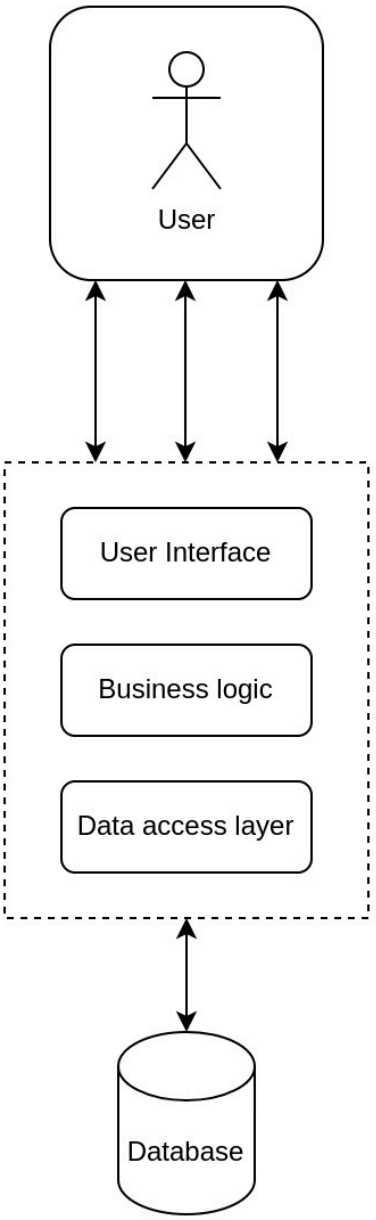

Figure 9. Three Tier Architecture Diagram.

Appendix 7 – REST API Endpoints List

```
AUTHENTICATION
├── POST /api/auth/login
└── POST /api/auth/lecturer/login

ANSWERS
├── GET    /api/answers
├── GET    /api/answers/{id}
├── POST   /api/answers/create
├── POST   /api/answers/update/{id}
└── DELETE /api/answers/delete/{id}

TASKS
├── GET    /api/tasks
├── GET    /api/tasks/{id}
├── GET    /api/tasks/class-diagram/{id}
├── GET    /api/tasks-meta-data/{id}
└── POST   /api/tasks/create

TASK ATTEMPTS
├── GET    /api/task-attempts
├── GET    /api/task-attempts-enriched
├── GET    /api/task-attempts/{id}
├── GET    /api/task-attempts-by-task/{task_id}
└── POST   /api/task-attempts/create

COLLECTIONS
└── GET /api/collections

FEEDBACK
└── GET /api/feedbacks

COMMON
├── GET /api/task-state-types
├── GET /api/difficulty-levels
├── GET /api/collection-state-types
├── GET /api/diagram-types
├── GET /api/problem-types
├── GET /api/statistics
└── POST /api/remove-history
```

Figure 10. REST API Endpoint List.

Appendix 8 – Web Pages URL List

```
GENERAL PAGES
├── /                    → Home Page
├── /login               → Student Login Page
├── /logout              → Logout Handler
├── /404                 → 404 Error Page
├── /about               → About Page

TASKS
├── /tasks                           → Task List Page
├── /tasks/class-diagram/{id}        → Class Diagram Solve Page
├── /tasks/package-diagram/{id}      → Package Diagram Solve Page

ATTEMPTS
├── /attempts            → Attempt History Page
├── /task-attempts/{id} → Task Attempt Details Page

STATISTICS
├── /statistics          → Statistics Page

FEEDBACK
├── /feedback            → Feedback Page

LECTURER PAGES
├── /lecturer/login              → Lecturer Login Page
├── /lecturer/class-diagram      → Lecturer Class Diagram Editor
├── /lecturer/package-diagram    → Lecturer Package Diagram Editor
└── /lecturer                    → Lecturer Default (Class Diagram Editor)
```

Figure 11. Web Pages URL List.

Appendix 9 – User Interface

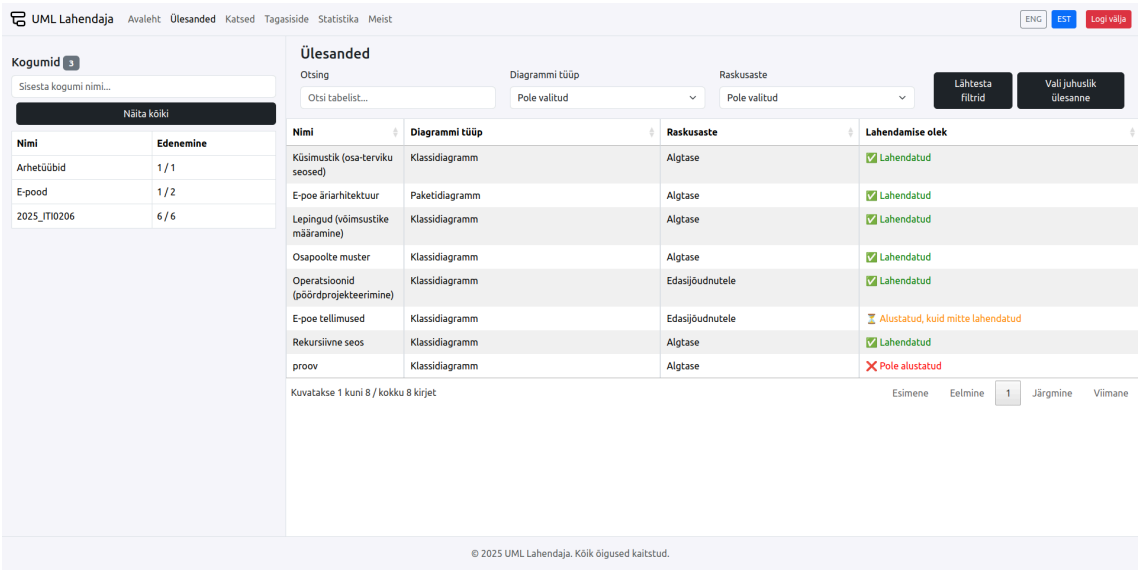

Figure 12. Task Overview Page.

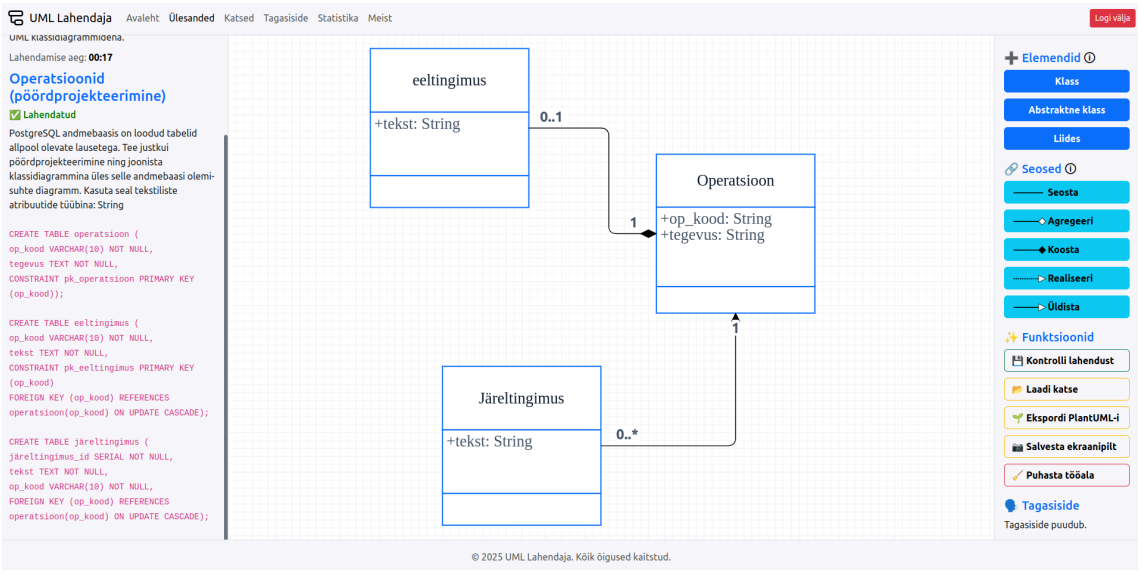

Figure 13. Solving a Class Diagram Task.

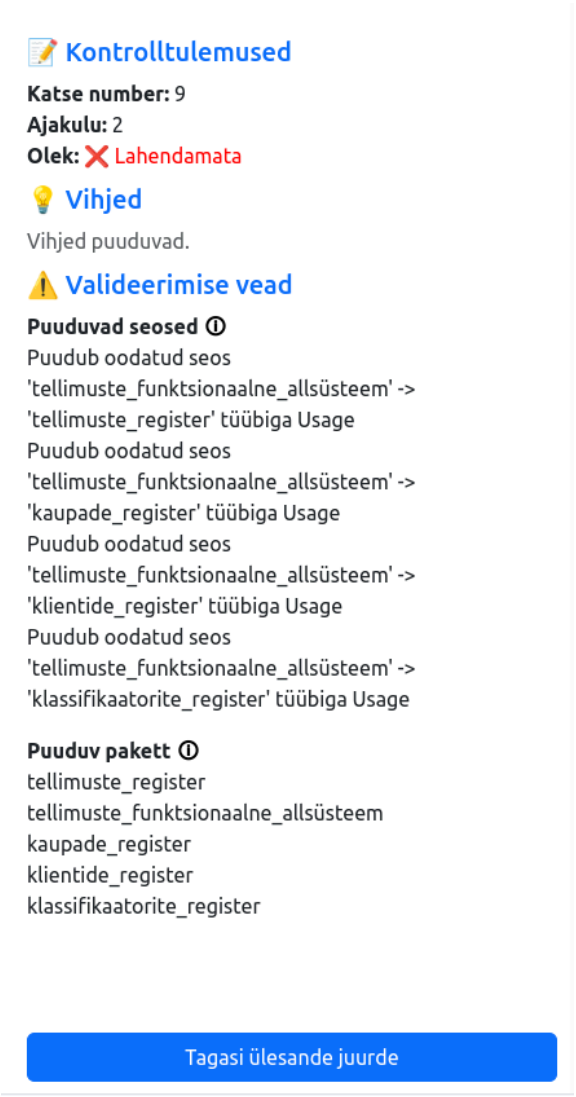

Figure 14. Validation Errors Sidebar View.

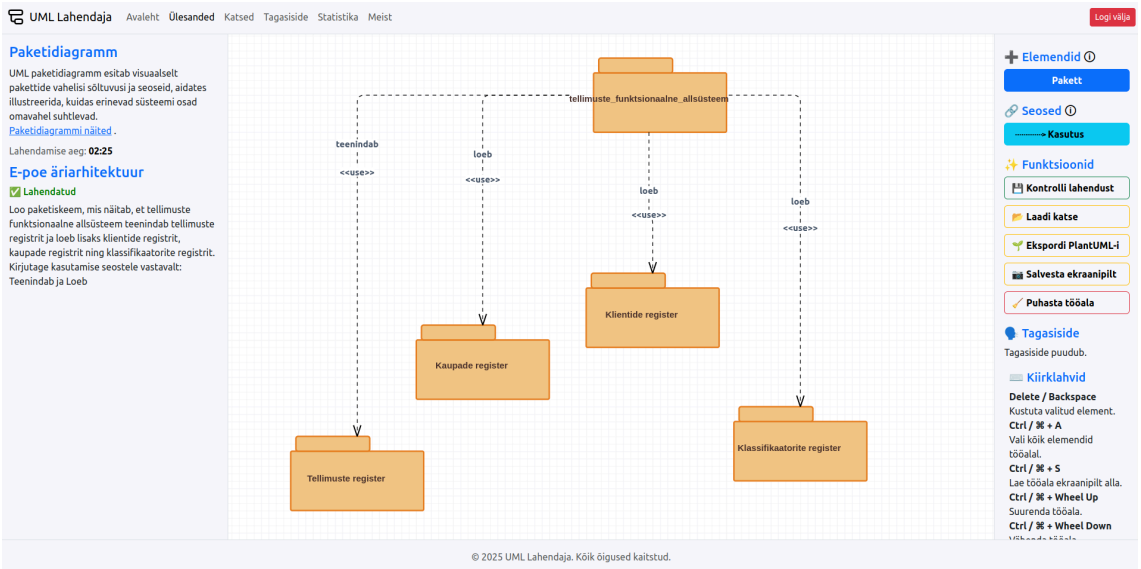

Figure 15. Solving a Package Diagram Task.

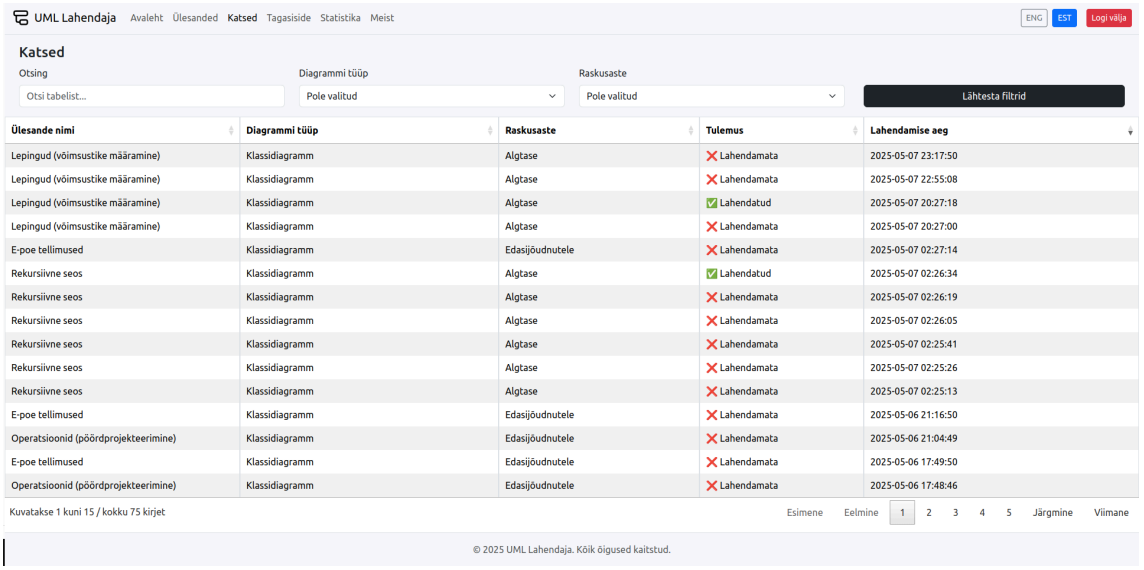

Figure 16. Attempt History View.

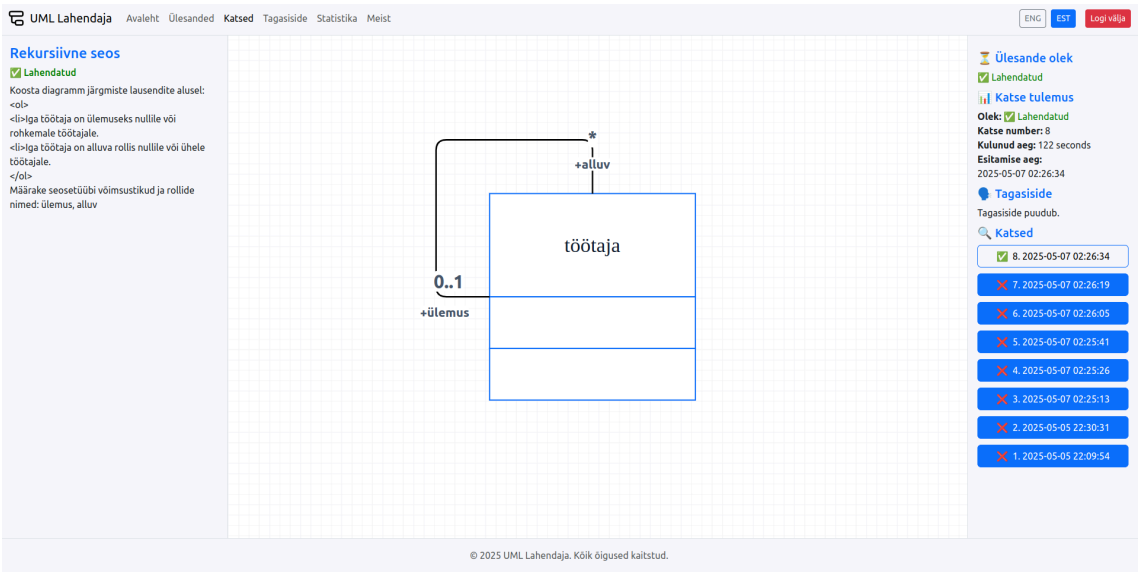

Figure 17. Detailed Task Attempt View.

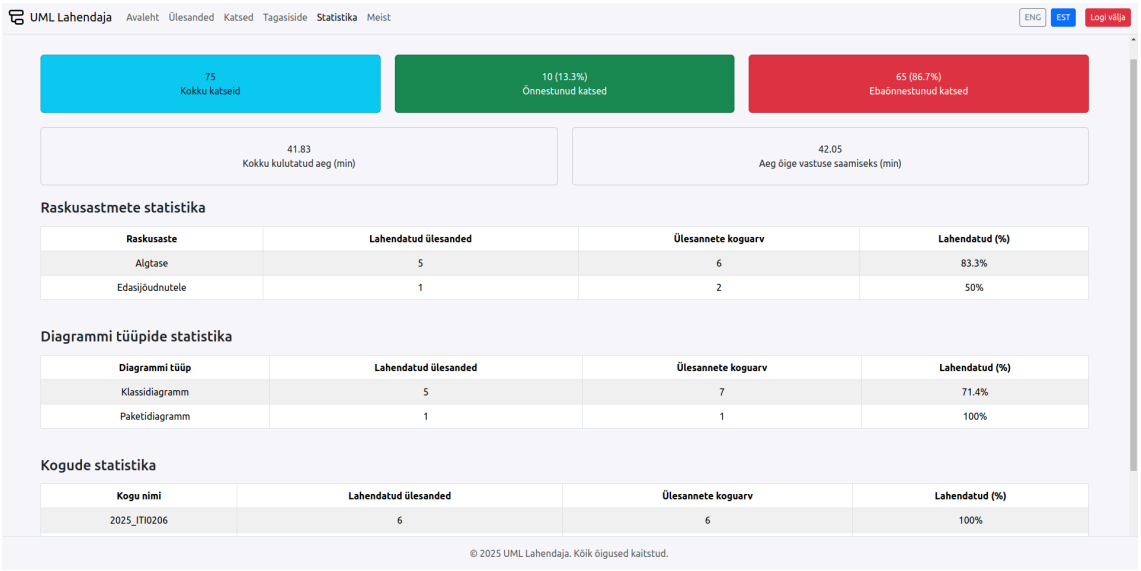

Figure 18. Statistics View.

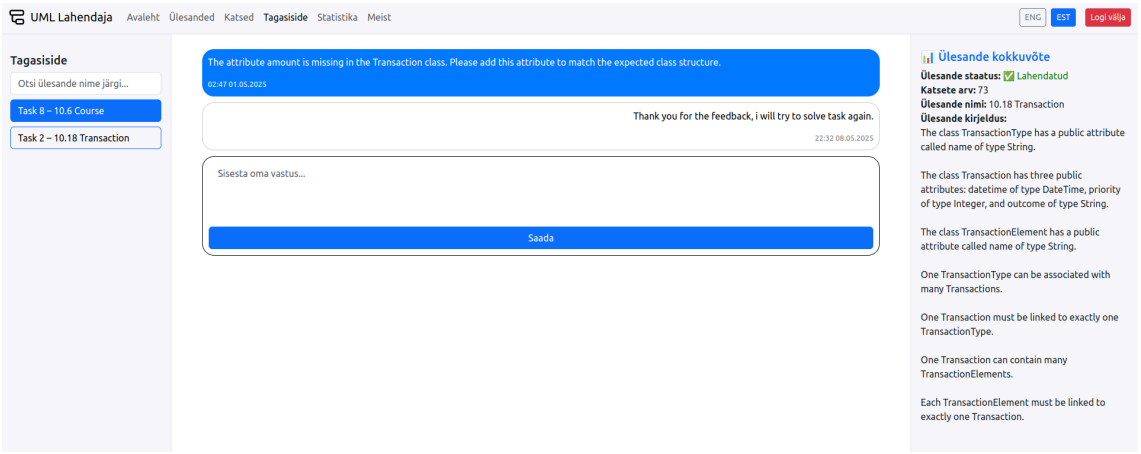

Figure 19. Feedback View.

Figure 20. Lecturer Class Diagram Sidebar View.

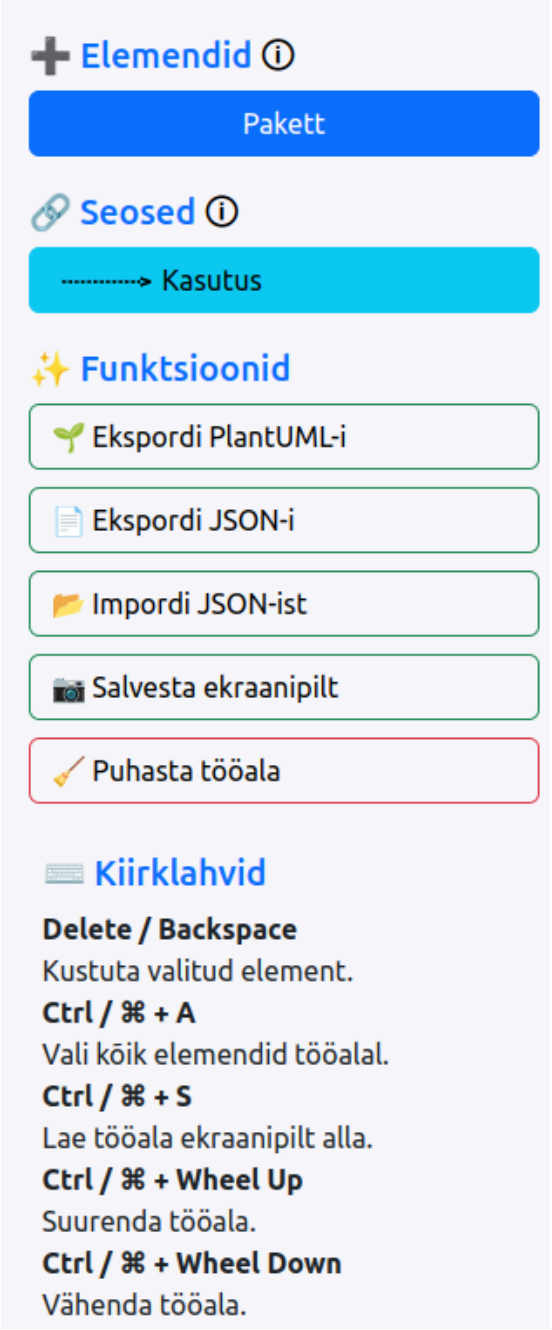

Figure 21. Lecturer Package Diagram Sidebar View.

Appendix 10 – Validation Errors

Table 5. UML Diagram Validation Error Types.

Error type	**Description**
Missing Classes	Classes that are expected but missing.
Extra Classes	Classes that are not expected but found.
Missing Package	A required package is missing.
Extra Package	An extra package is present.
Missing Relationships	Relationships that are expected but missing.
Extra Relationships	Relationships that are not expected but found.
Duplicate Elements	Duplicate elements are not allowed in the diagram.
Duplicate Relationships	Duplicate relationships are not allowed in the diagram.
Wrong Relationship Type	The relationship type is incorrect.
Wrong Multiplicity in Relationship	Multiplicity in the relationship is incorrect.
Missing Multiplicity in Relationship	Multiplicity is missing in the relationship.
Wrong Role in Relationship	The role in the relationship is incorrect.
Missing Role in Relationship	The role is missing in the relationship.
Wrong Name in Relationship	The name in the relationship is incorrect.
Missing Name in Relationship	The name is missing in the relationship.
Wrong Generalization Set	The generalization set label is incorrect.
Missing Generalization Set	Expected generalization set is missing.
Missing Constraints (Generalization Set)	Expected constraints are missing from the generalization set.

Wrong Constraints (Generalization Set)	Constraints of the generalization set do not match the expected ones.
Attribute Count Mismatch	Mismatch in the number of attributes.
Attribute Count Mismatch (Object)	Mismatch in attribute count on object level.
Missing Attribute	An expected attribute is missing from the class definition.
Multiplicity Format Error	The multiplicity format is incorrect or invalid.
Attribute Multiplicity Format Error	Multiplicity for attributes is incorrect or badly formatted.
Wrong Attribute Data Type	The data type of the attribute is incorrect.
Wrong Attribute Multiplicity	The multiplicity of the attribute is incorrect.
Missing Attribute Type	The data type of an attribute is missing.
Invalid Attribute Visibility	Attribute has an invalid or missing visibility modifier.
Method Count Mismatch	Mismatch in the number of methods.
Method Count Mismatch (Object)	Mismatch in method count on object level.
Missing Method	An expected method is missing from the object.
Missing Method Type	The return type of a method is missing.
Wrong Method Parameters	Mismatch in method parameter list.
Invalid Method Visibility	Method has an invalid or missing visibility modifier.

Appendix 11 – Task Example

Koosta diagramm järgmiste lausendite alusel:

1. Klassil Osapool on atribuut e_meil, mis on tüüpi String.
2. Klassil Isik on valikuline atribuut eesnimi, mis on tüüpi String.
3. Klassil Isik on valikuline atribuut perenimi, mis on tüüpi String.
4. Klassil Organisatsioon on atribuut ärinimi, mis on tüüpi String.
5. Klassid Isik ja Organisatsioon on klassi Osapool alamklassid ning nende üldistusseosed moodustavad koos üldistuste hulga nimega "Osapoole_liik". Kehtib piirang, et iga osapool on kas isik või organisatsioon, aga mitte mõlemat korraga (üldistuste hulga piirang)

Figure 22. Example of Task Description.

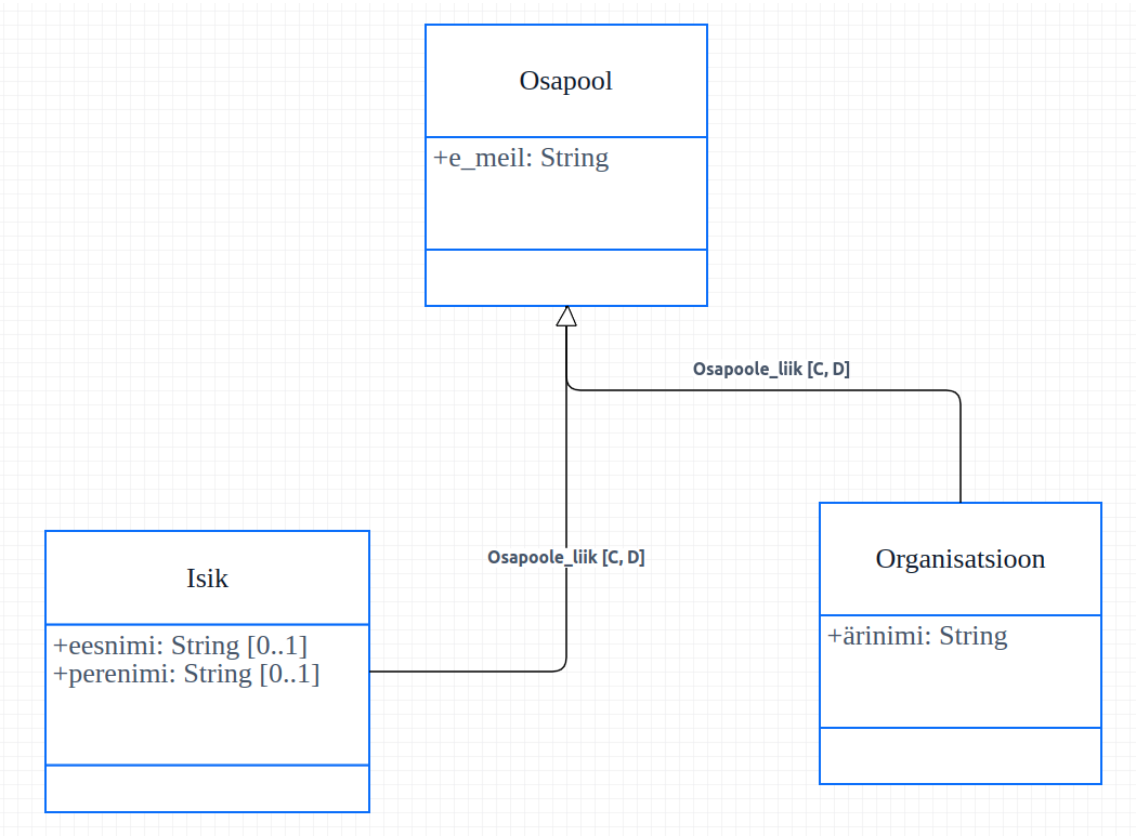

Figure 23. Example of Constructed Reference Diagram.

Appendix 12 – Students Testing Survey

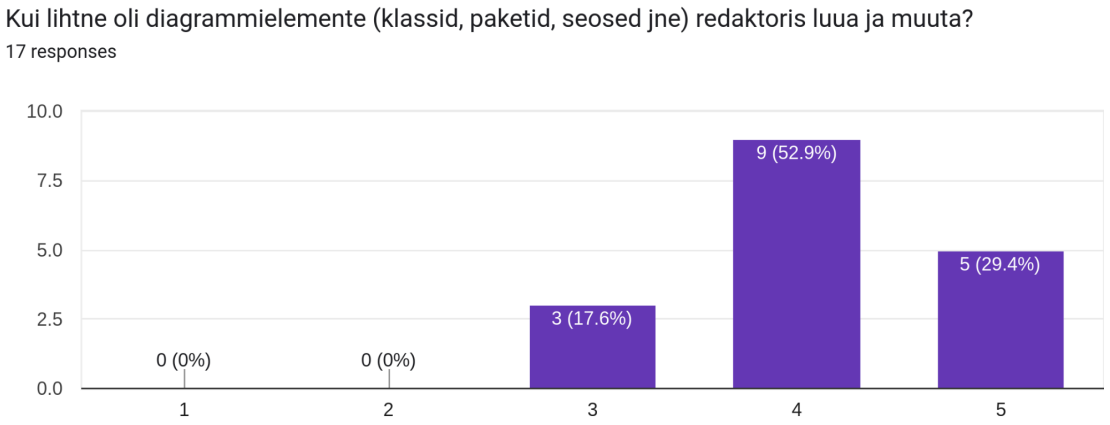

Figure 24. Ease of Creating and Modifying Diagram Elements.

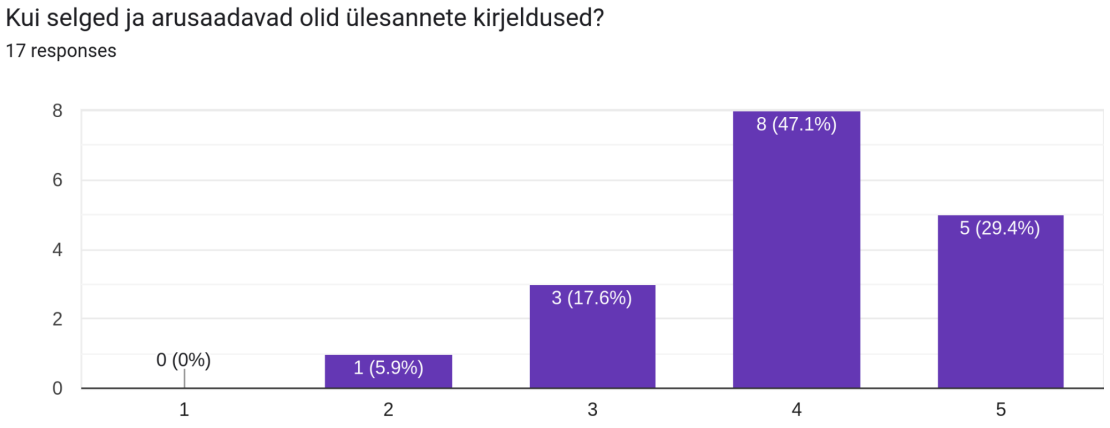

Figure 25. Clarity of Task Descriptions.

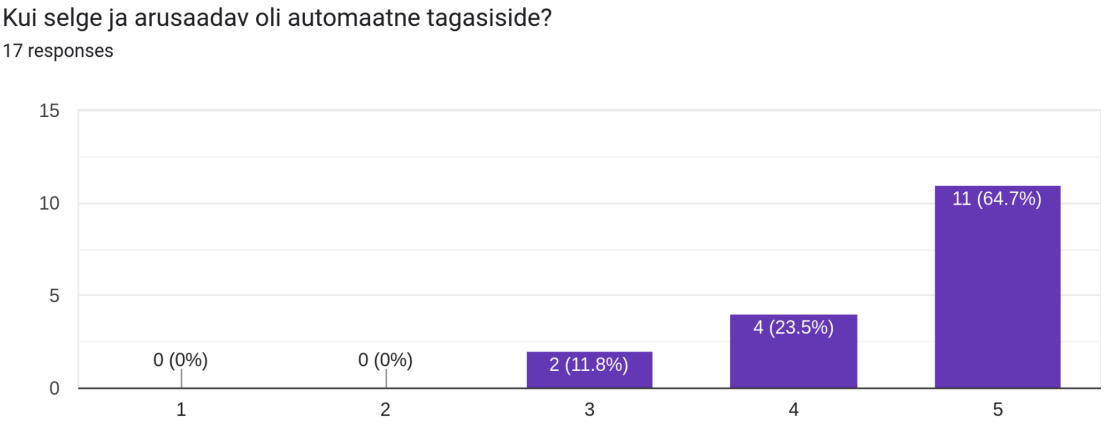

Figure 26. Clarity of Automatic Feedback.

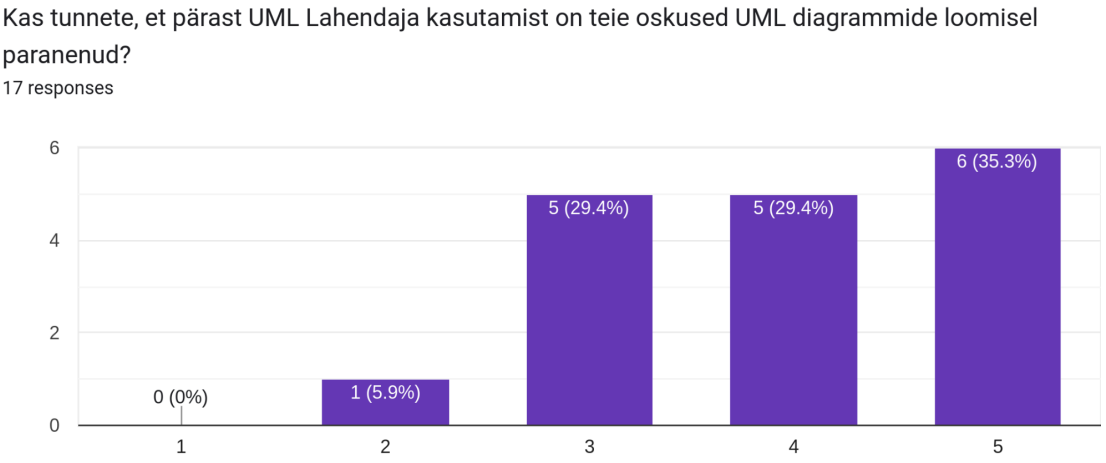

Figure 27. Perceived Improvement in UML Skills.

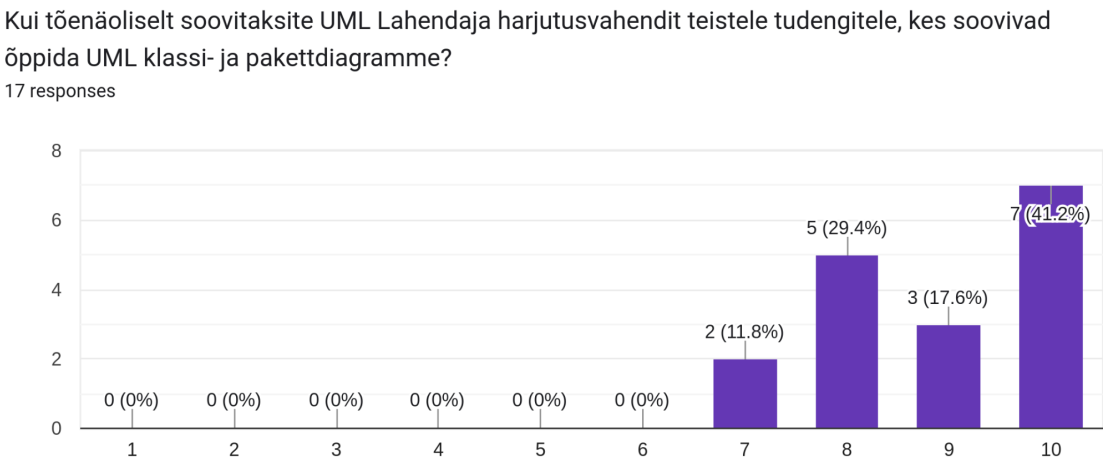

Figure 28. Likelihood of Recommending UML Solver.

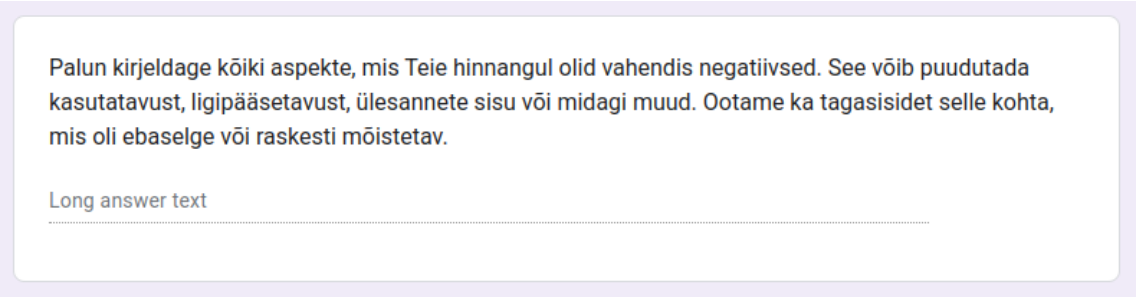

Figure 29. Negative Feedback Question.

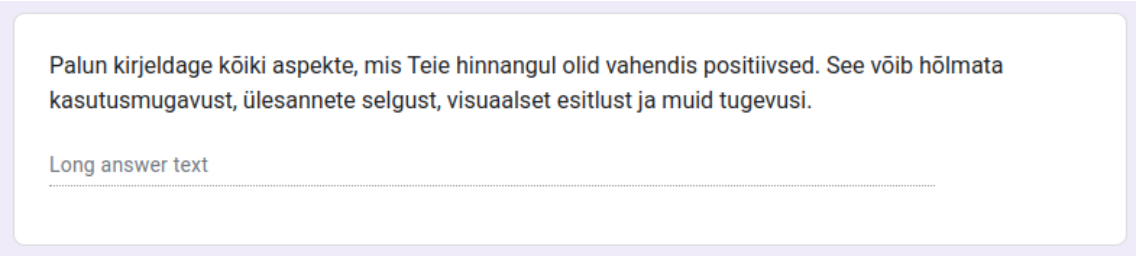

Figure 30. Positive Feedback Question.