Technical University of Munich

School of Computation, Information and Technology

-- Informatics --

Bachelor's Thesis in Informatics: Games Engineering

Scorpio: A Visual Studio Code Extension for Interactive Learning Platforms

Dennis Jandow

Technical University of Munich

School of Computation, Information and Technology
-- Informatics --

Bachelor's Thesis in Informatics: Games Engineering

Scorpio: A Visual Studio Code Extension for Interactive Learning Platforms

Scorpio: Eine Visual Studio Code-Erweiterung für interaktive Lernplattformen

Author:	Dennis Jandow
Supervisor:	Prof. Dr. Stephan Krusche
Start Date:	01.07.2024
Submission Date:	01.11.2024

I confirm that this Bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 01.11.2024 Dennis Jandow

Transparency in the use of AI tools

In preparing this thesis, I utilized ChatGPT[1] for final revisions, helping to enhance clarity, coherence, and academic tone across all chapters. Additionally, I used Grammarly[2] to correct typos, punctuation, and grammar throughout the document, ensuring precision and readability. GitHub Copilot[3] was used to generate code snippets for the developed functionality, streamlining the coding process and providing valuable suggestions during development. I have manually checked all the content created by AI tools and verified its correctness.

[1] https://chatgpt.com/
[2] https://app.grammarly.com/
[3] https://github.com/features/copilot

Acknowledgements

This Thesis has been a journey - full of ups and downs, marked by moments of intense focus, unexpected challenges and even more fulfilling breakthroughs. Nevertheless this journey was worth every second, and I am grateful to those who have support me along the way.

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Dr. Stefan Krusche, for giving me the opportunity to work on this thesis. His guidance and the discussions we shared provided new perspectives and valuable insights.

I also would like to thank the members of the `Applied Education Technology` chair for their expertise and advice, and sometimes even just playing the rubber duck for me. This applies in particular to Matthias Linhuber, Ramona Beinstingel, Maximilian Anzinger, Patrick Bassner, and Benedikt Geisberger.

A special thanks goes to Yannik Schmidt, who worked on the closely related topic about the online IDE. Our collaboration and exchange of ideas guided this thesis, and I am grateful for the strong relationship we developed.

On a more personal note, I want to thank my friends, who were understanding, everytime I had to dodge them because of this thesis. And also thanks for everytime they were my way of procrastinating.

Lastly, I extend my deepest gratitude to my parents and my brother, who have supported me unwaveringly throughout this journey. Though they may not have been able to help with the technical aspects of this thesis, their encouragement and readiness to assist have been invaluable to me.

To all of you, thank you for being part of this journey.

Abstract

Programming exercises are an essential part of computer science education as they offer students practical application to theoretical concepts. Computer science students typically solve these exercises in integrated development environments (IDEs). However, instructional materials for these exercises are often provided in a separate document or on a dedicated website. This separation can lead to a cognitive and coordinative overload for students, as they have to switch between the IDE and the application with instructions.

This thesis proposes an integrated Visual Studio Code Extension for the online learning platform Artemis to seamlessly incorporate the entire programming exercise life cycle directly within the student's IDE. This integration aims to enhance the overall usability and improve the learning experience for students by streamlining the workflow and reducing unnecessary context switches.

Zusammenfassung

Programmieraufgaben sind ein wesentlicher Bestandteil des Informatik Studiums, da sie den Studierenden ermöglichen, theoretische Konzepte in der Praxis anzuwenden. Studierende lösen diese Aufgaben typischerweise in integrierten Entwicklungsumgebungen (IDEs). Allerdings werden die Anweisungen zu den Aufgaben meist in Dokumenten oder auf separaten Websites bereitgestellt. Diese Trennung kann zu einer kognitiven und koordinativen Überlastung führen, da die Studierenden ständig zwischen der IDE und der Anwendung mit den Anweisungen wechseln müssen.

Diese Arbeit schlägt ein integriertes Visual Studio Code Plugin für die Online-Lernplattform Artemis vor, um den gesamten Lebenszyklus einer Programmieraufgabe direkt in die IDE der Studierenden zu integrieren. Ziel dieser Integration ist es, die Benutzerfreundlichkeit zu verbessern und das Lernerlebnis für Studierende zu optimieren, indem der Arbeitsablauf vereinfacht wird und unnötige Kontextwechsel reduziert werden.

Contents

1 Introduction

Universities make use of a variety of online learning platforms. These platforms allow the students to learn at their own pace and often provide immediate feedback on their learning progress. One of those platforms at the Technical University of Munich is Artemis[4]. Computer science students use it throughout their studies in various courses and exercises. The core functionality of Artemis includes the management of programming exercises and the automatic assessment of the student's submissions [KS18]. In recent updates, it was extended by functionalities such as an exam mode [Tur20], a chat for course internal communication [Kel23, Sch21], and an integrated chatbot answering questions about exercises [BFK24].

Artemis displays the instructions for programming exercises on its website, alongside a button to clone the exercise repository. Students typically clone the repository into their local integrated development environment (IDE), such as Eclipse[5], IntelliJ IDEA[6], or Visual Studio Code[7] (VSCode). In the IDE, the students solve the programming exercise and submit the solution back to the platform via `git`[8]. The platform then automatically assesses the submission and provides feedback to the students on the website.

1.1 Problem

The students commonly tackle programming exercises using IDEs because they support writing, debugging, and testing code [DL12], whereas an external platform or a website provides the instructions for the exercises. In the case of Artemis, it presents the instructions in the browser on its website. While students solve the exercise, they must alternate between the browser window, which displays the instructions, and the IDE, where they work on their submission. This switching back and forth between the two applications will occur multiple times until the student submits their work. After each submission the website displays feedback

[4]https://artemis.cit.tum.de
[5]https://eclipseide.org/
[6]https://www.jetbrains.com/idea/
[7]https://code.visualstudio.com/
[8]https://git-scm.com/

on the student's solution. The student has to switch back to the website again to see the feedback and then return to the IDE to improve their submission. As a result, the student has to continuously switch between the IDE and the browser throughout the exercise's lifecycle until the exercise is finally complete.

The problem is not limited to using a local IDE but also affects the currently developed online IDE Theia for Artemis [Sch24]. Here, the problem requires students to switch between the browser tab with the Artemis client containing the instructions and the submission feedback, and the browser tab for the online IDE. This issue is particularly significant for non-computer science students and novice programmers, the primary target group of the online IDE. These students need a more user-friendly workflow that provides an easy entry point.

1.2 Motivation

The separation of instructions and coding environment in Artemis (Figure 1) leads to repetitive context switches and disrupts the exercise experience. This separation is not only inefficient but also hinders the learning process of the students in three critical aspects:

1. Extraneous cognitive load

Extraneous cognitive refers to the mental effort required to process information. This load is influenced by the instructional procedures and the way they are presented. Ideally, this load should be minimized to free up cognitive resources for learning and problem-solving [SAK11].

However in Artemis, the separation of instructions and coding environment increases the extraneous cognitive load for students. Students must process unnecessary information by shift their attention back and forth between the IDE and the browser. This includes understanding which windows are open, locating the instructions, locating the code, and navigating between them. This additional cognitive load decreases the learning efficiency.

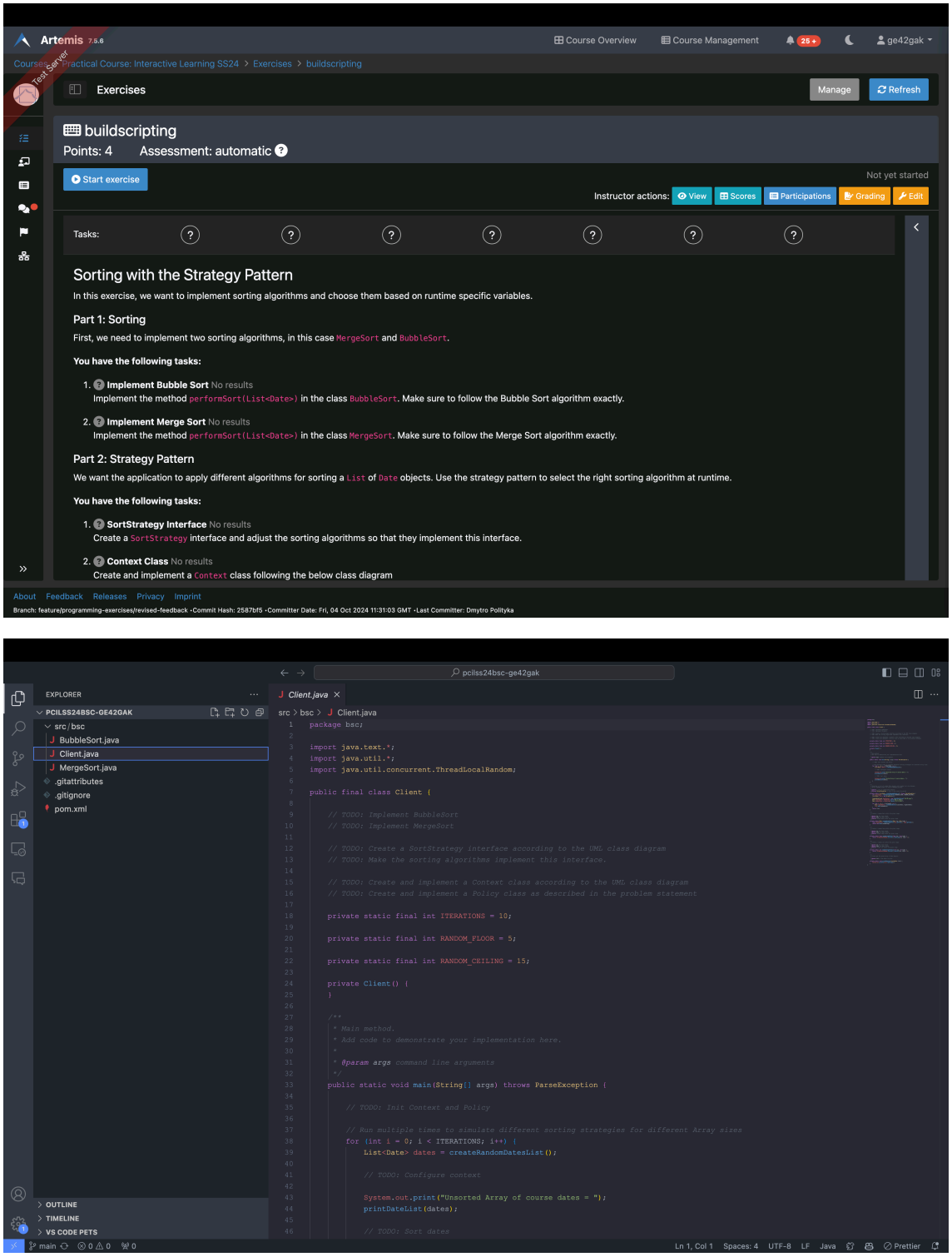

Figure 1: **Separation of Instructions and Coding Environment in Artemis.** The instructions for the exercise are displayed in the browser (top), while the student works on the exercise in the IDE (bottom). The student must switch between the two applications to complete the exercise.

2. Loss of focus

Each context switch increases the risk of losing focus, leading to distractions that hinder the exercise workflow. Research indicates that frequent context switches can lead to distractions and result in up to 50% of activies being unrelated to study tasks, ultimately making learning less effective [Jud15]. In Artemis, the need to constantly switch between the IDE and the browser to read instructions increases the likelihood of these focus breaks. Every switch requires students to disengage from the coding environment, significantly increasing the risk of losing focus. This fragmentation of attention impairs the learning experience, often leading to extended completion times.

3. Loss of time

Switching between different software applications consumes valuable time that the student could otherwise spend on solving the exercise. Time spent on managing context switches is a non-productive cost that directly impacts learning efficiency. To mitigate this effect, software tools should be designed to minimize context switches by combining tools into a single environment [WKM22].

The integration of the exercise instructions directly into the IDE addresses these problems directly. The students can focus on the task at hand without the need to switch between the IDE and the instructions. This minimizes the disruptions between reading the instructions and working on their solution. As a result the students can maintain their concentration and experience a smoother and more immersive exercise process.

Another significant advantage of integrating instructions within the IDE is the better mapping of code and feedback. Currently, when students submit their programming assignments, the feedback they receive is separated into the browser and hard to map to their code. With the integration of feedback and code within a single environment, students can better match submission feedback to specific parts of their code. This streamlined feedback loop ensures that students can

quickly identify and address mistakes, leading to faster iterations and deeper learning.

All the points mentioned above aim to improve the learning experience and usability for students by enhancing the mapping of instructions to code and reducing the context switches.

1.3 Objectives

In order to ease the process of programming exercises for students, this thesis aims to develop the VSCode Extension `Scorpio` for Artemis. We define the following objectives to achieve this higher-level goal:

Display of Problem Statement: The primary objective of this thesis is to develop an extension for Visual Studio Code that displays the Problem Statement of Artemis exercises within the IDE.

Control Exercise Lifecycle: A key objective of the integration is to incorporate the whole student facing exercise lifecycle within the IDE. The student should be able to authenticate themselves, start and work on exercises, as well as submit their solution and read the submission feedback. The extension should provide a seamless experience for the students by guiding them through the exercise process.

Integrate in Online IDE: The extension should be compatible with the Artemis Online IDE, enabling students to work on exercises without the need to switch between different browser tabs. This integration should provide a user-friendly workflow for students, especially for beginners.

Measure User Satisfaction: The extension must be able to evaluate the student's satisfaction by providing a survey about the exercise. The student's feedback should be stored on a server and be accessible to the course instructor.

1.4 Outline

The remainder of this thesis is structured as the following. Chapter 2 presents related work regarding the integration of Artemis exercise instructions into IDEs and the implementation of the Artemis Online IDE. Chapter 3 introduces the problem domain of the extension and explains the requirements. Chapter 4 describes the solution domain and the architecture of the extension. Finally, Chapter 6 concludes the thesis and provides an outlook on future work.

2 Related Work

This chapter will look at related work regarding the development of IDE plugins for online learning platforms. First, we will examine the IntelliJ plugin Orion, which was developed for the Artemis platform, and discuss the need for the new VSCode Extension for Artemis. Next, we will look at the currently developed Online IDE for Artemis, Theia, as the VSCode Extension should be integrated into it.

2.1 Orion

The plugin Orion[9] already explored the implementation of displaying exercise instructions and integrating the participation process in the IDE IntelliJ by JetBrains [Dun21, Gra23, Ung20]. IntelliJ primarily serves as a local IDE for Java Development, but Artemis supports a wider range of programming languages. As a result, a plugin for a broader target group is necessary. The majority of software developers mainly use the IDE Visual Studio Code[10], which offers a wide range of programming languages. Hence, the development of a new plugin supporting Visual Studio Code and the online IDE Theia is necessary.

2.2 Theia Online IDE

The Artemis Platform provides an online code editor to offer a low-barrier entry point for students participating in programming exercises. However, the current online editor lacks advanced features, such as code completion, syntax highlighting, and debugging capabilities. To enhance the experience, particularly for beginner users, Artemis is in the process of developing an online IDE based on the Theia IDE platform [Sch24]. Theia is an open-source cloud and desktop IDE platform[11], built on top of the Monaco Editor, the same code editor that powers VSCode. As a result, Theia provides a development experience similar to VSCode in both appearance and functionality.

[9] https://github.com/ls1intum/Orion

[10] https://survey.stackoverflow.co/2024/technology#most-popular-technologies-new-collab-tools

[11] https://theia-ide.org

The online IDE faces the same problem of repetitive context switches as the local IDE. To reduce these frequent switches between different browser tabs, the exercise instructions should be integrated directly into the online IDE. Theia offers three different types of plugins with which this integration can be achieved:

- **VSCode Extensions**: These plugins are simple to write, installable at runtime, and compatible with both Theia and VS Code. They are limited to the VS Code Extension API, which may restrict the use of some Theia native features.
- **Theia Extensions**: These are installed at compile time and grant full access to Theia's internal APIs. Theia Extensions are suitable for building highly customized features not covered by the VS Code Extension API.
- **Theia Plugins**: They are similar to VS Code Extensions but offer additional access to Theia-specific APIs and frontend components. However, Theia-specific features of these plugins are not compatible with VS Code.

Given that Artemis faces the problem of repetitive context switches across both local IDEs and online IDEs, it is essential to choose a plugin that can be integrated into both environments. VSCode Extensions offer the most unified and suitable solution, as they can be used in both Theia and VSCode.

3 Requirements Analysis

This section analyzes the use case for the Visual Studio Code Extension `Scorpio` within the Artemis system. The analysis begins by examining the general lifecycle of an exercise and the role of exercises in the Artemis ecosystem. We then present two scenarios to understand the student's interaction with the extension. From this, we derive both functional and non-functional requirements for the extension.

3.1 Exercise Lifecycle

The extension should allow users to participate and complete an exercise without leaving the IDE. Therefore, it is essential to understand the exercise process from a student's perspective. Figure 2 illustrates the lifecycle of an exercise from the student's viewpoint.

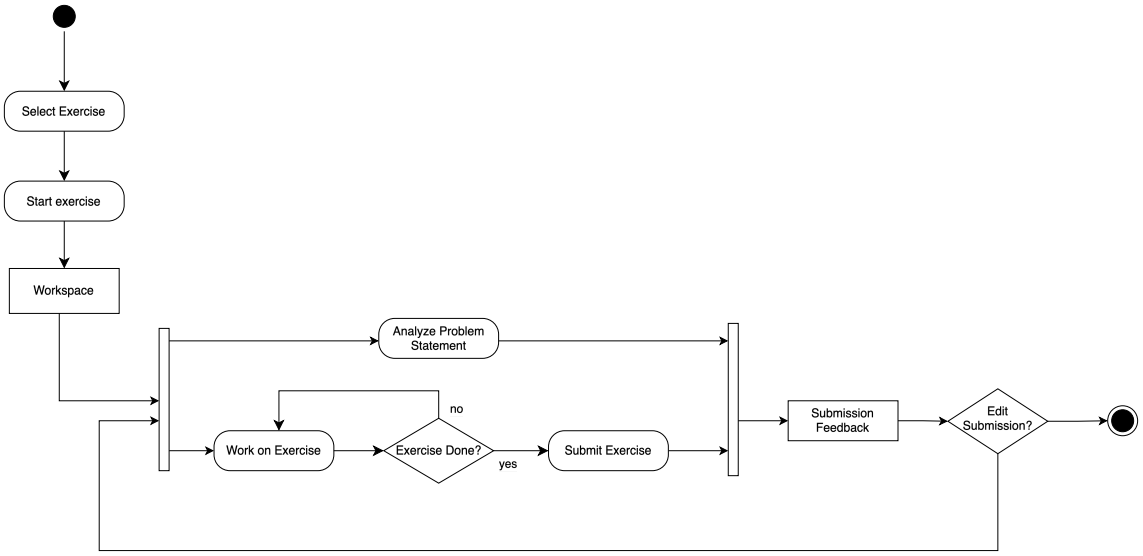

Figure 2: **Activity Diagram of the Exercise Lifecycle.** The diagram illustrates the process of completing an exercise from a student's perspective. The student selects an exercise, starts the exercise and works on the exercise while understanding the problem statement. After concluding, they submit the exercise, and receive feedback.

The exercise lifecycle begins with the student selecting an exercise from the list of exercises they are eligible to access. After selecting the exercise, they can start the exercise through a button, which triggers the creation of the student's repository in the background. The student can then clone this repository into their local

workspace within the IDE. At this point, the student can start working on the exercise. During this phase, they must read the problem statement to understand the task and associated requirements.

After completing the exercise, the student can submit their work. Artemis processes the submission, and the submission feedback is displayed to the student. Based on this feedback, the student can decide whether they reiterate the exercise and improve their submission or if they are satisfied with the result.

3.2 Analysis Object Model

Based on the exercise lifecycle, we can identify the core entities and their relationship. The Analysis Object Model (AOM) in Figure 3 illustrates these entities and their relationships in the Artemis ecosystem.

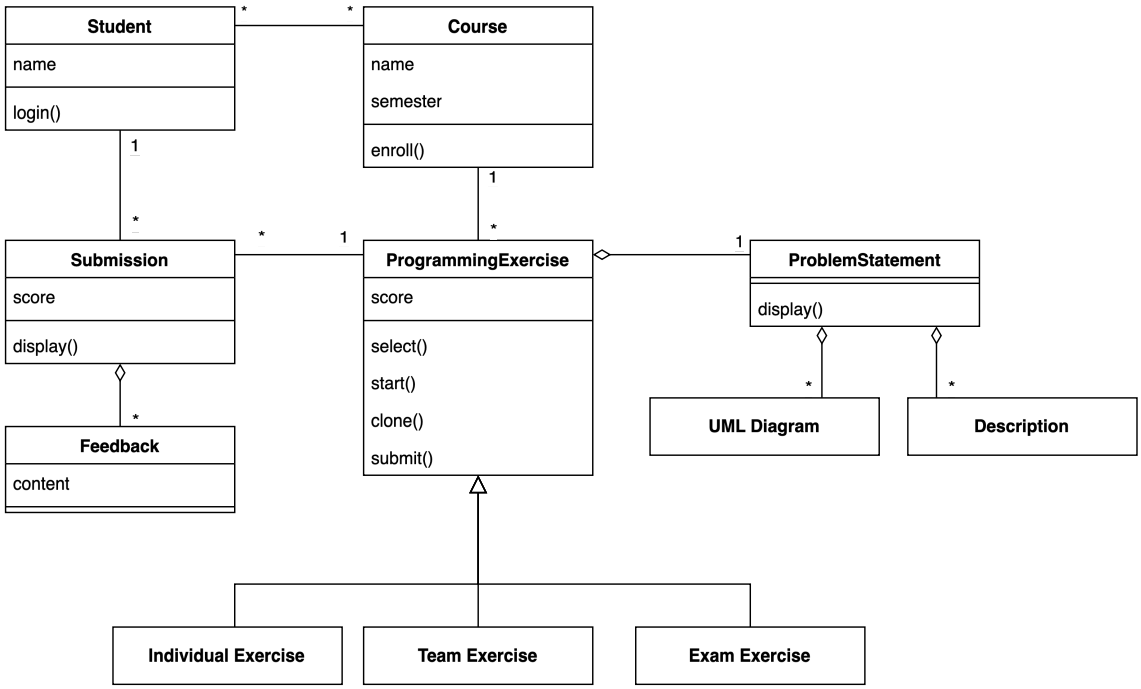

Figure 3: **Analysis Object Model of Artemis Exercises.** Students can enroll in multiple courses and access the exercises of these courses. Each exercise has a problem statement, which includes a textual description and UML diagrams. The student can submit their solution to an exercise, which is then evaluated by Artemis.

The primary actor in our system is the student, who can enroll in multiple `Courses`. Each course consists of multiple exercises. For our use case only the `Programming`

`Exercises` are relevant. These programming exercise are differentiated in "modes" based on the context they are worked on. This mode can either be:

- `Individual Exercises` are completed independently by students, typically as in-class assignments or homework.
- `Team Exercises` are often used for group projects to achieve bonus points.
- `Exam Exercises` are assigned during examinations and designed to be similar to individual exercises but with additional constraint related to exam situations.

Regardless of the exercise mode, each exercise has a problem statement, which provides students with the instructions needed to complete the task. The problem statement consists of a `Description` (textual guidance and tasks) and may also inlcude `UML Diagrams` to visually present structure and relationships of the task. Once the student has completed the exercise, they can submit their solution to Artemis. Artemis evaluates the `Submission`, provides a score, and generates `Feedback` for the student per submission.

3.3 Scenarios

Now that we have outlined the general workflow of an exercise and understand the role of an exercise in Artemis, we will explore two explicit scenarios that describe typical student interactions with the Artemis extension. We focus on two different scenarios: Scenario 1 describes the interaction of an inexperienced student working on an exercise in the online IDE, Scenario 2 describes the interaction of a student in an exam situation.

Scenario 1: Inexperienced Student using the Online IDE Justus is a first-semester economics student at LMU. Therefore, he has little experience in programming. In his course "Introduction to Programming", he receives the assignment to implement a simple calculator in Python. Justus prefers not to install any additional software on his computer, so he decides to use the online IDE Theia. He opens the IDE by using the "Open in Online IDE" Button in Artemis. The online IDE launches in a new tab with the problem statement automatically displayed in a sidebar next to the code editor. Justus reads the problem statement

and starts implementing the calculator. As he progresses, he scrolls through the problem statement for further instructions without leaving the IDE. Upon finishing the implementation, Justus submits the exercise using the "Submit" button. After a few seconds, the submission feedback is displayed in the sidebar of the IDE. Satisfied with his work, Justus decides not to improve his solution further.

Scenario 2: Student in an Exam Situation Steve is a fourth-semester computer science student at TUM. His programming exam in "Patterns of Software Engineering" is about to start. He opens his IDE, and as the exam begins, he quickly selects the relevant exam exercise from the list of exercises using the command line. The exercise is cloned, and the corresponding problem statement is automatically displayed in the sidebar. Steve starts by implementing the first task and submits it before moving on to the second task. While working on the second task of the exercise, the submission feedback for the first task is displayed in the sidebar. Steve reads the feedback and realizes he has made a mistake. He corrects the mistake and resubmits the first task alongside the completed second task. After a few seconds, the submission feedback is updated and displayed again. This time, Steve is satisfied with his score and hands in the exam.

3.4 Functional Requirements

After understanding how a student tackles an exercise with the help of the extension, we can define the functional requirements. The functional requirements are numbered and referenced throughout the document. The following requirements are necessary to provide a seamless integration of the Artemis platform into the IDE.

FR1	**Support Artemis authentication**: The extension must provide an authentication process via the Artemis API.
FR2	**Select exercises**: The student can select an exercise. The selection must clearly indicate which exercises are due next.
FR3	**Display problem statement**: The extension must display the problem statement of the selected exercise in a sidebar such that the code and the problem statement can be viewed simultaneously. The problem statement should include both text and UML diagrams of the exercise.
FR4	**Start exercise**: The student can start the exercise from within the IDE. This should create the corresponding repository on the Artemis server and clone it into the IDE.
FR5	**Clone repository**: The student can clone already-started exercises into the IDE. Repositories cloned by standard git commands should also be tracked.
FR6	**Detect repository automatically**: The extension should automatically detect any open Artemis repositories in the user's workspace and display the correct problem statement accordingly.
FR7	**Submit exercise**: The student can submit the exercise from within the IDE. The submission should be passed to the Artemis server.
FR8	**Display submission feedback**: The problem statement must include submission feedback. This should include per-task feedback and overall score.
FR9	**Collect user satisfaction**: After submitting an exercise, the extension should present a brief survey to collect user feedback. This data should be stored within a microservice architecture extending the Artemis platform and accessible to course instructors.
FR10	**Support different exercise modes**: The extension should support different exercise modes, such as individual, team-based, and exam exercises.
FR11	**Integrate into Artemis' Online IDE**: The extension should be integrated into the Online IDE of Artemis. All features available in the local environment should also be functional in the online IDE, with adjustments for context-specific adjustments.

Table 1: Functional Requirements

3.5 Nonfunctional Requirements

In this section, we list the requirements which are not directly related to the functionality of the system. We categorize them using the FURPS+ model (Func-

tionality, Usability, Reliability, Performance, Supportability, and Constraints) described in [Brü+04].

NFR1	**Artemis UI Components** (Usability): The extension should utilize UI components known to the user from the Artemis web client. This guarantees a consistent user experience.
NFR2	**User Interface for differently experienced users** (Usability): The extension should provide both a beginner-friendly GUI and a command line interface for advanced users.
NFR3	**Artemis Security** (Security) : The extension must interact with the Artemis API in a secure way without introducing new vulnerabilities.
NFR4	**VSCode Compatibility** (Constraint): The extension must be compatible with the Visual Studio Code IDE.
NFR5	**Theia Compatibility** (Constraint) : The extension must be compatible with the Theia IDE.
NFR6	**Online and local IDE** (Portability): The extension must be portable to both the online IDE (Theia) and the local IDE (Visual Studio Code) without any adjustments to the codebase.

Table 2: Nonfunctional Requirements

4 System Design

This chapter describes the system design of the `Scorpio` Extension for the Artemis platform. We will first derive the design goals from the non-functional requirements and prioritize them. We will then decompose the system into subsystems and describe the services provided by each subsystem. Finally, we will map the subsystems onto existing hardware and software components and discuss the security and authentication mechanisms of the extension.

4.1 Design Goals

The following section derives the Design Goals from the key non-functional requirements, specifically focusing on security, usability, and portability. These design goals are prioritized to reflect its integration into the existing Artemis platform. The design goals are listed below in order of their priority in the development process:

1. Security: The highest priority is security, given that the extension acts as a new client to an existing platform and is used by numerous students across a variety of courses. The extension should not introduce new vulnerabilities (NFR3), which could potentially leak sensitive student data, including personal information and academic submissions.

2. Usability: The second design goal is usability. The extension is a user-facing software meant to be integrated into the students' exercise workflow. Therefore, the extension must be usable for all experience levels (NFR2), from novice programmers to advanced users. Furthermore, the extension's usage should feel intuitive and familiar to the student. This can be achieved by using Artemis components known to the user from the web client (NFR1).

3. Portability: The final prioritized goal is portability. This aims to ensure that the extension can be used both locally and online. Since students have different preferences or technical constraints, the extension must run in both a local

VSCode environment (NFR4) as well as in the Theia Online IDE environment (NFR5). This portability allows students to freely choose their preferred working environment without sacrificing any functionality.

4.2 Subsystem Decomposition

The `Scorpio` Extension is divided into two main subsystems as illustrated in Figure 5: the `Exercise Lifecycle Controller`, which runs in a `Node.js` environment in the VSCode editor, and the `Plugin Sidebar`, which runs in a browser-like environment, a `webview`. This separation is necessary to interact with the VSCode API while also making use of the rendering capabilities of the webview.

The main purpose of the `Plugin Sidebar` is to provide a visual interface for the user, as seen in Figure 4. It displays the student's courses, exercises, submission feedback (FR8), and the exercise instructions (FR3) side by side with the code editor. The student can interact with the UI to select (FR2), start (FR4), clone (FR5) and submit (FR7) exercises.

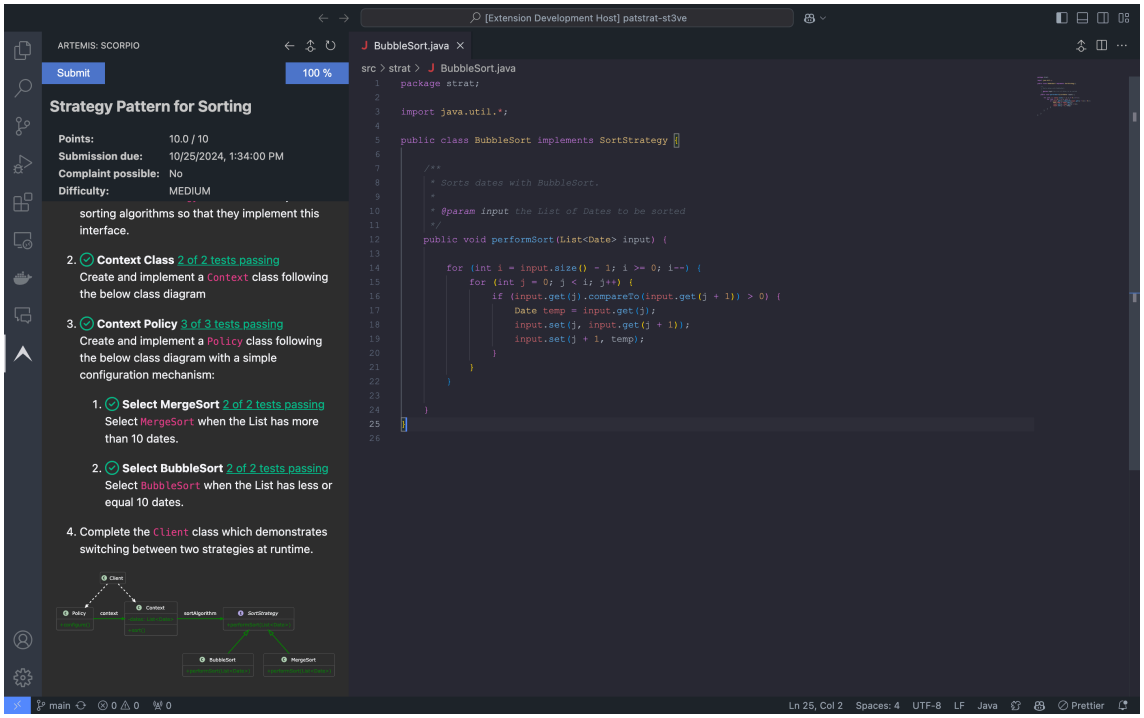

Figure 4: **Sidebar of the `Scorpio` Extension**: The sidebar provides a visual interface for the student to understand the exercise instructions and interact with the Artemis platform.

The `Exercise Lifecycle Controller` is the core interaction point between the `Scorpio` Extension and the VSCode API. It listens to events from the editor, such as newly opened workspaces, interacts with the `git` functionalities of VSCode, and manages states such as the authentication of the student (FR1). In addition, the `Exercise Lifecycle Controller` handles communication with external services, specifically the Artemis API, in a proxy-like manner. This involves fetching the student's courses, exercises, and submissions, as well as the capability to start an exercise. The `Exercise Lifecycle Controller` pushes any changed data to the webview.

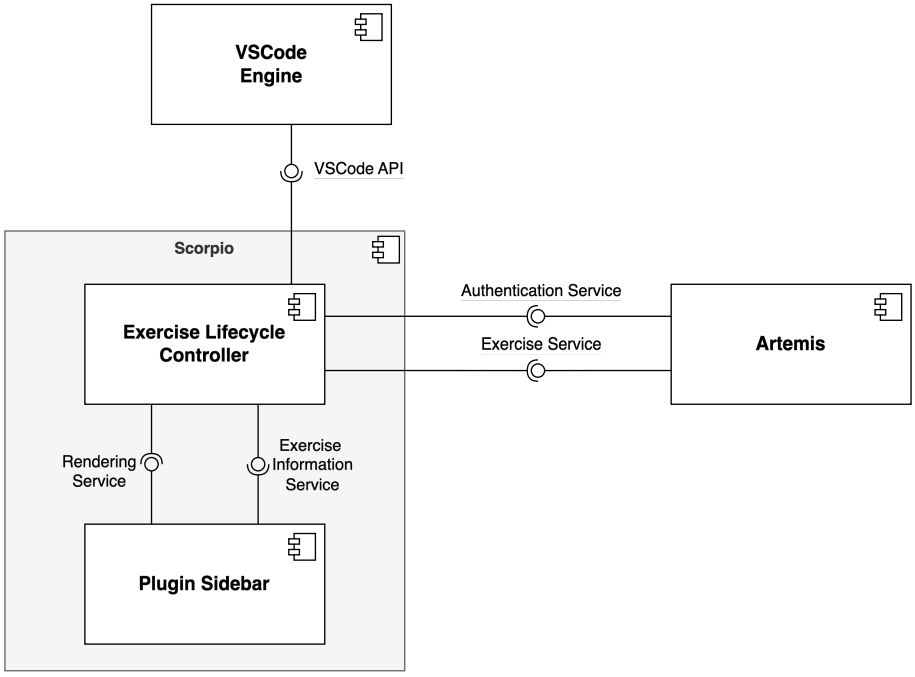

Figure 5: **Subsystem Decomposition of the `Scorpio` Extension.** The extension is divided into two main subsystems: the `Exercise Lifecycle Controller` and the `Plugin Sidebar`. The `Exercise Lifecycle Controller` interacts with the VSCode API and the Artemis API, while the `Plugin Sidebar` provides a visual interface for the student.

Both components communicate with each other via a message passing system. This allows the `Exercise Lifecycle Controller` to send data to the webview, which then updates the UI accordingly. The webview can also request data proactively to react to the user's input.

4.3 Hardware Software Mapping

The `Scorpio` Extension can run both in a locale IDE and an online IDE as required by NFR4 and NFR5.

Figure 6 illustrates the deployment on the student's machine. As depicted, the extension can run as an instance in both environments: one instance within the local VSCode IDE installed on the student's machine, while the other instance runs in the browser as part of the Theia online IDE. Both instances communicate in a client-server architectural style with the Artemis platform via the Artemis REST API.

The Artemis API is deployed on a remote server to centralize data processing and management. The extension instances act as a client that request data such as exercise information and submission feedback from the Artemis server.

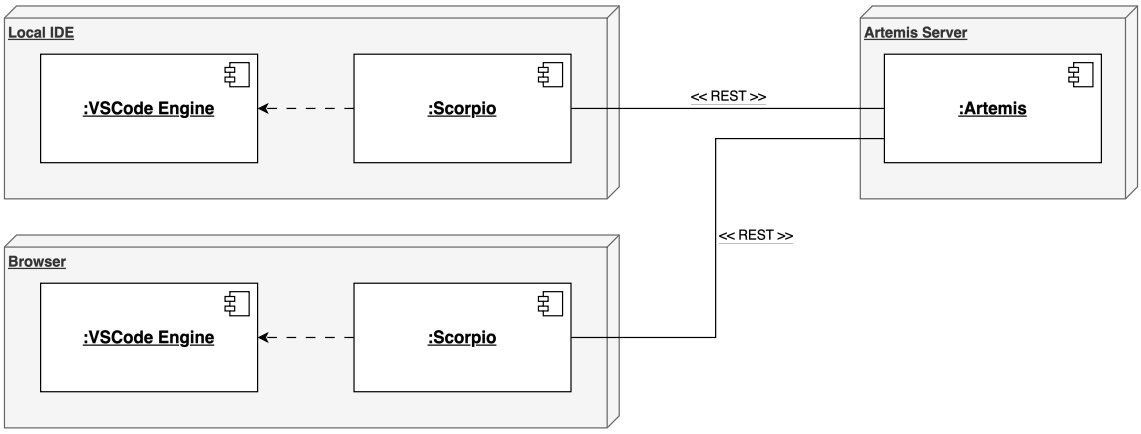

Figure 6: **Hardware Software Mapping for Browser and Local IDE.** The `Scorpio` Extension can run in both the local VSCode IDE and the Theia online IDE. Both instances communicate with the Artemis API via REST.

4.4 Security and Authentication

This chapter discusses the authentication process for the `Scorpio` Extension in both the local IDE and the online IDE environment. It analyzes how the security architecture of Artemis must be adjusted to guarantee a seamless authentication experience with minimal user input while maintaining the integrity of the system. The chapter will then propose a Single Sign-On (SSO) solution as a potential

enhancement to the existing authentication process, which could further reduce the number of credential inputs and improve the architectural design.

4.4.1 Authentication for Local IDE

The authentication process for the local IDE environment is illustrated in Figure 7. The student authenticates by providing their credentials through the `Scorpio` Extension. The extension forwards these credentials to the Artemis API, which authenticates the student and returns a JSON Web Token (JWT) [JBS15]. The JWT is then securely stored in the VSCode secrets storage, ensuring it is available for all subsequent requests made to the Artemis API.

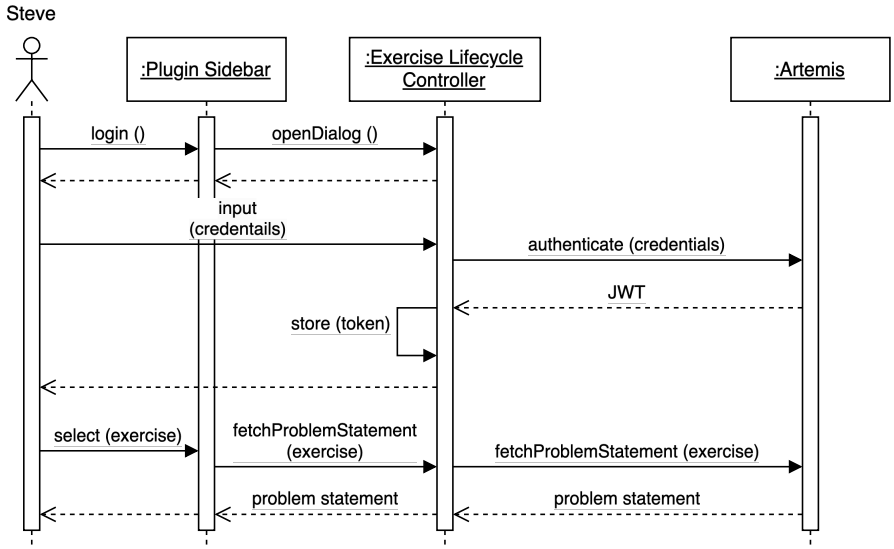

Figure 7: **Authentication Process for Local IDE.** The student authenticates against the Artemis API from within the `Scorpio` Extension. The returned JWT is securely stored in the VSCode secrets storage.

All requests to the Artemis API are routed through the `Exercise Lifecycle Controller`, which attaches the JWT to the request headers. This process ensures that the JWT is never exposed to the webview, minimizing the risk of unauthorized access to the secrets. Furthermore, the webview, as it is running in a browser-like environment, is subject to Cross-Origin Resource Sharing (CORS). This prevents the webview from directly interacting with the Artemis API, which is hosted on a different domain. The `Exercise Lifecycle Controller` component acts as a secure

proxy, forwarding the request to the Artemis API and returning the responses to the webview.

4.4.2 Authentication for Online IDE

The authentication process for the online IDE environment differs slightly from the local IDE, as shown in Figure 8. In this context, the student is already authenticated in the Artemis web client. Entering his credentials a second time within the online IDE would be redundant and should not be required. To address this, a new token is generated explicitly for use within the Theia IDE. The structure of this token can be seen in Appendix A.

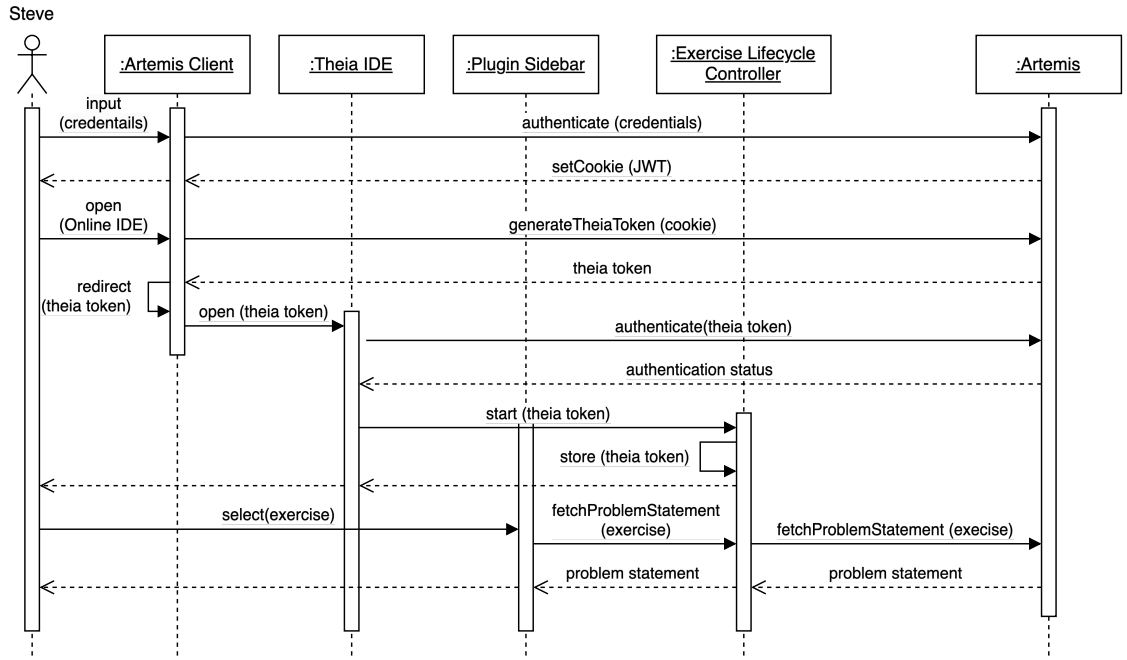

Figure 8: **Authentication Process for Online IDE.** The student is already authenticated in the Artemis Web Client. A new *Theia Token* is generated for the online IDE, which is used to authenticate the online IDE's session. The token is then forwarded to the `Scorpio` Extension for subsequent API requests.

In this process, the student first logs into the Artemis Web Client, which triggers the standard client-server authentication flow with the Artemis API. The generated JWT is stored in the browser's cookies. When the student opens the online

IDE, the cookie is automatically sent to the Artemis API to generate a new *Theia Token* for use within the online IDE.

This Theia token has limited access rights and is restricted to accessing only routes required for the exercise workflow according to the `least privilege principle` [SS75]. Its maximum lifetime is limited to either the remaining lifetime of the JWT or a maximum of one day. This ensures that the token is only valid for a short period of time, reducing potential security risks.

The Theia token is passed along with the redirect to the online IDE, where it is used to authenticate the online IDE's session and forwarded to the `Scorpio` Extension for subsequent API requests.

4.4.3 Single Sign-On

To further streamline the authentication process, the implementation of a Single Sign-On (SSO) mechanism is proposed. While the implementation is beyond the scope of this thesis, SSO could minimize the number of credentials inputs required by the student while improving the overall security and scalability of the authentication architecture.

Figure 9 illustrates the changed authentication flow for the Online IDE with the integration of SSO. The process for the local IDE can be derived from this.

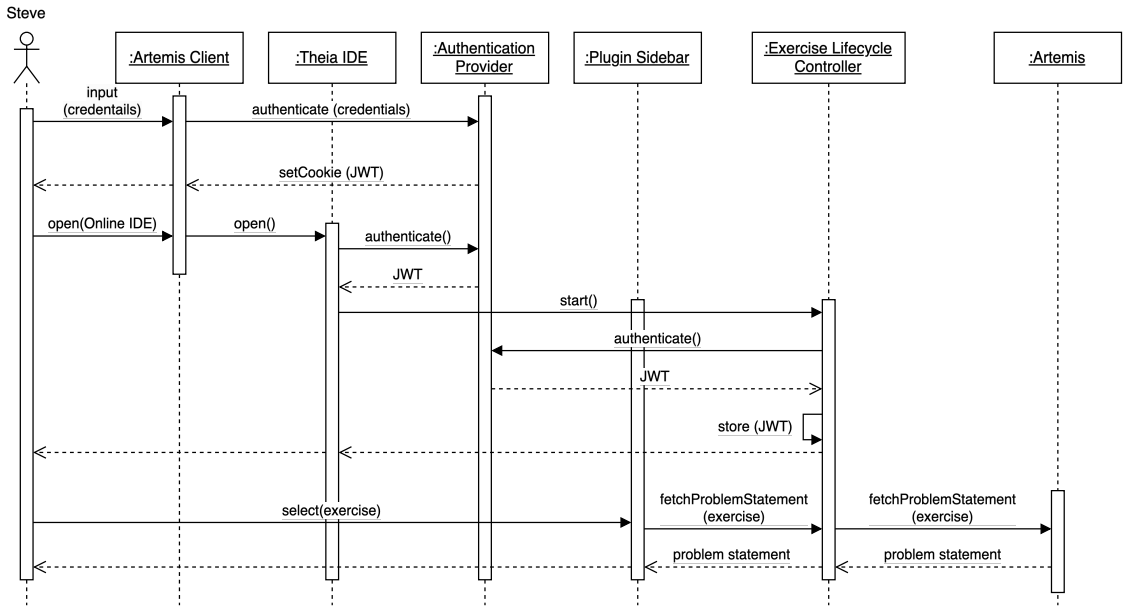

Figure 9: **Authentication Proposal with SSO.** The student authenticates against an authentication provider that supports SSO. The online IDE and the `Scorpio` extension authenticate against the same authentication provider, which issues a token for each service. The tokens are securely stored and used for subsequent API requests.

In the proposed SSO flow, the student logs into the Artemis Web Client as before. The server-side authentication is handled by an authentication provider that supports SSO. This authentication provider could either be integrated into the Artemis API or exist as a separate service.

When the student opens the online IDE, the IDE authenticates against the same authentication provider. If the student has an active session, the provider issues a token. This token is only used to authenticate the online IDE and is discarded afterward. Importantly, this process happens in the background without the student noticing.

In parallel, the online IDE initializes the `Scorpio` Extension, which also authenticates against the authentication provider. The returned token is securely stored in the VSCode secrets storage, allowing the extension to interact with the Artemis API as needed. Again, this entire process happens in the background without the student noticing and without additional input from the student.

The key advantage of the SSO approach is that the student only needs to authenticate once, while each service can independently authenticate against the authentication provider. This architecture allows for more granular access control, as the authentication provider can decide which service is allowed to access which resources. This separation of concerns allows for a more modular and secure system design.

5 Challenges in Former Implementations

In a former implementation of the extension, the `Exercise Lifecycle Controller` did not serve as a proxy for handeling requests. Instead, each subsystem maintained its own state for the currently displayed exercise and communicated directly with Artemis. This architecture, seen in Figure 10, presented several challenges and was ultimately deemed unsuitable for the final solution. The evolution of this approach should be explained, highlighting the complexity encountered and transition to the final design.

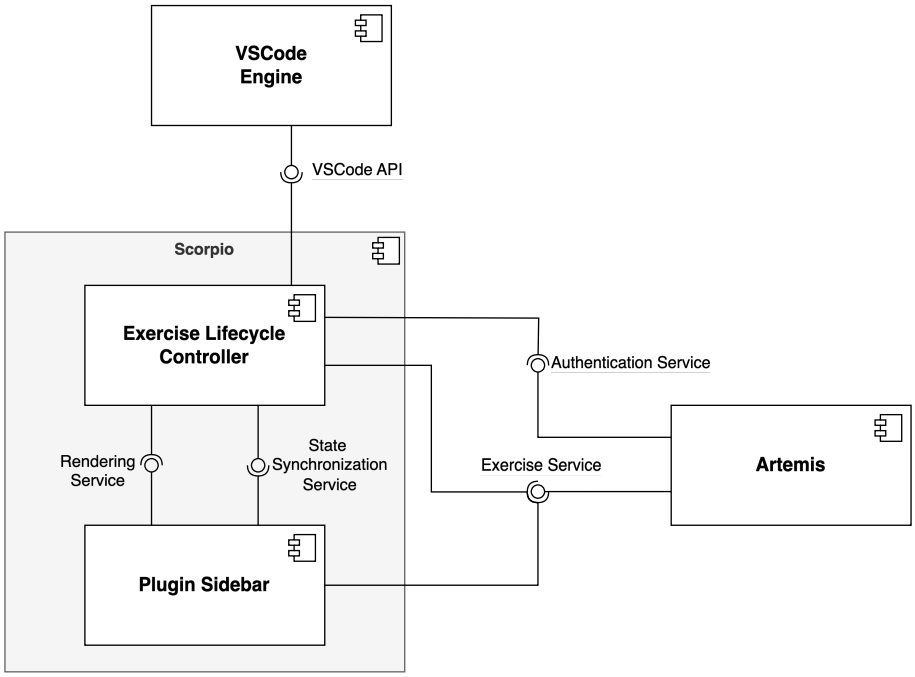

Figure 10: **Former Subsystem Decomposition of the `Scorpio` Extension.** The extension is divided into two main subsystems: the `Exercise Lifecycle Controller` and the `Plugin Sidebar`. Each requesting and keeping their own state of the currently displayed exercise. The `Exercise Lifecycle Controller` additionally interacts with the VSCode API.

5.1 Capabilities of VSCode Webviews

One of the key requirements was to display custom components within the sidebar, such as the problem statement. The most suitable solution for achieving this in a VSCode Extension is through a webview[12]. A webview is a custom HTML-

[12]https://code.visualstudio.com/api/extension-guides/webview

based view that can be displayed in the sidebar, editor, or panel of the IDE. It supports the rendering of a basic HTML string. Therefore, all necessary styling and scripting must be embedded in this HTML content. To maintain a modular and manageable codebase, the HTML and JavaScript code was separated into distinct files. A build pipeline then combines these files into a single HTML string, which can be passed to the webview. This approach enhanced the maintainability of the codebase, as it allowed for clear separation and structured code.

5.2 Embedding Artemis via Iframe

As the problem statement is already rendered effectively in the Artemis web client, reusing the existing functionality without reimplementing the component seemed like a reasonable approach. By embedding parts of the Artemis web client directly in the extension sidebar, the user experience would remain consistent while minimizing the codebase that requires active maintenance.

The initial solution involved embedding the problem statement in an iframe in the sidebar. A dedicated route already existed in the Artemis web client for specifically displaying the problem statement. Several adjustments were necessary to make use of this route.

5.2.1 Enabling Cross-Origin Resource Sharing (CORS)

As VSCode webview runs in a browser-like environment, it is subject to cross-origin policies. The webview's host differs from that of the Artemis API, which means that requests made by the webview fall under the restrictions of Cross-Origin Resource Sharing (CORS). To allow the webview to communicate with the Artemis API, the API must permit requests from the webview's origin. When sending requests to the Artemis API in a local development environment, one could observe that the request originated from a URL similar to a deep link `vscode-webview://*`. This origin pattern can be explicitly added to the list of allowed origins in the CORS configuration of the Artemis API, thereby enabling cross-origin requests from the VSCode webview.

5.2.2 Utilizing Artemis Cookies for Authentication

The Artemis web client uses cookies to authenticate with the Artemis API. Since the problem statement route, which is embedded in the iframe, makes internal requests to the Artemis API to retrieve exercise details, the authentication cookie has to be used in the webview.

We implemented a simple re-keying mechanism on the Artemis API to reuse the token already stored in the VSCode's secrets storage from the initial login. This mechanism receives the authentication token in the request body and, after validation, returns the same token as a `Set-Cookie` header in the response, effectively setting the token as a cookie.

However, as the cookie is set for the Artemis host, it is not directly accessible by the webview, which has a different host. The `SameSite` attribute of the cookie must be set to `None` to make it accessible for cross-origin requests from the webview. This configuration allows the cookie to be sent with requests originating from different domains, thereby enabling authentication within the iframe. Setting the `SameSite` attribute to `None` inherently makes the cookie vulnerable to Cross-Site Request Forgery (CSRF) attacks.

5.2.3 Implementing Cross-Site Request Forgery (CSRF) Protection

To minigate the vulnerability exposed by the `SameSite=None` cookie attribute, a CSRF protection mechanism had to be implemented. The implemented mechanism was based on the custom headers approach, as recommended by the Open Web Application Security Project (OWASP)[13].

In this approach, each request to the Artemis API must include a custom header, in our case, `X-ARTEMIS-CSRF`. This header forces the request into the CORS preflight check. The CORS preflight check verifies that the request originates from an authorized domain. If the custom header is missing or if the request originates from an unauthorized domain, the request is blocked. This measure ensures that only permitted origins can access the Artemis API, protecting it from CSRF

[13]https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#employing-custom-request-headers-for-ajaxapi

attacks while allowing cross-origin functionality within the iframe.

This iframe solution provides users with a consistent experience, leveraging the existing Artemis web client to maintain a minimal codebase. At the same time, it ensures the security and integrity of the Artemis API.

5.3 CORS Challenges with Remote Servers

While the iframe-based solution described above operates perfectly in a local development environment, significant challenges arise when transitioning to a remote server. Specifically, in remote setups, the VSCode webview host is not included in the request headers. As a result, the Artemis server can not verify the origin of the request, and the request is blocked. Allowing such requests would pose a security risk to the Artemis server.

VSCode addressed this issue in a feature proposal by using Cross-Origin-Isolation (COI) in VSCode webviews[14]. However, as this feature is still experimental, we decided not to use it in the final solution.

5.4 Transition to Native Rendering

While the iframe solution proved effective in the local development setting, it was ultimately unsuitable for remote server deployment. Additionally, future adjustments, such as offline capabilities, would be challenging to implement with the iframe approach. To address these limitations, a migration to a native rendering solution was proposed. This approach would only fetch exercise details directly from the Artemis API and render them autonomously in the extension sidebar, eliminating the need for an iframe and the associated CORS and authentication challenges.

5.4.1 Separating Problem Statement into Logic and View

The Artemis web client already implements the rendering capabilities of the problem statement. As mentioned earlier, this component performs internal

[14]https://github.com/microsoft/vscode/issues/137884

requests to the Artemis API to fetch exercise details, making it subject to authentication and CORS restrictions. These limitations were avoided by shifting the data fetching to the `Exercise Lifecycle Controller` of the `Scorpio` Extension like seen in Chapter 4. This component runs in a `Node.js` environment connected to the VSCode API, which enables it to manage its own authentication and is not subject to CORS policies. Modularizing the problem statement component into logic and view, allowed sharing of the rendering capabilities between the web client and the `Scorpio` Extension. The rendering part was encapsulated into an Angular library, which can be imported and used by the `Scorpio` Extension. This approach allows the extension to leverage existing rendering functionalities without replicating code.

5.4.2 Migrating the Sidebar to Angular

Since the Artemis web client is built with Angular, the rendering portion of the problem statement component was also implemented using Angular. Hence, the extension's sidebar had to be migrated to Angular to allow integration of the Angular library. This transition required a modified build pipeline to compile the Angular code into a single HTML string[15]. Following the migration, the library is utilized to render the problem statement directly in the sidebar of the extension.

This approach allows the extension to use the same rendering framework as the web client, keeping a consistent experience for the user and a minimal codebase to maintain.

The transition to native rendering resolved the challenges faced with the iframe approach. It also laid the foundation for future enhancements, such as offline capabilities. The migration to Angular further streamlined the rendering process, ensuring a consistent user experience across the Artemis platform while reducing maintenance overhead for developers.

[15]https://github.com/microsoft/vscode-webview-ui-toolkit-samples/tree/main/frameworks/hello-world-angular

6 Summary

This chapter summarizes the development of the VSCode Extension `Scorpio` for the Artemis platform. We will first discuss the current status of the project as of writing this thesis. The status is divided into realized and open goals. We will then conclude the thesis by discussing the contributions of the extension to the Artemis platform and giving an outlook on potential future work.

6.1 Status

Abbreviation	Functional Requirement	Status
FR1	Support Artemis authentication	Done
FR2	Select exercises	Done
FR3	Display problem statement	Done
FR4	Start exercise	Done
FR5	Clone repository	Done
FR6	Detect repository automatically	Done
FR7	Submit exercise	Done
FR8	Display submission feedback	Done
FR9	Collect user satisfaction	Open
FR10	Support different exercise modes	Partially Done
FR11	Integrate into Artemis' Online IDE	Partially Done

Table 3: Status of Functional Requirements

As shown in Table 3, all functional requirements related to the exercise lifecycle have been successfully implemented. The `Scorpio` Extension allows students to authenticate with the Artemis API, select, start, and clone exercises, display problem statements within the IDE, submit exercises, and view submission feedback. Additionally, the extension can automatically detect open repositories within the workspace and display the corresponding problem statement, providing a seamless experience for the user.

At the time of writing, the extension fully supports individual and team exercises. However, the functionality for exam exercises has not yet been fully tested. Given that exam exercises are expected to closely resemble individual

exercises in structure and interaction, supporting exam exercises should require minimal modifications to the extension's codebase.

The online IDE-specific features of the extension have been implemented. This includes functionalities such as the automatic cloning of repositories using a provided URL and an automatic authentication process without additional credential input. A comprehensive integration test of these features is scheduled for November 5th, 2024, to ensure the online IDE operates as expected and provides a comparable experience to the local IDE.

One key feature that remains unimplemented is the collection of User Satisfaction Feedback. Due to time constraints, this feature was postponed. The plan is to implement this in future iterations of the extension, allowing instructors to gather valuable feedback from students after exercise submissions.

6.2 Conclusion

In this thesis, we successfully integrated the whole exercise lifecycle of the Artemis platform into the student's coding environment. We developed the `Scorpio` Extension for Visual Studio Code, which allows students to authenticate with the Artemis API, select, start, and clone exercises, display the exercise instructions next to the code, submit exercises, and view submission feedback. Additionally, the extension can automatically detect open repositories within the workspace and display the corresponding problem statement, providing a seamless experience for the user.

6.3 Future Work

As this thesis presents the initial creation of the extension, there are numerous opportunities for future enhancements and additional features. The following section outlines potential directions for future development:

6.3.1 Linking problem statement snippets directly to code sections

In programming exercises, instructions are often directly related to specific code elements, such as class names, method signatures, or structural diagrams like

UML. Linking these snippets directly to the code can help students to easily navigate to the relevant code sections.

For example, by clicking on a class name mentioned in the problem statement, the IDE could automatically navigate to the corresponding file with the class definition. Similarly, a more refined navigation mechanism could allow students to interact with UML diagrams. By selecting a method name in the diagram, the IDE would jump directly to the relevant code line where that method is defined.

6.3.2 Iris Chat Extension

Since most educational institutions do not allow the use of AI assistants, Artemis has developed its own restricted language model assistant `Iris` [BFK24]. Unlike most powerful AI assistants, `Iris` is trained to respond to students' requests without providing direct solutions. It focuses on clarifying the problem statement and helping the students understand their tasks.

A future version of the `Scorpio` Extension could integrate `Iris` to provide additional support to students within the IDE. Similar to the popular extension `github copilot`[16], this feature would allow students to ask questions about the problem statement, the tasks or the codebase and receive guidance without exposing the full solution.

Additionally, a restricted form of code completion could be added to the extension. This would allow Iris to generate non-critical code elements, such as documentation or getter and setter methods, to help students focus on the core logic of their solution. Customizable code completion would offer a balance between educational guidance and real-world application.

6.3.3 Tutor Capabilities

Currently, the `Scorpio` Extension only offers functionality for the student's side of exercises. However, extending the functionality to support instructors and tutors could add significant value to manual assessment workflows. The main task for tutors is to manually review and assess students' submissions, ensuring that

[16]https://github.com/features/copilot

automated tests have evaluated the code correctly. A potential feature could enable an in-code assessment to make the process more efficient and faster.

The extension would visually mark code sections associated with failed tests in a student's submission, allowing the tutor to quickly navigate to the relevant lines and evaluate the test results. A mockup of this feature can be seen in Figure 11.

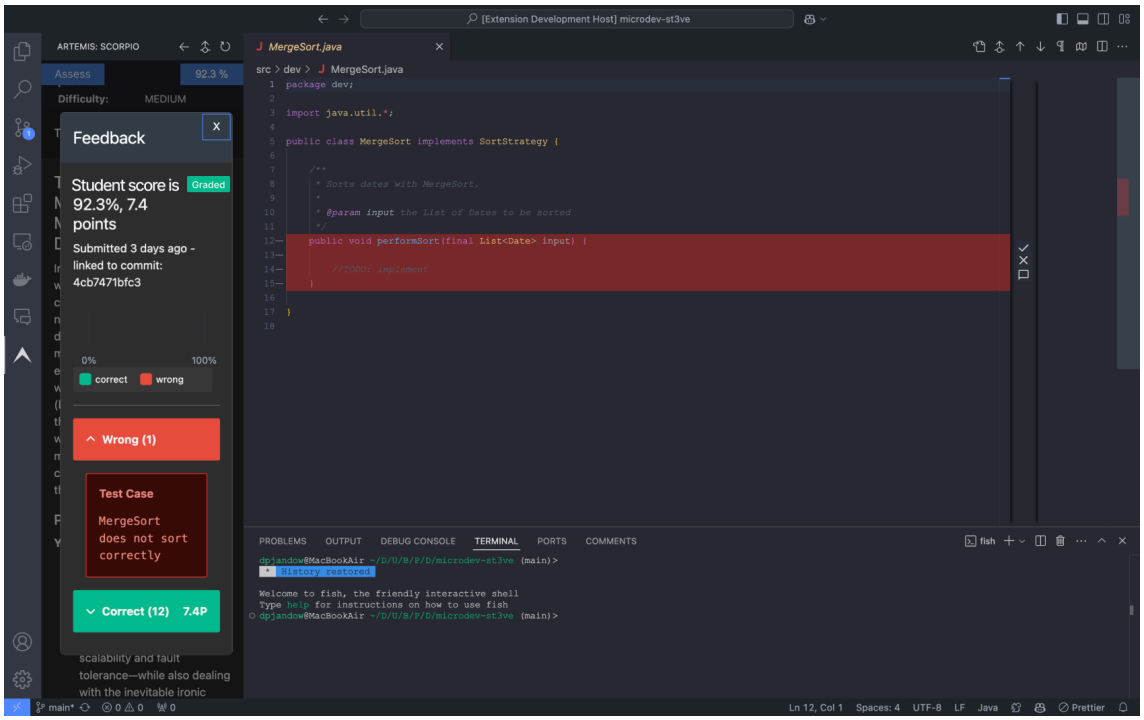

Figure 11: **Mockup of Tutor Assessment Feature**: The tutor can see the code section that caused failed tests and navigate to them. The tutor can then accept or deny the test assessment manually.

The tutor sees highlighted sections associated with a failed test directly in the code. He can then either accept or deny the test assessment by the buttons provided on the right. Additional details of the failed test can be seen in the sidebar. Navigation tools similar to VSCode's merge editor enable the tutor to quickly navigate between flagged sections. This feature would offer tutors an efficient, integrated interface for providing feedback to students.

List of Figures

List of Tables

Appendix A Theia Token

Encoded Token:

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJzdDN2ZSIsImF1dGgiOiJST0xFX1VTRVIiLCJ0b29scyI6IlRIRUlBIiwiZXhwIjoxNzMwMTE5MjgyfQ.0ctItZR4CM8HLKe4RjdB8UGXinSGqusDNzcaj7V071Lg-alhXQIZsCOXnuNZMw6MRZas5W15zxM22H0ApiKV_w

Decoded Token:

```
{
  "alg": "HS512"
}
{
  "sub": "st3ve",
  "auth": "ROLE_USER",
  "tools": "THEIA",
  "exp": 1730119282
}
HMACSHA512(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  256-bit-secret
)
```

Bibliography

[KS18] S. Krusche and A. Seitz, "ArTEMiS: An Automatic Assessment Management System for Interactive Learning," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Baltimore Maryland USA: ACM, Feb. 2018, pp. 284–289. doi: 10.1145/3159450.3159602.

[Tur20] A. Turdiu, "Online Exams in Artemis," 2020.

[Sch21] L. Schlesinger, "METIS: Multiplying Engagement Through Interacting Socially on the Artemis Learning Platform," Nov. 2021.

[Kel23] L. Keller, "Amplifying Engagement, Interaction, and Communication in Learning Platforms," Dec. 2023.

[BFK24] P. Bassner, E. Frankford, and S. Krusche, "Iris: An AI-Driven Virtual Tutor For Computer Science Education." May 2024. doi: 10.1145/3649217.3653543.

[DL12] J. C. W. Debuse and M. Lawley, "The Learning and Productivity Benefits to Student Programmers from Real-World Development Environments," 2012.

[Sch24] Y. Schmidt, "Inclusive Learning Environments in the Cloud: Scalable Online IDEs for Software Engineering Education," 2024.

[SAK11] J. Sweller, P. Ayres, and S. Kalyuga, *Cognitive Load Theory*. New York, NY: Springer, 2011.

[Jud15] T. Judd, "Task Selection, Task Switching and Multitasking during Computer-Based Independent Study.," *Australasian Journal of Educational Technology*, vol. 31, 2015, doi: 10.14742/AJET.1992.

[WKM22] J. Wathen, A. Kans, and G. Malik, "CODI – a Web Application to Facilitate Live, Remote Programming Lab Sessions," *2022 IEEE Global Engineering Education Conference (EDUCON)*, pp. 2053–2061, 2022, doi: 10.1109/EDUCON52537.2022.9766755.

[Ung20] A. Ungar, "Development of an IDE Plugin for ArTEMiS," 2020.

[Dun21] M. Dunker, “Development of Manual Assessment for Programming Exercises in the Orion Plugin,” 2021.

[Gra23] J.-L. Grabowski, “Enhancing Assessment and Feedback for Programming Exercises in Orion,” 2023.

[Brü+04] B. Brügge, B. Bruegge, A. H. Dutoit, and A. H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns, and Java*, 2. ed., internat. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004.

[JBS15] M. B. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” no. 7519. RFC Editor, May 2015. doi: 10.17487/RFC7519.

[SS75] J. Saltzer and M. Schroeder, “The Protection of Information in Computer Systems,” *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975, doi: 10.1109/PROC.1975.9939.