Development of a Gamifiable Application for Programming Education

By Svenja Sutter, Lukas Messmer, Mathias Fischler

Bachelor's Thesis
Department of Computer Science
Eastern Switzerland University of Applied Sciences
Advisor: Prof. Dr. Frieder Loch
Co-Examiner I: Dr. Juliane Fischer
Co-Examiner II: Prof. Frank Koch
Date: 14.06.2024

OST
Eastern Switzerland
University of Applied Sciences

Abstract

This Bachelor's thesis documents the implementation and design of *Codable*. *Codable* is a user-centered application for creating, managing and solving exercises in an academic environment. Its main goal is to improve the organizational and qualitative problems of exercises in the Department of Computer Science, which were identified in the preceding semester thesis written by Lukas Messmer and Mathias Fischler. The paper focuses on the architectural design of *Codable* and documents aspects such as the implemented plugin system, the automation process for exercise submissions and evaluations, as well as the architectural decomposition of the system.

About this Document

This document is structured into three parts:

1. **Product Documentation**: The first part documents the product that was developed as part of the Bachelor's thesis. It is written in a non-time-bound manner using the arc42 Template (Starke & Hruschka, 2024) and includes a description of the architecture, the implemented features as well as other product-related topics. This part is intended to be used for further development after the completion of this Bachelor's thesis.
2. **Project Documentation**: The second part documents the Bachelor's thesis itself. It contains a rough overview of the milestones, a description of the conducted work as well as other formalities. This part is only meant to serve as a reference for the examination committee.
3. **Appendix**: The last part lists relevant work items for both the product and project documentation. It is indented to provide additional details on certain topics if their documentation was not considered useful in any of the other parts.

Table of Contents

1. Management Summary

1.1. Introduction

In this Bachelor's thesis, we were tasked with the creation and implementation of *Codable*. *Codable* is a new user-centered application for creating, managing and solving exercises in an academic environment. Its main goal is to improve the organizational and qualitative problems of exercises in the Department of Computer Science, which were identified in the preceding semester thesis written by Lukas Messmer and Mathias Fischler (Messmer & Fischler, 2023).

1.2. Details & Result

In its current state, *Codable* allows the creation and management of exercises by lecturers and assistants. Exercises are structured into courses and can be granularly configured both in terms of their content (i.e. what the exercise should contain) as well as their logic (i.e. how the exercise should be evaluated). At the same time, the application provides a uniform interface for students to solve and submit exercises either locally with Git or via the browser.

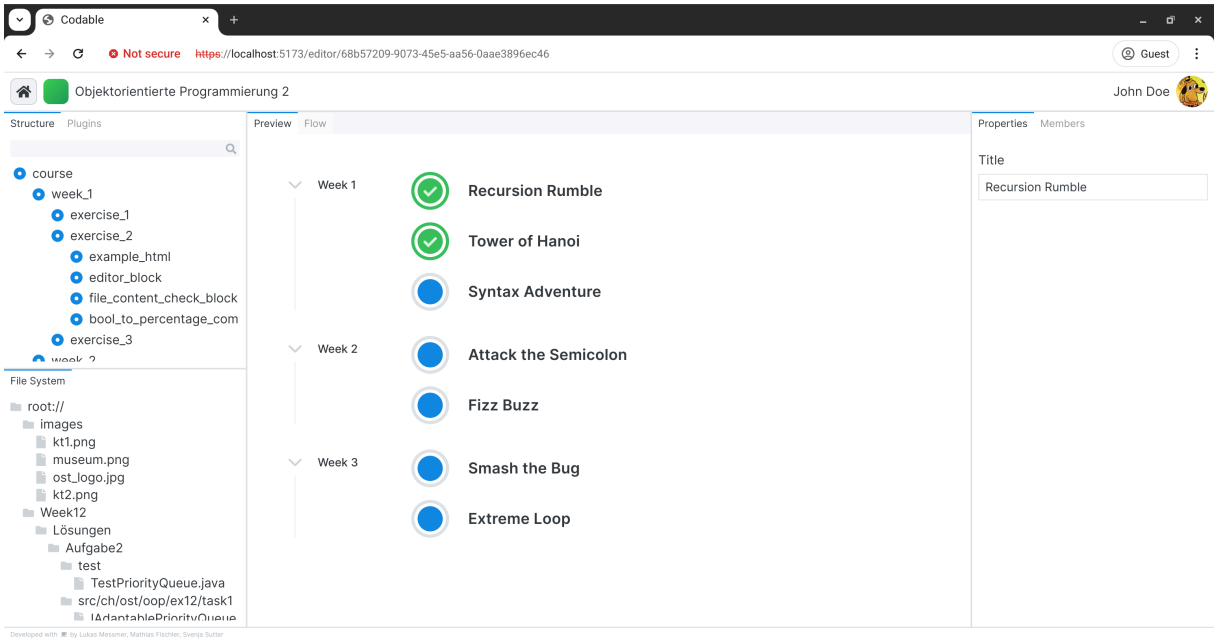


Figure 1: *Codable* allows extensive customization of exercise contents and logic

The main strength of *Codable* comes from its modular plugin system, which allows lecturers to extend the functionalities of the application with regard to the exercise contents by implementing a dedicated C# interface. In addition, the application enables modeling exercise logics using a flow-engine-like system, which means that sophisticated workflows can be created that automatically run when students submit their exercises. Both of these features facilitate the streamlining of exercises across different courses, making it easier for students to solve and submit exercises even when complex tools or evaluation processes are required in the background.

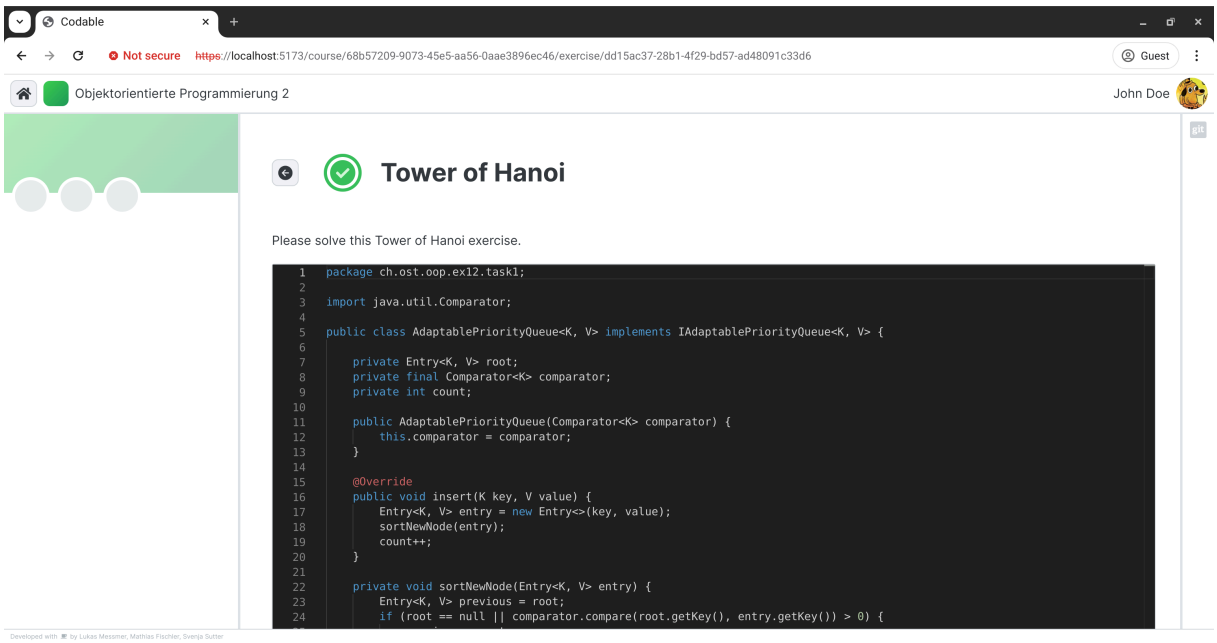


Figure 2: All exercises can be solved in the browser or locally using Git

1.3. Outlook

Codable is intended to be used productively in the coming semesters at the Eastern Switzerland University of Applied Sciences. If the application is well received, further development might take place to improve upon its current features. We are confident that *Codable* can considerably increase both interest and engagement in exercises if resources are provided to ensure the sustainable growth of the system. Additionally, the foundational work done in both this and the preceding thesis can be used to implement and expand the gamification mechanism initially envisioned.

Product Documentation

2. Introduction & Goals

2.1. Context

This paper documents the architecture of *Codable*[1]. *Codable* is a user-centered application for creating, managing and solving exercises in a academic environment. Its main goal is to improve the organizational and qualitative problems of exercises in the Department of Computer Science, which were identified in a semester thesis written in 2023 (Messmer & Fischler, 2023). The application is being developed in collaboration with the Department of Computer Science at the OST – Eastern Switzerland University of Applied Sciences.

2.2. Requirements Overview

Codable allows the creation and management of exercises by lecturers, assistants or similar (so-called *exercise makers*). Exercises are structured into courses and can be granularly configured both in terms of their content (using a modular plugin system) as well as their logic (using a flow-engine-like system). At the same time, the application provides a uniform interface for students (called *exercise solvers*) to solve and submit exercises either locally using Git or via the browser.

The functional requirements of the system are defined using 59 use cases organized into 6 epics, which are shown in Table 1. All use cases can be found in the appendix under Section II.A or in the internal GitLab repository.

IID	Requirement
163	The application allows creating and modifying courses.
164	The application allows adding custom exercise contents (plugins).
165	The application allows configuring exercise logic via a flow system.
166	The application allows working on exercises online via browser.
167	The application allows working on exercises locally via Git.
168	The application allows user management using OST systems.

Table 1: The main functional requirements (i.e. epics) of the system

The functional requirements are primarily derived from 101 feature requests (written as user stories), which describe the general needs and wants of the targeted user base.[2] These feature requests can be found in the appendix under Section II.B or in the internal GitLab repository.

[1] Due to legacy reasons, some resources may still use the former name “CodingQuiz”.
[2] Most of these user stories were formulated as part of the aforementioned semester thesis.

2.3. Quality Goals

Figure 3 shows the three most important quality goals of the system.

QA1	QA2	QA3
Extendable Exercises	**Minimal Workload**	**Uniform Interface**
The specific content or logic of exercises must be highly modifiable by exercise makers.	The modification of courses and exercises must be completable with a minimal amount of manual effort.	The user interface for exercise solvers must be consistent across different courses.

Figure 3: The main quality goals of the system

These quality goals were identified as part of the aforementioned semester thesis (Messmer & Fischler, 2023) and are important due to their high architectural significance. All quality goals and quality attribute scenarios can be found in Section 9. Details on how these quality goals influence the system can be found in Section 4.2.

2.4. Stakeholders

Table 2 shows the stakeholders of the system.

Name	Role & Expectations
Department of Computer Science	The Department of Computer Science is responsible for financing the further development of the system. They need to be convinced that the system is beneficial to achieve their organizational objectives in order to provide the necessary resources for further development.
I3 Institute for Interactive Informatics	The I3 Institute for Interactive Informatics is responsible for the further development of the system. They require a sufficient documentation in order to understand and continue work on the system.
Lecturers & Assistants	Lecturers and assistants are the main providers of content for the system. They need to be convinced that the system is suitable for them to invest their time and effort into it.
Students	Students are the primary users of the system. Although they have no direct influence on the system, they must be satisfied in order for the system to provide actual value to the Department of Computer Science.

Table 2: The stakeholders of the system

3. Context & Scope

3.1. System Context

Figure 4 shows the context of the system in relation to its external communication partners (i.e. other systems and users) using a C4 context diagram (Brown, 2018).

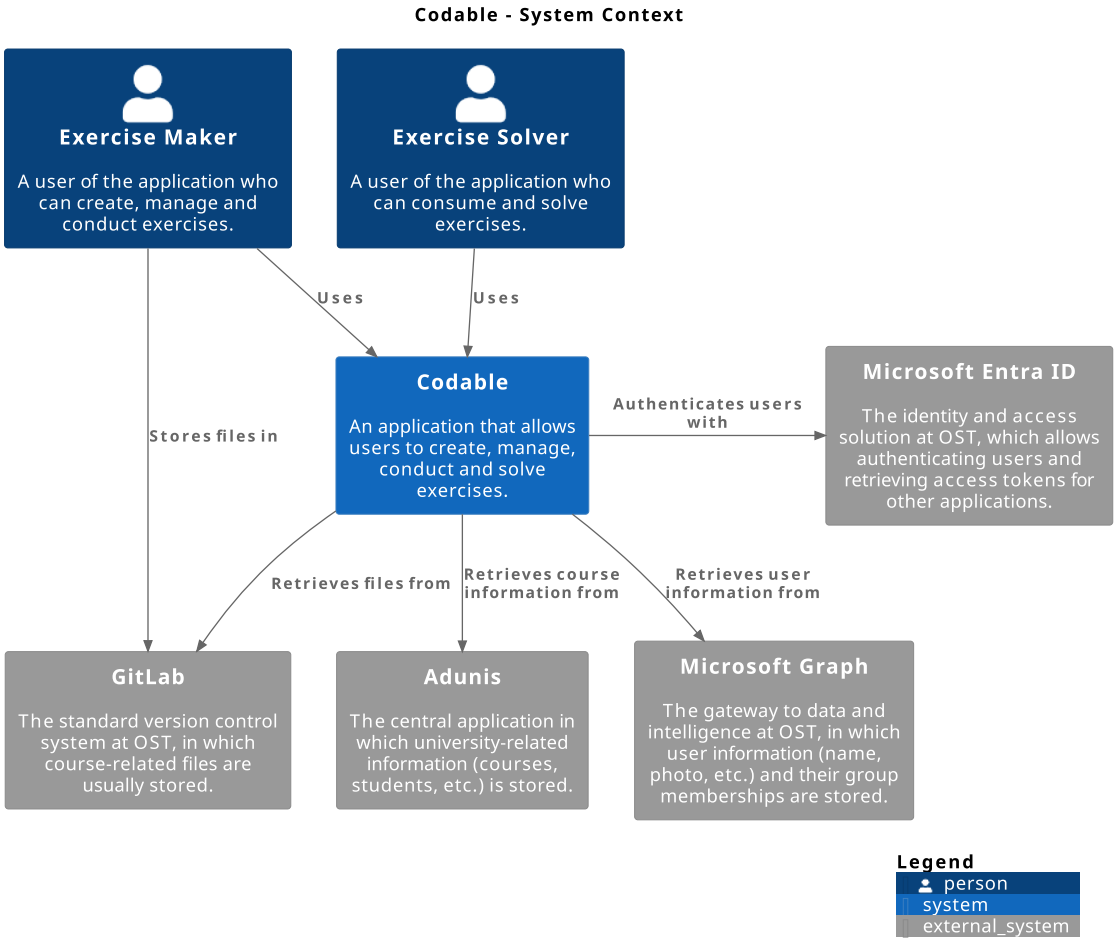


Figure 4: An overview of the scope and context of the application

3.1.1. Rationale

Y[3]	In the context of the external dependencies to GitLab, Adunis, Microsoft Entra ID and Microsoft Graph, facing the need to reduce management overhead for exercise makers (QA2), we decided to include these systems as communication partners, to achieve the ability to automatically transfer information (such as course name, members, files) without manual work by exercise makers, accepting the downsides of additional dependencies and complexity in our system.

ADR 1: Why do we accept so many external dependencies?

[3] Y-Statements (read "Why-Statements") are designed to summarize architectural decisions in a single, structured sentence (Zimmermann, 2022). This structure is used throughout this document.

Microsoft Entra ID: Microsoft Entra ID was chosen as the single sign-on (SSO) provider for the system because there is a guarantee that all of our targeted users (i.e. students, lecturers and other OST staff) will have an OST Microsoft account. This is not the case with other single sign-on providers such as GitLab, which requires users to first log in to the OST GitLab instance. At the same time, we do not want to be concerned with the security risks and administrative overhead of implementing our own authentication mechanisms, which is why we opted for a single sign-on provider in the first place. For more information about authentication and authorization, please refer to Section 8.4.

Microsoft Graph: Because we have chosen Microsoft Entra ID as our single sign-on (SSO) provider for the reasons stated above, Microsoft Graph is the most convenient choice for accessing data and information about the logged-in user.[4] Microsoft Graph stores various information (such as the user's name, photo and group membership) that we require for both the user experience and authorization mechanisms. For more information about authorization, please refer to Section 8.4.2.

[4]This is because Microsoft Graph is a protected API that requires authentication using an access token provided by Microsoft Entra ID, which we already use anyway. (Microsoft, 2023)

4. Solution Strategy

4.1. Organizational Decisions

The architecture of this system is primarily derived from feature requests (see appendix Section II.B) expressed as part of the user-centered design process employed by the development team (International Organization for Standardization, 2019). The conceptual foundation of the architecture is described in a semester thesis from 2023 (Messmer & Fischler, 2023). This user-centered approach should be maintained to ensure a long-lasting and healthy development of the system.

4.2. Architectural Decisions

This section provides rationals for the core architectural decisions of the system, particularly with regards to the quality goals defined in Section 2.3.

4.2.1. Plugin System

Y	In the context of extendable exercise content and logic, facing the need to provide highly modifiable exercises (QA1), we decided to develop a plugin system using .NET DLLs, and neglected other options such as providing predefined building blocks or allowing extension through technologies such as iframe[5], to achieve independent and extensive modifiability and extendability of exercises by exercise makers, accepting the downside of giving exercise makers access to all .NET Runtime features (R2) as well as significant additional complexity in our system.

ADR 2: Why are we implementing a plugin system?

Y	In the context of the plugin system, facing the need to provide a uniform and consistent experience for exercise solvers (QA3), we decided to create a system where courses and their hierarchical structure remain consistent with only the exercise content and logic being able to change, and neglected other options such as giving exercise makers the ability to change the course hierarchy using the plugin system, to achieve consistent workflows and interfaces across different exercises and courses.

ADR 3: Why is the plugin system limited to exercises?

Remarks: Balancing the need for highly modifiable exercises (QA1) with the need for a consistent interface (QA3) creates conflicts, since the plugin system supports extensive customisation that may introduce inconsistencies in the user interface. In addition to limiting the plugin system to exercises, frequently used plugins may be included as standard building blocks in the future to ensure uniformity.

[5]This could mean that we allow the exercises to be written in HTML, but if further logic is required, an iframe must be included.

4.2.2. Automation & Dependencies

Y	In the context of the workload of exercise makers, facing the need to minimize it as much as possible (QA2), we decided to integrate as many external dependencies as possible (such as importing course information from Adunis), to achieve a minimal amount of manual effort for our functional requirements (Section II.A), accepting the downside of having additional dependencies that are outside of our control.

ADR 4: Why do we integrate with multiple external dependencies?

Remarks: The need for minimal workload (QA2) also influences other aspects such as the information architecture (Laubheimer, 2022) of the system, for which we generally prefer a flat content hierarchy with a high information density per page. The intention is to enable quick creation and solving of exercises without having to navigate between different pages, resulting in a more streamlined experience.

Wherever possible, external dependencies should also be integrated on an optional basis in order to avoid rendering the application unusable if an external dependency is not available. This is not always possible, however, for example when authenticating users using the single sign-on (SSO) provider.

4.2.3. File Storage

Y	In the context of course and exercise file storage, facing the need to allow local work and reduce the disk space required for courses, we decided to use a Git server powered by Gitolite, and neglected implementing Git's "smart" or "dumb" HTTP protocol (Git Contributors, n.d.-a) manually or attempting any custom implementations of Git-like features, to achieve the ability to manage courses and their files using branching, versioning, and fine-grained access control, accepting the downside of having to run a quite unknown program (Gitolite) that falls under the GPL v2 license[6].

ADR 5: Why do we use a Git server (Gitolite)?

Remarks: By using Git, we can reduce the disk space of a course by storing all files of exercise solvers in the same repository but on separate branches. This allows Git to reuse the same *object* (Git Contributors, n.d.-b) as the hashes of the original course files and the files of exercise solvers are identical, meaning we only need to save the difference when an exercise is submitted.

[6]GPL v2 means that if we make any modifications to the Gitolite software, we are required to publish it again as open source under the GPL v2 license.

4.3. Technology Decisions

This section outlines the important technologies used in the system with additional explanations as to why they were chosen. Some of the technologies listed (i.e. .NET, React and SQLite) have already been examined and established in the semester thesis of 2023 (Messmer & Fischler, 2023). For information on where these technologies are used within the system architecture, please refer to Section 5.1.

4.3.1. Core Application (Backend)

Table 3 outlines the key technologies and libraries used in the Core Application.

Technology	Description
.NET (Backend)	.NET was chosen for its ecosystem and due to the existing experience and familiarity of the development team and the Department of Computer Science, which needs to be able to write plugins. It is also cross-platform, enabling development on multiple operating systems.
Autofac (IoC Container)	Autofac was chosen because it is an advanced Inversion of Control (IoC) container for .NET that simplifies dependency injection in our application. It allows for flexible and efficient management of dependencies and object lifetimes, ensuring that services are properly configured and resolved at runtime.
Funcky (Library)	This library enables the proper application of functional programming paradigms in C#.
EF Core (ORM Library)	Entity Framework (EF) Core is a well-established, open-source object-relational mapping (ORM) framework for .NET.
Microsoft Identity Web (Authentication)	The Microsoft Identity Web library was chosen because it is the official Microsoft library for integrating Microsoft Entra ID into ASP.NET applications.

Table 3: The important technologies of the Core Application

4.3.2. Client (Frontend)

Table 4 outlines the key technologies and dependencies used in the Client.

Technology	Description
React (Frontend)	Chosen because it is open source, has an active maintainer base, and is scalable, making it well suited for potential growth.
Zustand (State Management)	Selected for its simplicity, ease of implementation, and minimal overhead in comparison to Redux.

Technology	Description
OpenAPI (API Client Generation)	Utilized to generate TypeScript clients from OpenAPI specifications, streamlining API integration without the use of Java codegen in builds.
Radix (Component Library)	Provides a selection of unstyled UI primitives that are fully functional, accessible and highly customisable.
Monaco (Web Editor)	Monaco, the editor powering Visual Studio Code, offers syntax highlighting, IntelliSense, and multi-language support, enriching the online development experience.
MSAL React (Authentication)	The Microsoft Authentication Library (MSAL) for React was chosen because it is the official Microsoft library for integrating Microsoft Entra ID into React applications.

Table 4: The important technologies of the Client

4.3.3. Persistence

Table 5 outlines the technologies used for persistence.

Technology	Description
Gitolite (Git Server)	Gitolite was chosen because it allows per-branch permissions, can execute custom commands whenever a commit is pushed to a branch, and allows remote and fine-grained programmatic control of user authentication. All of these features are necessary to integrate the Git server with our system.
SQLite (Database)	SQLite was chosen for its simplicity and interchangeability. Using SQLite with Entity Framework (EF) Core allows for future drop-in replacement with other relational SQL databases (e.g. PostgreSQL, MariaDB/MySQL, etc.), which offer broader feature sets and have existing EF Core providers (Microsoft, 2024). Additionally, unlike other databases, SQLite requires no setup or installation as it is simply file-based.

Table 5: The important technologies for persistence

4.3.3.1. About Gitolite

As Gitolite is a quite unknown technology that serves an important role in our system, here are some additional remarks: Gitolite's repository and branch permissions are managed through an "admin" repository, which also contains the user's SSH keys. Gitolite can also be configured to allow executables (specifically *VREFs* (Chamarty, n.d.)) to be defined for events over this repository. This enables automatic deployment of such executables using the same mechanisms as updating SSH keys or adding/removing repository permissions.

5. Building Block View

5.1. Whitebox of the System

Figure 5 shows the system architecture using a C4 container diagram (Brown, 2018).

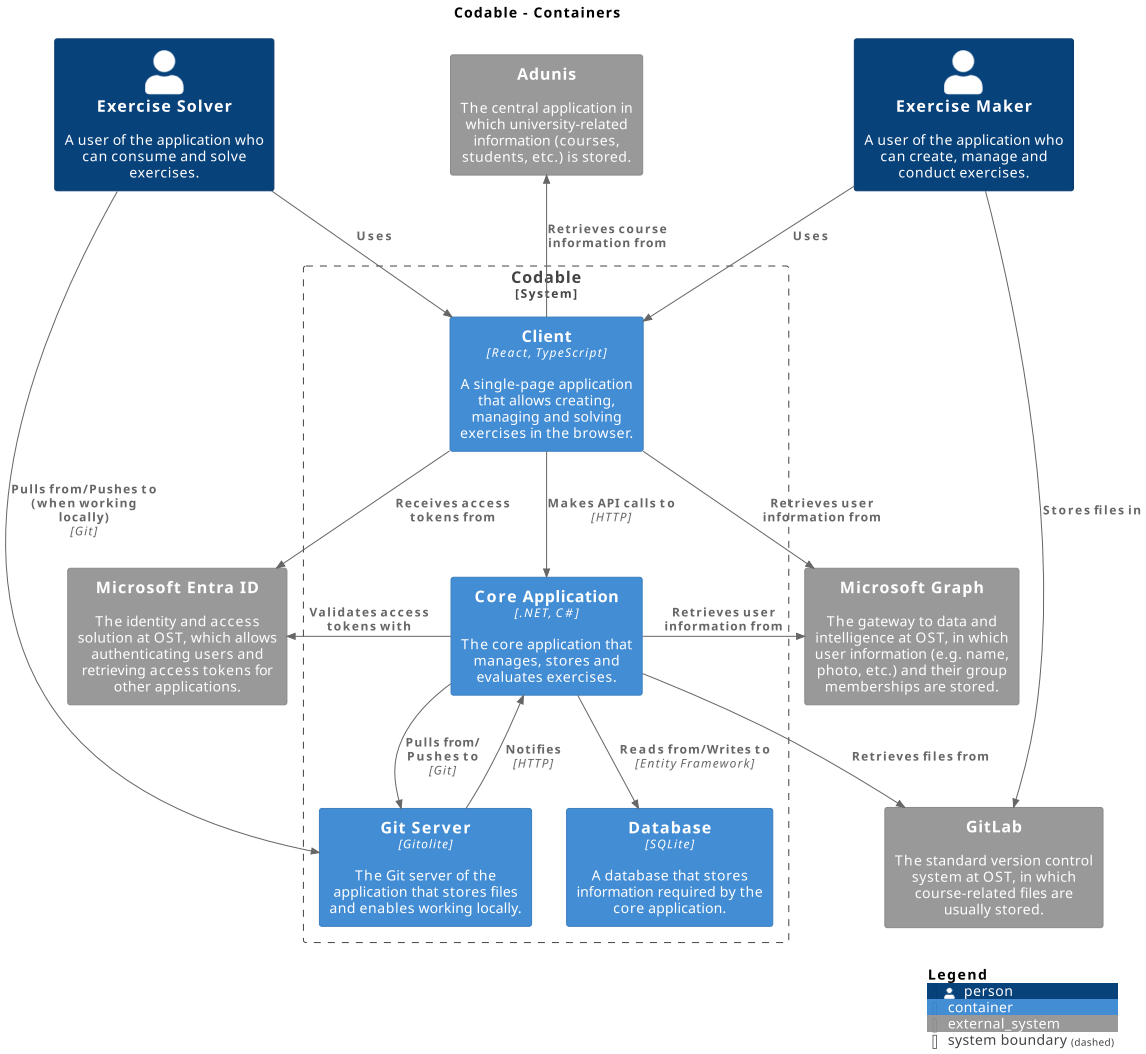


Figure 5: An overview of the system architecture

5.1.1. Rationale

Y	In the context of the decomposition of the *Codable* application, we decided on a *Three-Tiered Architecture* using the *Remote User Interface* and *Remote Database* patterns, and neglected other distribution patterns, to achieve a faster time-to-market in the early stages of application development, accepting the downside that switching to another pattern (such as *Distributed Application Kernel*) in order to address the concerns described under R1 will require additional effort in the future. (Renzel & Keller, 1997)

ADR 6: Why do we implement a three-tiered architecture?

5.1.2. Additional Remarks

Git Server: As mentioned in Section 4.3.3.1, Gitolite allows the definition of executables that can be run whenever a commit is pushed to a branch, allowing us to notify the Core Application whenever changes are made to course files. Since these executables can also receive information about which file was changed in which branch, we can clearly identify which user changed which file in which course/exercise, meaning we can run exercise evaluations whenever an exercise solver pushes something from their local machine (see Section 6.1).

These executables can also write output to the user, allowing us to provide exercise solvers a direct link to our application in their shell whenever they push any changes, simplifying the workflow and improving convenience and integration.

Since we also allow users to edit their exercises in the browser (i.e. via the Client), we can fetch, update and then push changes to Gitolite from within the Core Application whenever the Client makes an API request to update a file (see Section 6.1). This means that our Git Server always contains the most up-to-date changes of the exercise solver's files, which is why we consider it our source of truth.

Adunis: At the time of writing, Adunis is only accessed in the client to ensure that the course information (such as name, duration, etc.) does not have to be manually entered by a user. The core application has no direct relation to Adunis and functions regardless of whether an Adunis course exists or not.

5.2. Building Blocks - Level 2

5.2.1. Client (Sitemap)

Figure 6 shows the sitemap of the Client single-page application.

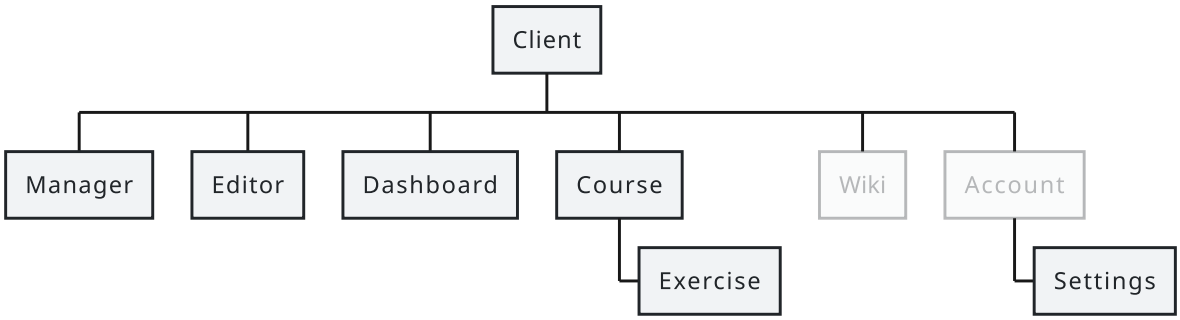


Figure 6: The sitemap of the Client single-page application

Site	Details	
Manager	**Route** `/manager`	**Users** Exercise Makers
	Description The manager is the homepage for exercise makers. It contains a list of all courses (see R3) as well as a *Wizard* for creating new courses. It is intended to serve as a *Dashboard*, with further content being added as the system is expanded (Tidwell et al., 2020).	
Editor	**Route** `/editor/{courseId}`	**Users** Exercise Makers
	Description The editor allows exercise makers to create, modify and delete the contents of a course. It is designed to resemble integrated development environments (IDEs) and game engines commonly used by software engineers. It is loosely based on the *Canvas Plus Palette* pattern (Tidwell, Brewer, & Valencia, 2020).	
Dashboard	**Route** `/dashboard`	**Users** Exercise Solvers
	Description The dashboard is the homepage for the exercise solvers. It contains a list of all courses stored within the system (see R3). It is intended to serve as a *Dashboard*, with further content being added as the system is expanded (Tidwell, Brewer, & Valencia, 2020).	

Site	Details	
Course	**Route** `/course/{courseId}`	**Users** Exercise Solvers
	Description The course contains an overview of all exercises within a course, organized into their corresponding weeks. It is designed to resemble the level screens of various video games (Strachan, n.d.) as well as being based on a didactical concept known as learning paths (de. *Lernpfade*) (Universität Zürich, n.d.).	
Exercise	**Route** `/course/{courseId}/exercise/{exerciseId}`	**Users** Exercise Solvers
	Description The exercise contains the actual contents of an exercise. As the contents of an exercise are defined through the customizable block system (see Section 8.2), this page primarily displays the render outputs of the configured blocks.	
Wiki	**Route** `/wiki`	**Users** All
	Description The wiki is intended to contain various information about the system, such as instructions for creating custom blocks and for uploading plugins. This page has not yet been implemented.	
Account	**Route** `/account`	**Users** All
	Description The account is intended to contain various personal information about the signed-in user. It may also be the place where users can personalize their profile and view their earned badges. This page has not yet been implemented.	
Settings	**Route** `/account/settings`	**Users** All
	Description The settings is the place where users can change their settings (e.g. access tokens). It makes use of the *Settings Editor* pattern (Tidwell, Brewer, & Valencia, 2020) and may require enhanced categorization if more settings are added as the system expands.	

5.2.2. Core Application (Components)

Figure 7 shows the components of the Core Application container using a C4 component diagram (Brown, 2018).

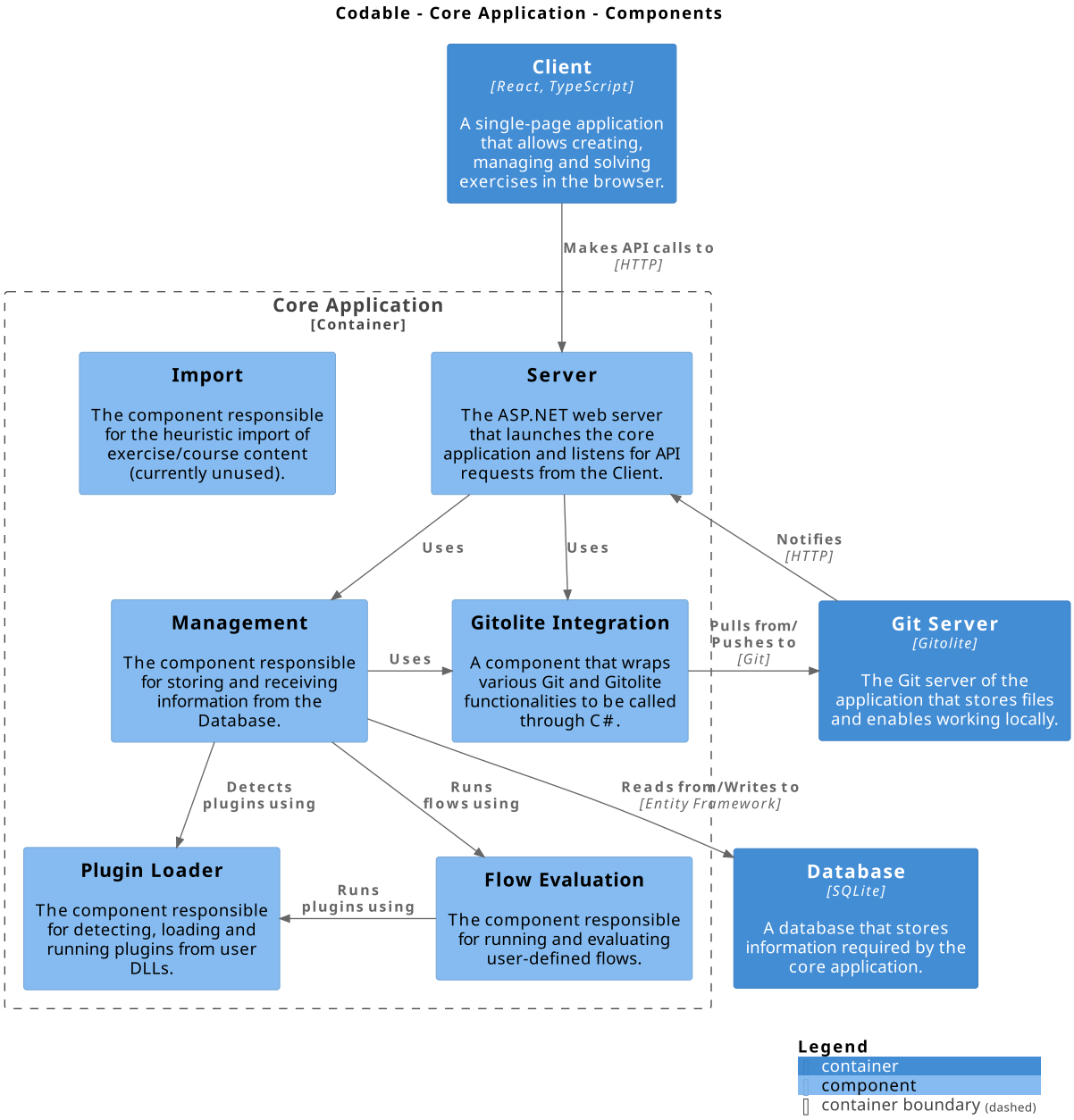


Figure 7: An overview of the Core Application container

6. Runtime View

6.1. Working on Exercises

Figure 8 to Figure 11 show different processes related to working on exercises using UML sequence diagrams (Fakhroutdinov, 2015a). To emphasize the separation of the repository storage in the Core Application's file system and Gitolite's internal storage, the *Core Application File System* has been added to the diagrams, which is not visible in other diagrams such as the system architecture in Section 5.1.

Prerequisites: For the diagrams to be valid, existing courses, exercises and flows as well as a correctly configured SSH key for local work are required.

6.1.1. Opening an Exercise

Figure 8 is only valid if no evaluation is running. Please refer to Section 6.1.4 for the sequence diagram showing an ongoing evaluation.

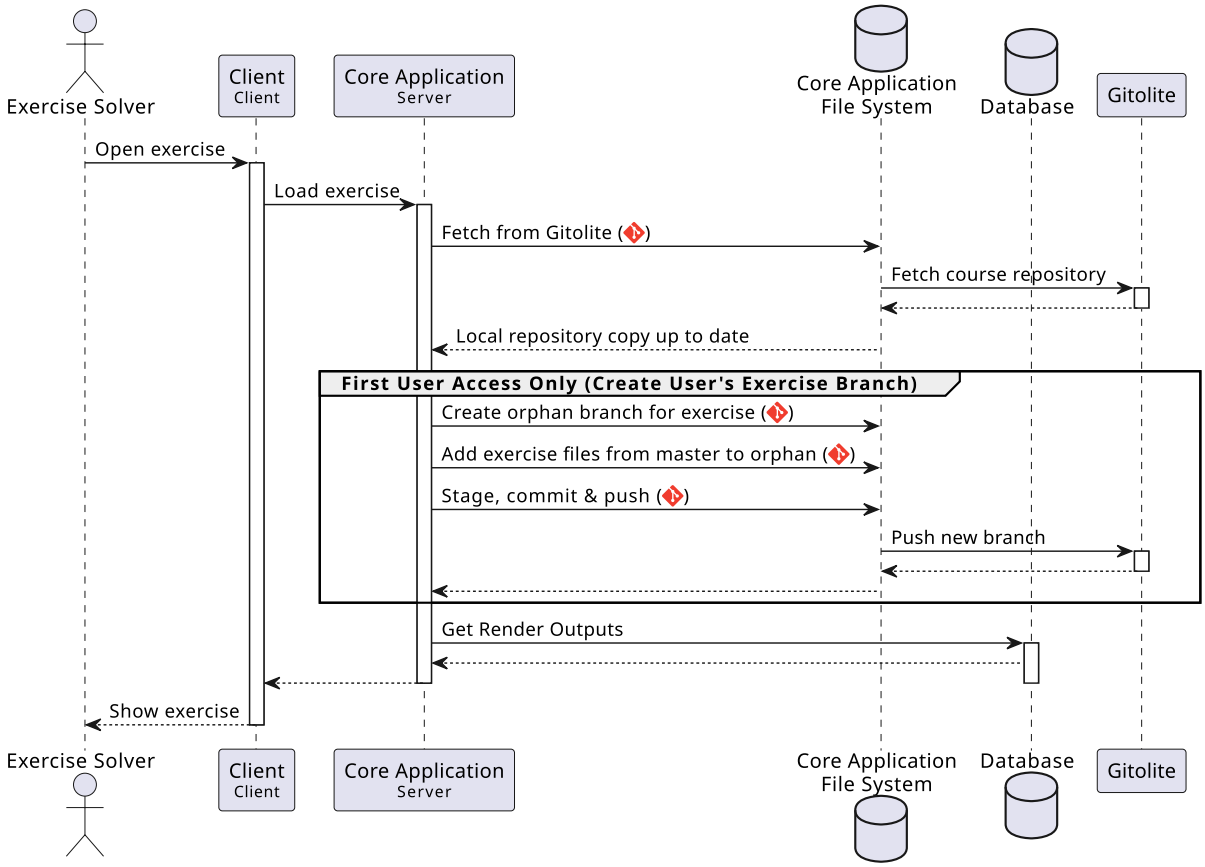


Figure 8: The process when opening an exercise

6.1.2. Editing an Exercise in the Browser

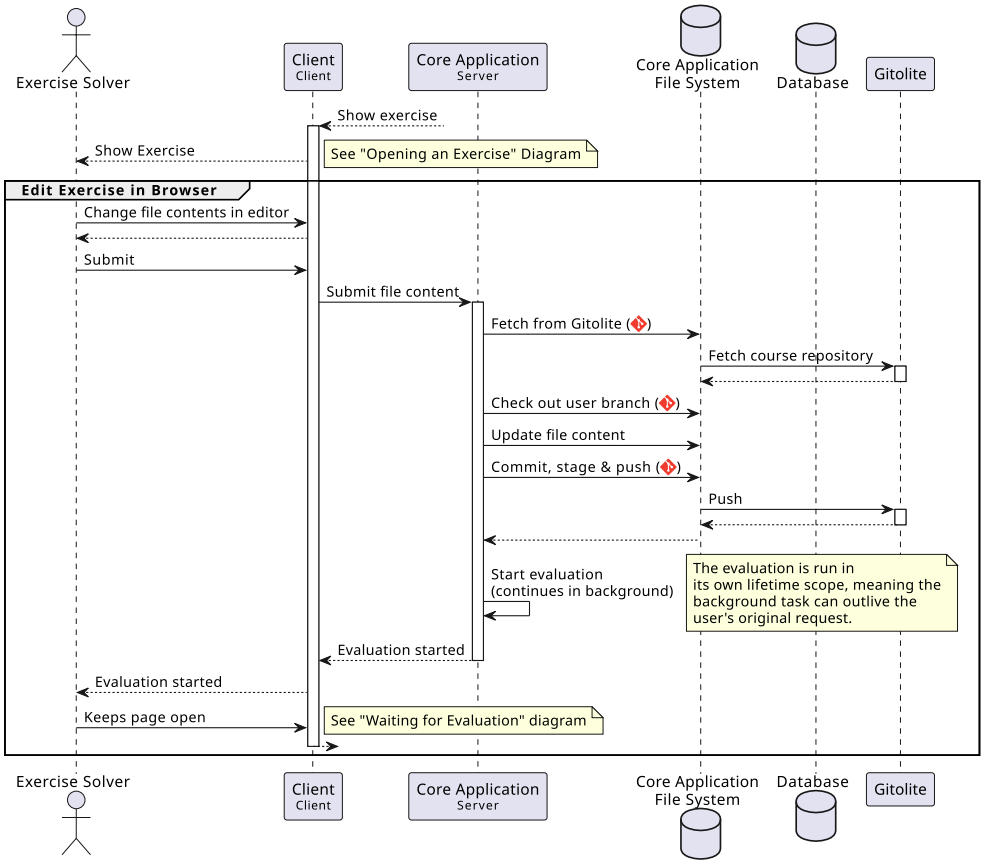


Figure 9: The process when editing an exercise in the browser

6.1.3. Editing an Exercise Locally

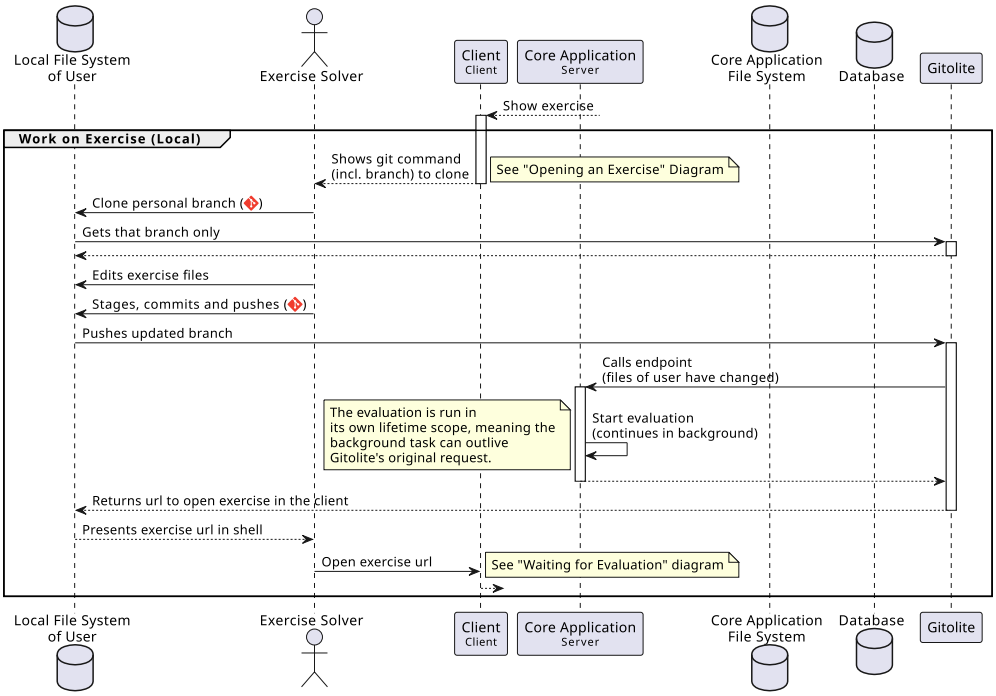


Figure 10: The process when editing an exercise locally

6.1.4. Waiting for Evaluation

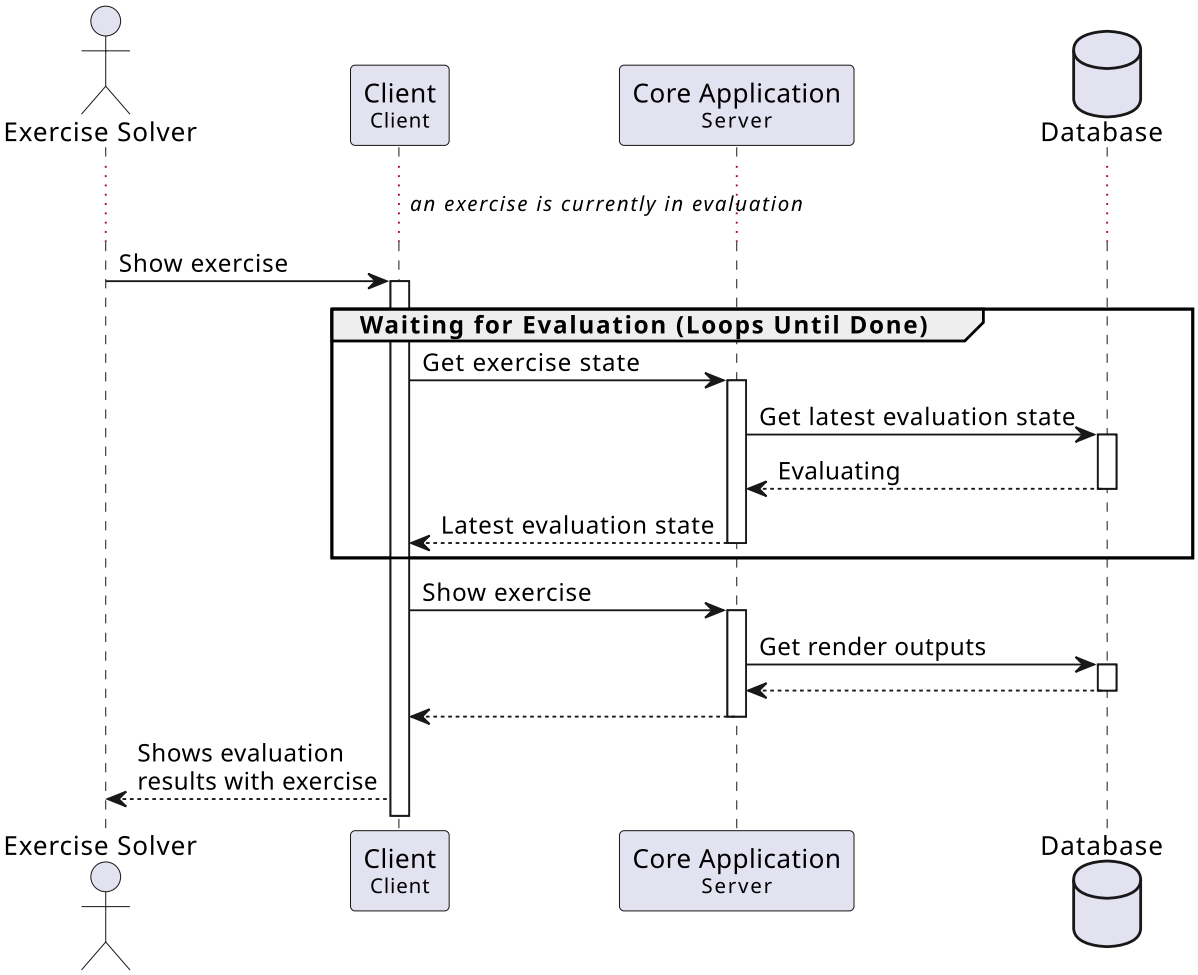


Figure 11: The process when waiting for evaluation

6.2. Authentication & Authorization

6.2.1. Sign-In Process

Figure 12 shows the sign-in process of the system, including the acquisition of access tokens and the retrieval of detailed user information (such as name, photo, etc.) from Microsoft Graph, using a UML sequence diagram (Fakhroutdinov, 2015a).

Prerequisites: For the diagram to be valid, the user must not be signed in already and none of the requested information (e.g. access token) must be cached in the browser.[7]

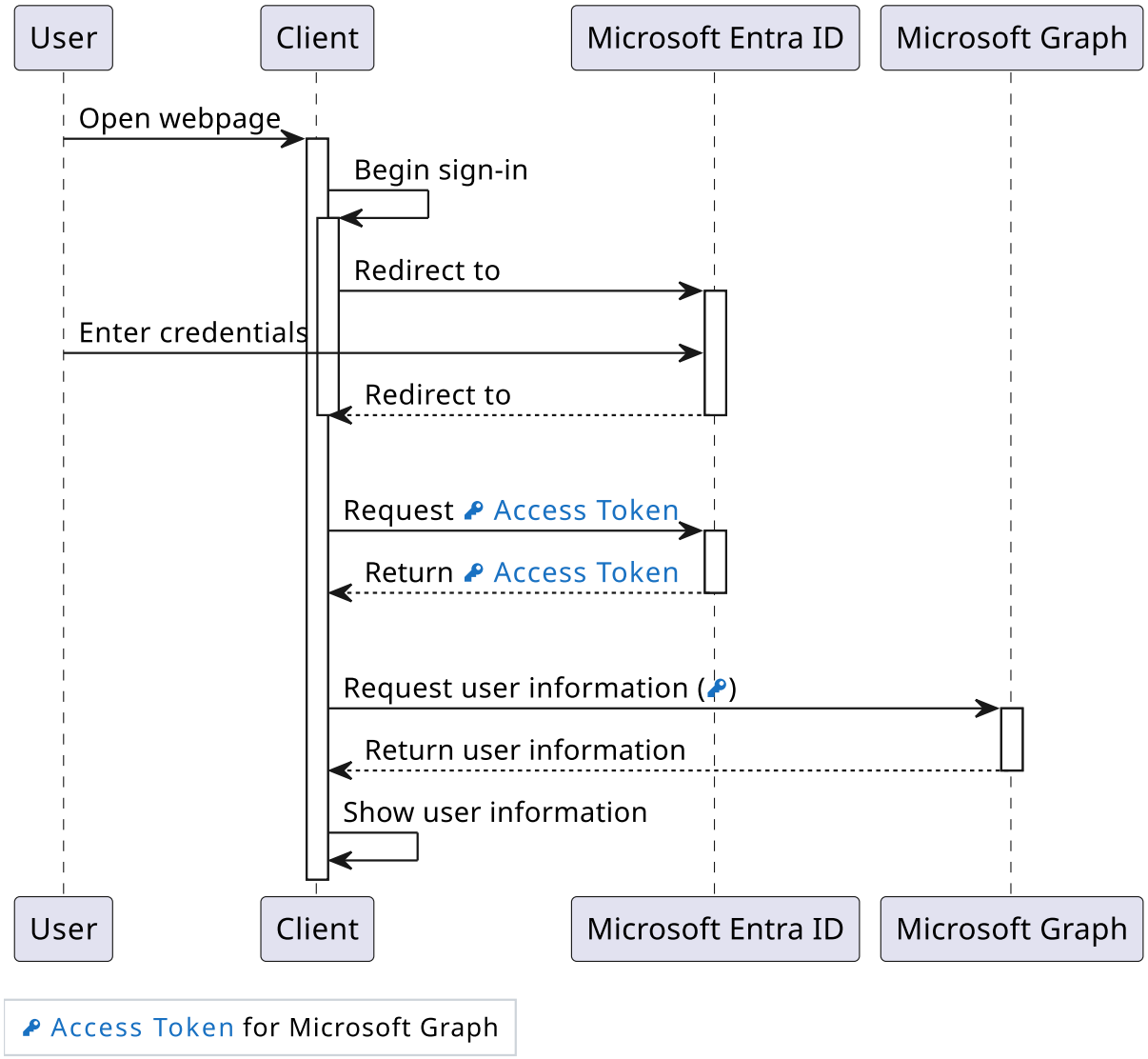


Figure 12: The sign-in process of the system

[7]Please note that caching (which is activated) causes some of the shown requests to be skipped. For example, Microsoft Entra ID is not called if a valid access token is present in the cache.

6.2.2. Authorization Process

Figure 13 shows the authorization process when calling API endpoints of the Core Application from the Client using a UML sequence diagram (Fakhroutdinov, 2015a). It also shows the difference between the different protection levels, i.e. endpoints for all users (*course reading*) versus endpoints only for exercise makers (*course creation*), which is explained in more detail in Section 8.4.2.

Prerequisites: For the diagram to be valid, the user must be signed in, must be an exercise maker (i.e. belong to the "OST Staff" group) and none of the requested information (e.g. access token) must be cached in the browser or on the server.[8]

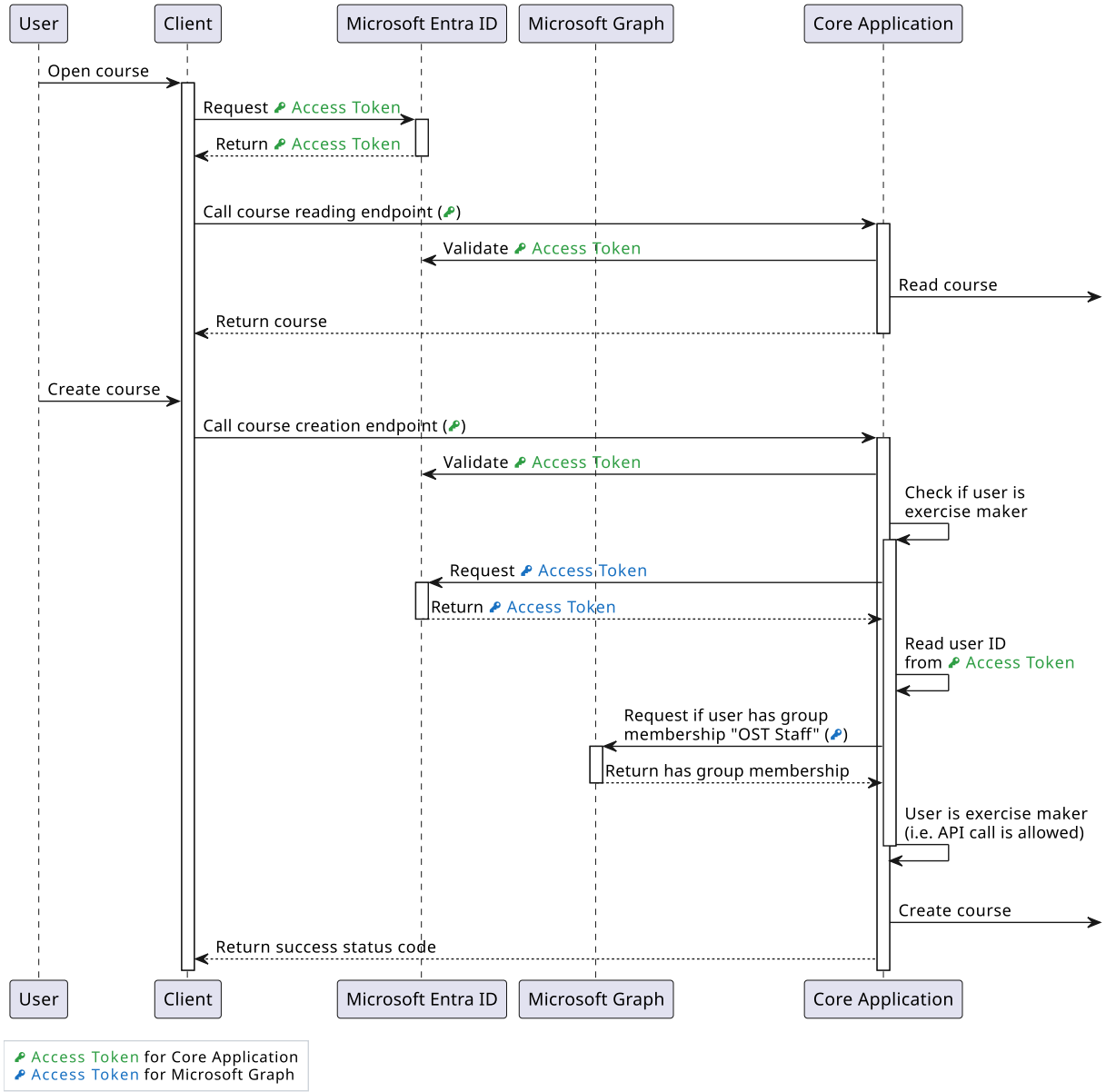


Figure 13: The authorization process when calling API endpoints

[8]Please note that caching (which is activated) causes some of the shown requests to be skipped. For example, Microsoft Entra ID is not called if a valid access token is present in the cache.

7. Deployment View

7.1. Development (Local)

Figure 14 shows the deployment of the system when working locally (i.e. in the *local development environment*) using a UML deployment diagram (Fakhroutdinov, 2015b). For instructions on how to set up your local development environment, please refer to the readme found in the internal GitLab repository.

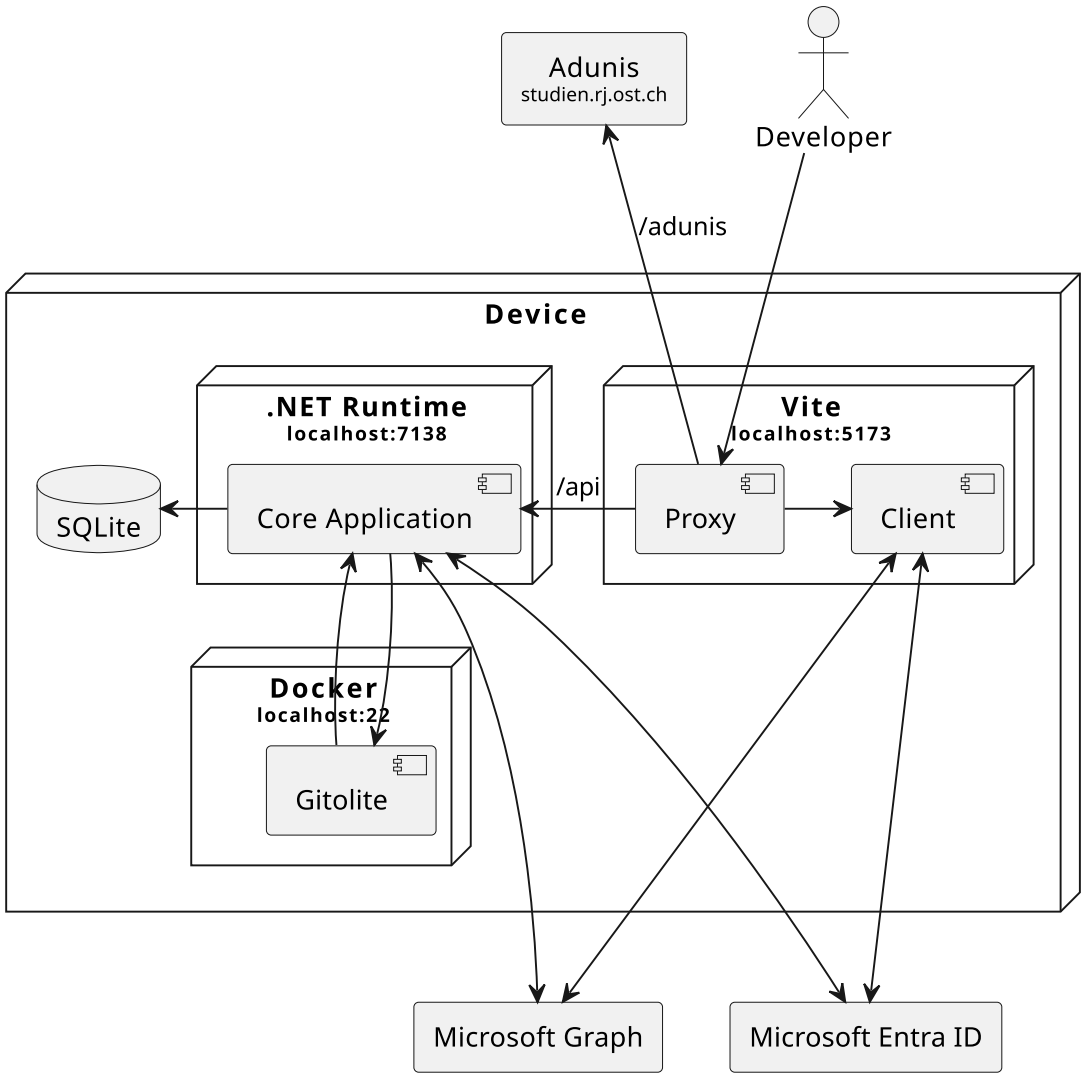


Figure 14: The deployment of the system when working locally

7.1.1. Rationale

Proxy: The main reason we are using a Vite Proxy is to enable the system to access Adunis from the Client. Adunis does not have CORS enabled on its resources, which means that these resources cannot be accessed directly from the Client running in the browser. For this reason, all requests to Adunis must be routed over a proxy.[9]

[9]We highly suspect that this is a misconfiguration on Adunis' side, as public APIs should usually have CORS enabled.

7.2. Production (Remote)

Figure 15 shows the deployment of the system on the OST Portainer instance (i.e. in the *remote development environment* and the *production environment*) using a UML deployment diagram (Fakhroutdinov, 2015b). There are currently two Portainer stacks running (i.e. `codable` and `codable-dev`) that mirror each other in their deployment, but have slight alterations in their configuration. For more information on the deployment process, please refer to Section 7.2.2.

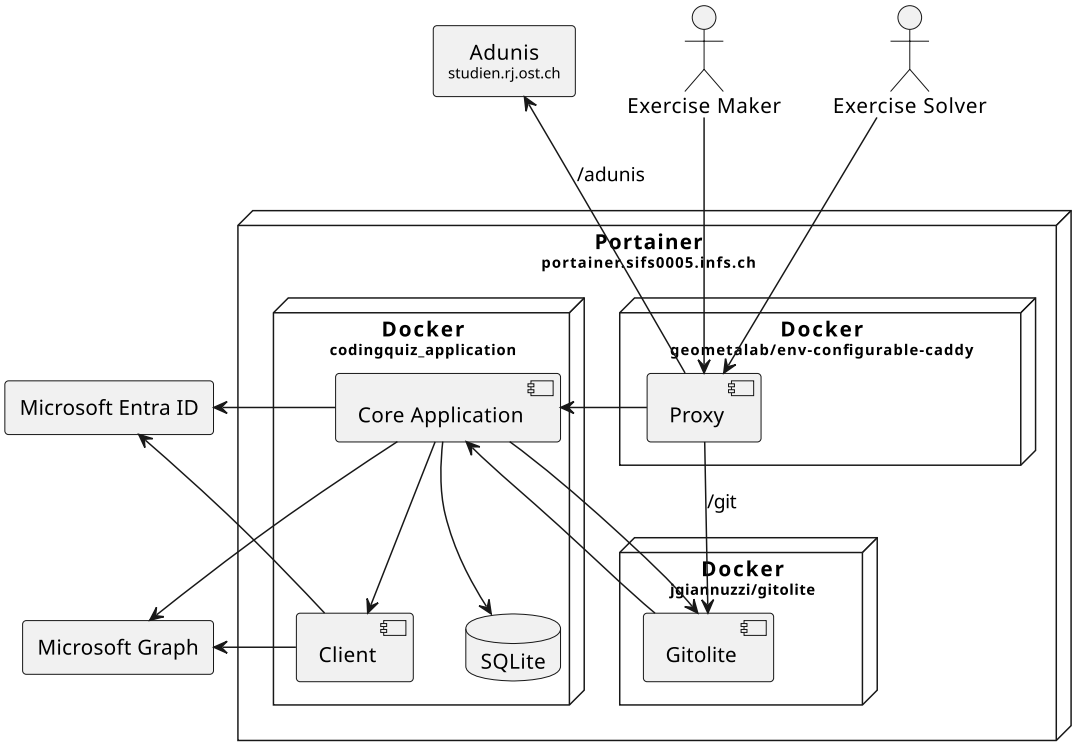


Figure 15: The deployment of the system on the OST Portainer instance

7.2.1. Rationale

Proxy: We are using a specific Caddy Proxy (`geometalab/env-configurable-caddy`) because it is the recommended image of the IFS Institute for Software to be used on the OST Portainer instance (Jordan, 2021).

7.2.2. Deployment Process

At the time of writing, the deployment process of the system is partially automatic and partially manual. Each time a new commit is pushed to either the `main` or `release` branch, a new Docker image named `codingquiz_application` with the tag `latest` or `stable` respectively is automatically created through the GitLab pipeline configured in the internal GitLab repository. Afterwards, the new Docker image must be manually pulled and restarted on the OST Portainer instance.

In general, the production environment uses the `stable` image, while the remote development environment uses the `latest` image. However, please refer to the Docker Compose (`docker-compose.yml`) files located on the OST Portainer instance for the concrete and up-to-date configuration of these two environments.

8. Crosscutting Concepts

8.1. Domain Model

Figure 16 shows the **conceptual** domain model of the system. Please note that this diagram does *not* show any classes or database entities, but instead depicts the system from a non-technical, domain-oriented perspective. If you are interested in the corresponding implementation, please generate a diagram from the code itself.

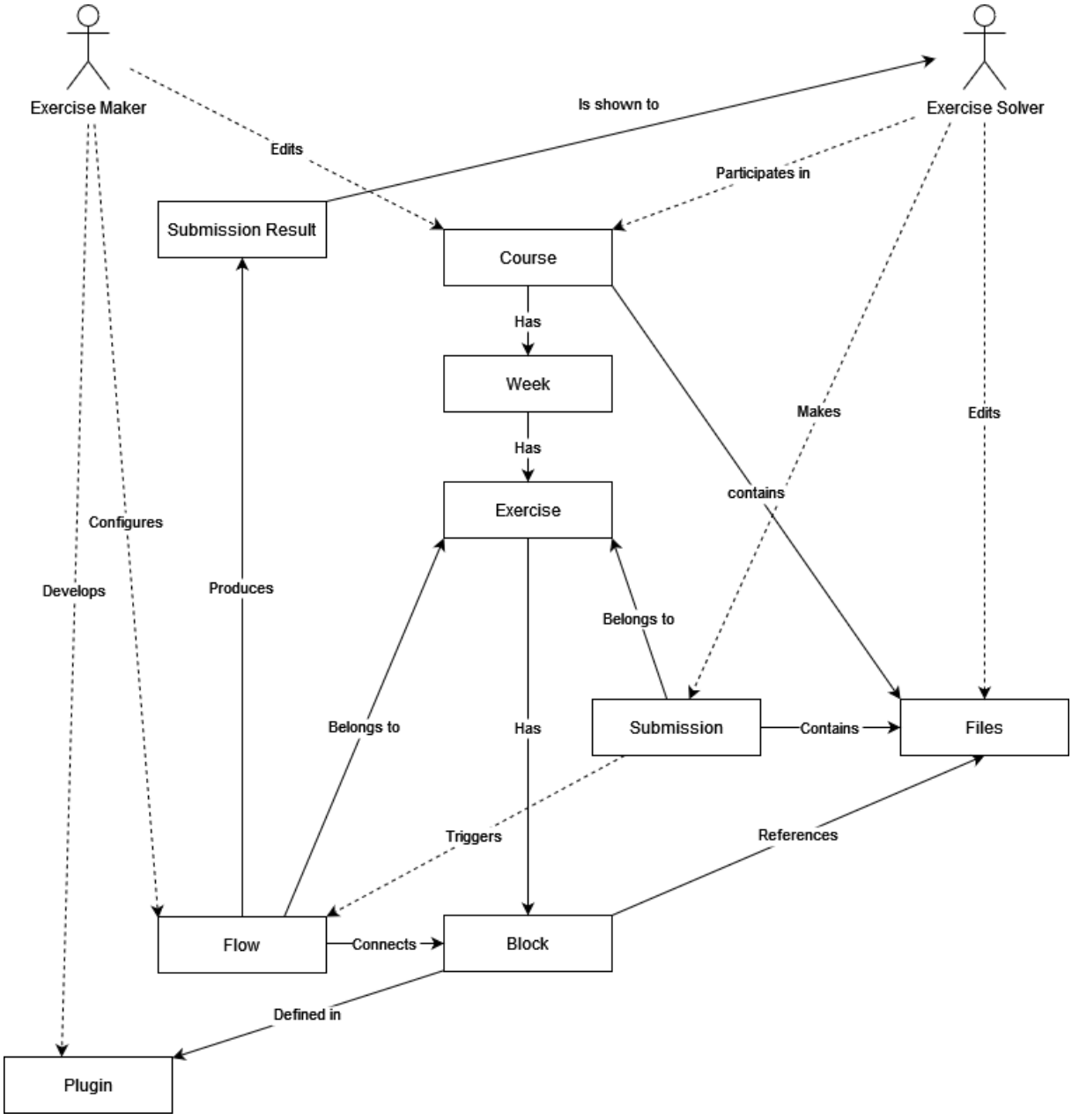


Figure 16: The conceptual domain model of the system

8.2. Plugin System

This section provides an overview over the technical design of the plugin system that allows exercise makers to provide their own evaluation and rendering logic, without directly modifying or redeploying the system.

The section uses many loaded words with regards to the plugin system. Please refer to the glossary (Section 11) for clarifications.

8.2.1. Plugin Detection

Exercise makers can upload .NET DLLs (*plugin files*) containing blocks in the course editor. These plugins are then scanned for implementations of these blocks using reflection. The detected blocks and their metadata (e.g. name, properties, flow inputs and outputs) are then stored in the database. After uploading a plugin file, the blocks are available to be assigned to exercises.

8.2.2. Plugin Loading & Evaluation

There are two scenarios when a block instance[10] is evaluated:

1. The block contains no flow inputs or outputs, but contains properties
2. The block contains at least one flow input

Semantically, a block without any inputs and properties, as well as a block that only has inputs but no outputs makes no sense.

8.2.2.1. Statically Rendered Blocks

In the first scenario, the block instance is evaluated each time a property is changed by an exercise maker. All properties must have a value for the evaluation to be possible[11]. Such blocks can be used, for example, to create an exercise description.

8.2.2.2. Dynamically Rendered Blocks

In the second scenario, the block instance is evaluated each time an exercise solver changes their files, as currently the only event causing flow evaluation is the submission of file changes by exercise solvers.

At the time of writing, a block instance can only be evaluated if all of its inputs are provided in the flow configuration. Moreover, all of its properties, if any are defined, must also be set. Outputs do not need to be connected. No validation of the connection data types is currently being done. The consequence is a runtime error during evaluation if the output connected to the block instance's input cannot be assigned, e.g. a `string` to an `int`.

8.2.2.3. Constraints

The defined flow must resemble an acyclic, directed graph in order to avoid cyclicality. The inputs of an exercise (at the time of writing only `completion`) as well as the outputs (at the time of writing only the files of the exercise solver) are not considered the same node of the graph for this purpose.

[10] *Block Instance* means that the block is assigned to an exercise. Each assignment of a *Block (Definition)* is a separate *Block Instance*. Please refer to the glossary (Section 11) for details.

[11] This is only partially precise, as default values are provided for most Primitives.

8.2.2.3.1. Branching & Merging Flows

At the time of writing, no re-merging of branched paths is possible inside the flow system. Such a feature could be implemented by deferring the evaluation of block instances that do not have all inputs available from other nodes.

8.2.3. Plugin API

The API to be implemented by plugin authors consists of two interfaces.

```
1  namespace PluginLoading;
2
3  // Internal-use only marker interface
4  public interface IBlock
5  {
6      internal Task<object?> EvaluateInternal();
7  }
8
9  // TReturnValue could be a primitive, or a complex type with annotations
10 public interface IBlock<TReturnValue> : IBlock
11 {
12     async Task<object?> IBlock.EvaluateInternal()
13         => await Evaluate();
14
15     Task<TReturnValue> Evaluate();
16 }
```

Code Snippet 1: API definition for the `IBlock` and `IBlock<T>` interfaces

The first interface is non-generic and simplifies usage inside the application, as it removes generics from the reflection code. It is *not* meant to be implemented by hand.

The second interface provides a default implementation for the non-generic interface, as well as the generic method that plugin authors have to implement: `Evaluate`. Please refer to Section III for some reference implementations.

8.2.3.1. Annotations

The API provides the annotations shown in Code Snippet 2 and Code Snippet 3 in order to register classes, parameters and properties in the plugin system.

```
1  // This namespace could be a little more descriptive and could
2  // also contain a version (e.g. Codable.Plugin.V1) to simplify
3  // the rollout of a second generation of the plugin API.
4  namespace PluginLoading;
5
6  [AttributeUsage(AttributeTargets.Parameter)]
7  public class FlowInputAttribute(string? keyOverride = null)
8    : Attribute
9  {
10     public string? KeyOverride { get; } = keyOverride;
11 }
```

Code Snippet 2: API definitions for annotations related to block construction (1/2)

```
[AttributeUsage(
  AttributeTargets.Property | AttributeTargets.ReturnValue)]
public class FlowOutputAttribute(string? keyOverride = null)
  : Attribute
{
    public string? KeyOverride { get; } = keyOverride;
}

[AttributeUsage(AttributeTargets.Parameter)]
public class BlockPropertyAttribute(string? keyOverride = null)
  : Attribute
{
    public string? KeyOverride { get; } = keyOverride;
}

[AttributeUsage(AttributeTargets.Class)]
public class BlockDisplayName(string value)
  : Attribute
{
    public string Value { get; } = value;
}
```

Code Snippet 3: API definitions for annotations related to block construction (2/2)

8.2.3.2. Render Output

Blocks can return both values to be used in flows as well as output to be rendered. Currently, Markdown, plain text, iframes and HTML are supported. The application will persist the render outputs for each evaluated block on every run, so that the block does not have to be run every time a user opens an exercise. This is important as blocks are expected to contain longer running operations such as compilation or execution of test suites. A block can return multiple render outputs by using a custom type shown in Code Snippet 4 with the appropriate annotations (Section 8.2.3.1).

```
using Funcky;
namespace PluginLoading;

// NonExhaustive to make adding new variants for the
// server a non-breaking change for plugins.
[DiscriminatedUnion(NonExhaustive = true)]
public abstract partial record RenderOutput
{
    public partial record Markdown(string Value) : RenderOutput;
    public partial record RawString(string Value) : RenderOutput;
    public partial record InlineFrame(string Url) : RenderOutput;
    public partial record Html(string Content) : RenderOutput;
}
```

Code Snippet 4: API definitions of `RenderOutput`

8.2.3.2.1. Encapsulation of Render Outputs

In order to ensure encapsulation and separation from page-level JavaScript and CSS in the Client, exercise render outputs are enclosed within a shadow DOM (MDN Contributors, 2024a). This encapsulation shields the internal structure of the Client's DOM tree and secures it from any external manipulation by the render outputs[12].

The resulting DOM structure is illustrated in Figure 17.

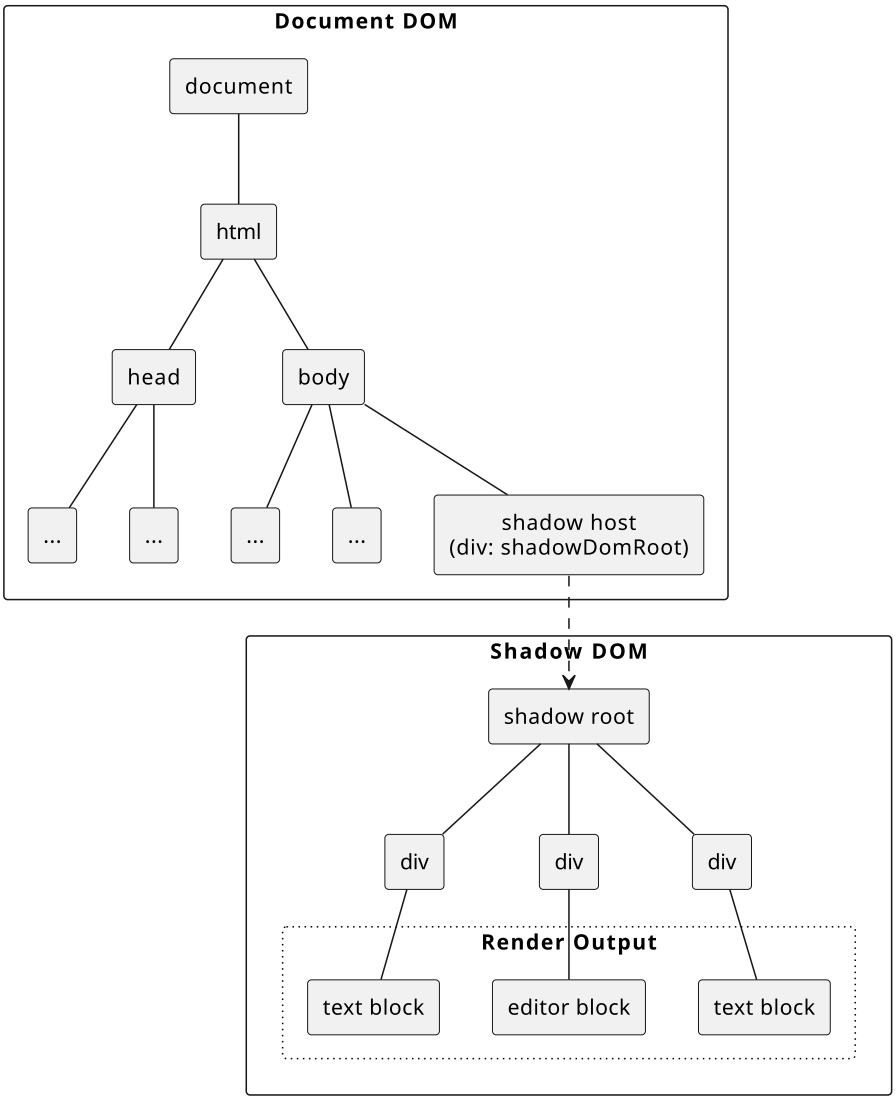


Figure 17: Encapsulation of render output using shadow DOM

The render outputs may also contain web components (MDN Contributors, 2024b) that are provided by the Client for plugin authors to use. These web components are predefined, reusable custom elements that extend the set of elements available in the browser, making it easier for exercise makers to create engaging exercise content. At the time of writing, the system only provides one web component, the `<monaco-editor>` component.

[12] Although only trusted plugin authors can create blocks that generate render outputs, we still want to protect our system as much as possible (see R2).

8.2.3.3. File Abstractions

The following section describes three different interfaces related to course files to be used by plugins. Their usage is best understood by taking a look at the reference implementations found in the appendix under Section III.

The first two interfaces are used to define properties. A reference to any file present in the course file storage can be assigned to properties of these types. In this case, `IStaticallyReferencedFile` marks a course file that **cannot** be edited by exercise solvers, while `IUserEditableFile` marks a course file that **can** be edited.

```
1  public interface IStaticallyReferencedFile
2  {
3      string RootRelativePath { get; }
4
5      string ReadAllText();
6
7      byte[] ReadAllBytes();
8  }
```

Code Snippet 5: API definition of `IStaticallyReferencedFile`

```
1  public interface IUserEditableFile
2  {
3      string RootRelativePath { get; }
4  }
```

Code Snippet 6: API definition of `IUserEditableFile`

The third interface, `IFile`, is used for flow inputs meant to be connected to the exercise's file output. The correct constructor argument would be `IEnumerable<IFile>`, annotated with `[FlowInput]`. When connected to an exercise, the flow system will pass all files configured through properties defined as `IUserEditableFile` for the exercise under evaluation. This provides access to the exercise solver's latest file contents for any given file. Please refer to Section III for a usage example.

```
1  public interface IFile
2  {
3      string RepositoryRelativePath { get; }
4
5      Task<string> ReadAllTextAsync();
6  }
```

Code Snippet 7: API definiton of `IFile`

8.3. Gamification

Figure 18 shows the motivators of the system using the Octalysis Framework (Chou, 2015). A motivator is any implicit (i.e. given by the circumstances) or explicit (i.e. specifically designed) incentive that motivates users to engage with the system. For an overview of some implicit and explicit motivators that may be utilized, please refer to the game techniques described in the Octalysis Framework.

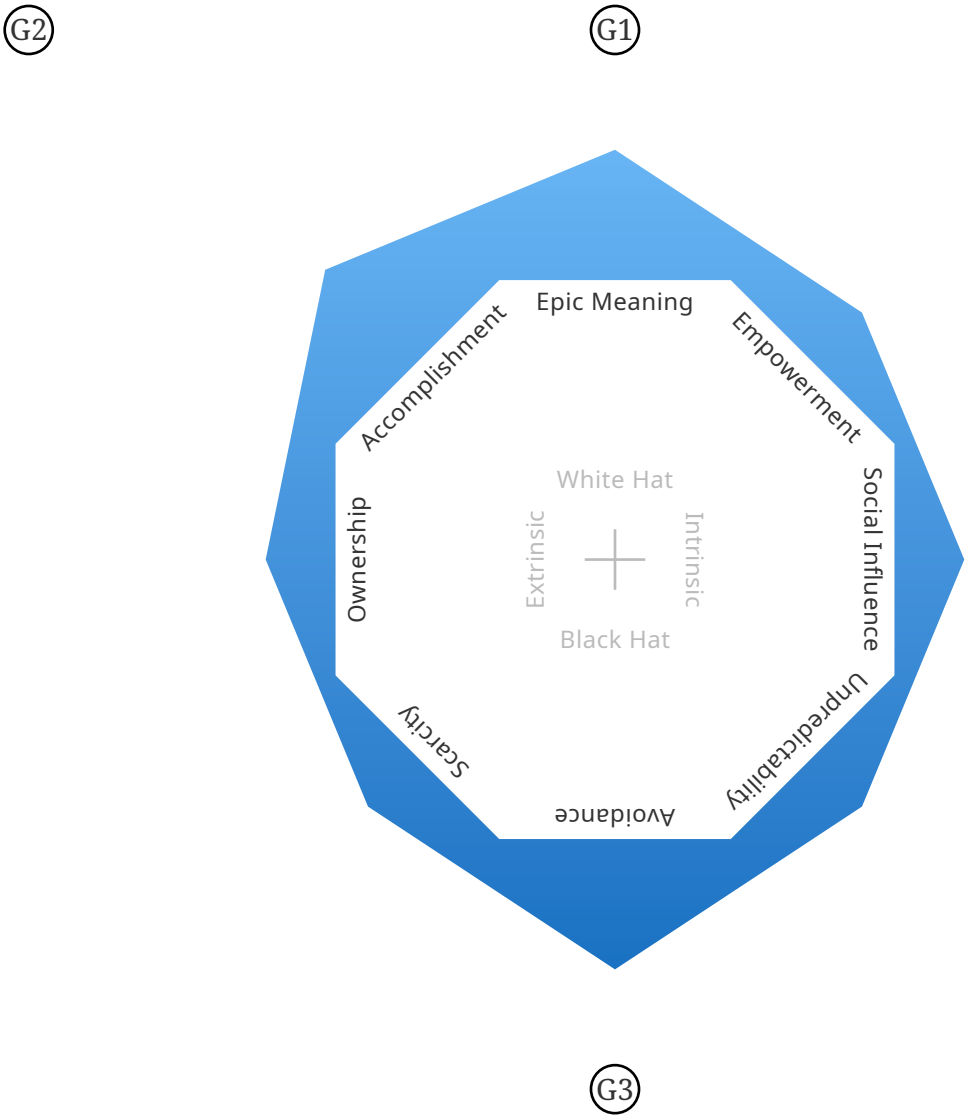


Figure 18: The motivators of the system using the Octalysis Framework

8.3.1. Epic Meaning

#	Motivator	Description
G1	Studying out of personal interest (Implicit)	An important motivator for exercise solvers is whether they are genuinely interested in the provided exercises and want to complete them for their own personal interest. As this is highly dependent on the course and the individual, we cannot directly control this motivator.

8.3.2. Accomplishment

#	Motivator	Description
G2	Progressing in the course (Explicit)	The user interface for courses has been specifically designed to resemble the level screens of various video games (Strachan, n.d.). The intention is to give exercise solvers a sense of accomplishment as they progress through the course, in addition to providing a good overview of which exercises have been completed or not.

8.3.3. Avoidance

#	Motivator	Description
G3	Avoid failing the exam (Implicit)	In contrast to motivator G1, exercise solvers may engage in exercises simply to avoid failing the final exam. This should be considered when designing features that could conflict with this motivator (e.g. time-bound exercises that may no longer be solvable during the exam phase).

8.4. Authentication & Authorization

8.4.1. Authentication

In order for the system to allow authentication using the Microsoft Entra ID single sign-on (SSO) provider, two “app registrations” had to be configured on the OST Microsoft Entra ID tenant[13]:

- **OST IFS CodingQuiz**: The app registration used by the Client
- **OST IFS CodingQuiz API**: The app registration used by the Core Application

These app registrations are required for users to sign in with their OST Microsoft accounts and in order to obtain and validate access tokens in the Client and Core Application. Figure 19 shows the current configuration in regards to the access tokens and their scopes. However, for the concrete and up-to-date configuration, please refer to the actual configuration in the OST Microsoft Entra ID tenant.

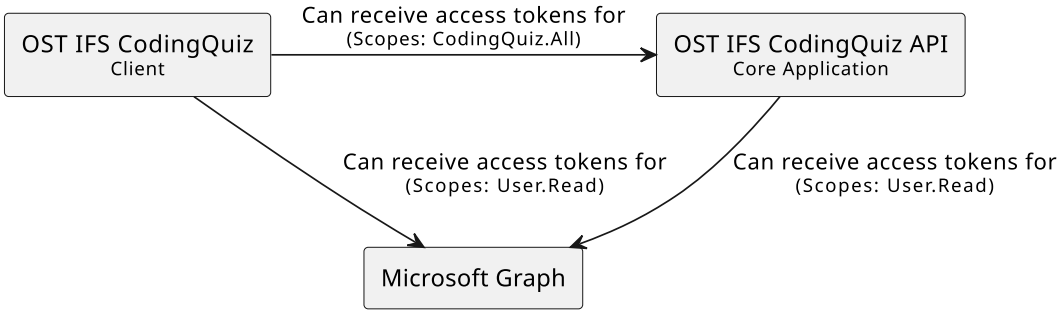


Figure 19: An overview of the Microsoft Entra ID access token scopes

[13] Access to the OST Microsoft Entra ID tenant must be requested from the OST ICT department.

While there are many examples and explanations in various qualities on how to set up and configure single sign-on using Microsoft Entra ID (especially in regards to the libraries used in this system, as described in Section 4.3), we recommend the examples “Calling a protected web API” (Erişen, 2023a), “Call Microsoft Graph (JavaScript)” (Erişen, 2023b) and “Call Microsoft Graph (.NET)” (Microsoft, 2023), on which this configuration is largely based on.

8.4.2. Authorization

At the time of writing, the Core Application uses user information accessible via Microsoft Graph to restrict certain API calls based on the user’s group membership. This means that API functions intended only for exercise makers (such as creating or updating courses) can only be performed by users belonging to the “OST Staff” group.[14] All other functions (such as reading or solving courses) are accessible to all signed-in users.

There is currently no per-course access management set up, which means that all exercise makers can edit all courses and that all exercise solvers can read and solve all courses. This is not the intended final behavior and should be changed if the system is to be used productively by multiple exercise makers in the future (R3).

For an example of the authorization process, please refer to Section 6.2.2.

8.5. Testing

8.5.1. Automatic Testing

The system undergoes automatic testing using technology-appropriate frameworks to ensure the continuous functioning of its features. Since no testing requirements are imposed by any of the stakeholders, no specifications for aspects such as test coverage or similar are defined to avoid arbitrary or unreasonable quality standards. The development team is encouraged to test their functionalities in accordance with common industry standards.

8.5.2. Usability Testing

As intended by the user-centered design process (International Organization for Standardization, 2019), regular usability tests are conducted to ensure the system meets the expectations and requirements of the targeted user base. Suitable scenarios for both exercise makers and exercise solvers, as well as protocols of conducted usability tests, can be found in the appendix under Section V.

[14] The “OST Staff” group was chosen because it includes all users who might be interested in trying out the system. Since all OST employees are contractually obligated to not intentionally harm the organization, we consider such an access range reasonable.

9. Quality Requirements

9.1. Quality Attributes

Table 6 shows the quality attributes of the system. Quality attributes should be specific in their scope (see S.M.A.R.T. (Doran, 1981)). The categorization follows the ISO/IEC 25010 standard (ISO/IEC, 2023).[15]

#	Specification	Scenarios
QA1	The specific content or logic of exercises must be highly modifiable by exercise makers.	QAS1
QA2	The modification of courses and exercises must be completable with a minimal amount of manual effort.	QAS2
QA3	The user interface for exercise solvers must be consistent across different courses.	QAS3
QA4	The user interface must be accessible in accordance with the OST recommendations (Loch & Stolze, 2022).	QAS4
QA5	The disk space required for course data must be minimized to ensure scalability for large numbers of users.	QAS5, QAS6

Table 6: The quality attributes of the system

9.2. Quality Attribute Scenarios

Table 7 shows the quality attribute scenarios of the system. Quality attribute scenarios must be specific and measurable in order to determine whether a quality attribute is met or not (see S.M.A.R.T. (Doran, 1981)). Since this is not a legal document, terms such as “all” or “most” are sufficient for the development team to assess the fulfillment of a quality attribute scenario.[16]

#	Scenario	Fulfilled?
QAS1	Changes to the content or logic of exercises can be made without having to change or redeploy the system.	Yes
QAS2	All existing or derivable information is added automatically when courses or exercises are created.	Mostly[17]
QAS3	All courses require equivalent actions to complete the user scenarios as defined in Section V.B.b (excluding the actual exercise assignment itself).	Yes

[15] The categorization is currently omitted, as it would introduce unnecessary clutter into this (currently quite short) table. The categories are therefore written as comments in the source code.

[16] The scenarios were last validated on the 05.06.2024 in a peer review by the development team.

#	Scenario	Fulfilled?
QAS4	All user interfaces for exercise makers and exercise solvers have a Lighthouse Accessibility score of above 90.	No[18]
QAS5	Redundant exercise data (e.g. unchanged files) is not stored to minimize disk space usage.	Yes
QAS6	Exercises data is only instantiated when progress is submitted by exercise solvers. Since not all exercise solvers will complete their exercises, disk space should not be wasted unnecessarily.	Yes

Table 7: The quality attribute scenarios of the system

[17]We automatically integrate all possible external dependencies such as Adunis and Gitlab. The heuristic import of courses has been deprioritised as explained in the project management section.

[18]During development, we focused on some accessibility issues, such as tab indexes and contrast ratios. Extensive lighthouse testing has not been done due to time constraints. As *Codable* moves forward, it would be beneficial to perform these tests thoroughly.

10. Risks and Technical Debt

10.1. Technical Risks

Table 8 shows the accepted technical risks of the system.

#	Risk	Description
R1	Dependence on core application	The system relies on a single core application for all core logic and evaluation. If this application fails, the entire software becomes unusable. This approach simplifies implementation but creates a single point of failure. Future versions may explore distributed evaluation to improve reliability.
R2	Trust in plugin authors	The system includes a plugin system that allows exercise makers to create .NET DLLs for integration. This design enhances extensibility, but also introduces risk, as we must trust these authors to use all .NET Runtime features responsibly and securely.
R3	No per-course access management	Currently, exercise makers and exercise solvers have access to all existing courses. Exercise solvers should be restricted to the courses they belong to and exercise makers should be restricted to specified courses.
R4	Inconsistent client-server synchronisation	Client and server updates are inconsistent due to the lack of bi-directional synchronisation. For example, when code is submitted via an editor block or when changes are committed locally, evaluations are triggered, but the UI is not notified. Instead, only the evaluation status is updated via a polling mechanism, which is suboptimal. Given that an exercise solver can restore accidentally overwritten submissions using git, this is a minor issue as there is no permanent loss of data. Future releases will require consistent bi-directional synchronisation to address this issue.

Table 8: The technical risks of the system

10.2. Business & Domain Risks

Table 9 shows the business and domain risks of the system.

#	Risk	Description
R5	Productive use of the application	Despite a comprehensive user need assessment conducted during the semester thesis, the further development and productive use of the application depends on convincing both the exercise makers and the stakeholders at the Department of Computer Science about the capabilities of the system.
R6	Exercise Quality	Even with the implementation of the quality goals in Section 9, the exercise makers are still responsible for maintaining the quality of the exercise content.

Table 9: The business and domain risks of the application

11. Glossary

11.1. Terms

Term	Stands for
Block (Definition)	A unit of logic, which can be assigned to exercises. This assignment to an exercise creates a Block Instance based on the Block Definition.
Block Instance	May be parameterized by exercise makers through Properties, and can be evaluated statically (once after any Property is changed), if no inputs exist (Statically Rendered Block), or dynamically (meaning once per user submission) through the *flow* system, if dependent on other blocks.
Connection	A directed connection from a (Flow) Output to a (Flow) Input.
Evaluation	The process of running a Block Instance with the configured inputs, generating output (which can be Render Output or Primitives to be used in connected blocks).
Evaluation Triggers	Evaluation can be triggered for a Block Instance without any (Flow) Input when the value for a Property is changed by an exercise maker, or for a whole exercise, when any new value for the exercise's inherent outputs (currently only *user files*) triggering the evaluation of the whole flow, following configured Connections, evaluating all Block Instances on the way.
(Exercise) Flow	Many connected Flow Nodes. To allow evaluation, a flow definition must be a acyclic & directed graph to avoid infinite recursion. Remark: The Exercise Node itself does not break the acyclic graph requirement, as it is interpreted as a set of outputs (value providers) which start the graph, and a set of inputs which, if connected to, define the end of the graph, leading to no additional events that may cause further evaluation of other blocks. This also means that an exercises completion can not be used as output of the exercise, as it must be settable through an input.
Exercise Maker	A user who creates, manages and conducts exercises (e.g. a lecturer or assistant).
Exercise Solver	A user who consumes and solves exercises (e.g. a student).

Term	Stands for
(Flow) Input	A Block (Definition) may contain inputs, for which a value will be provided during the evaluation. To provide the source of the input value, an exercise maker must connect the output of a Block Instance or the output of an exercise.
Flow Node	A Flow Node provides Flow Inputs and Flow Outputs. Each (Exercise) Flow contains exactly one exercise providing its inherent in- and outputs, as well as any number of blocks instances connected to those. Remark: Block instances that do not contain any inputs may not contain any flow outputs and are also not viable to be used in flows.
(Flow) Output	A Block (Definition) may contain outputs, which the Block Instance will have to return after being called with all required inputs and properties during evaluation. Outputs can be connected to other inputs as part of the flow, or provide concrete values for inherent properties of exercises, e.g. the exercise solvers completion of an exercise in percent.
Plugin	A collection of features that can be developed by exercise makers using the defined APIs. A plugin may contain many Block Definitions.
Primitives	Explicitly supported types for the flow system. At the time of writing, the following .NET types are supported: `bool`, `string`, `int` and `decimal`.[19]
Property	A Block (Definition) may contain Properties, for which a value will be provided during the evaluation. A Block Instance can have its Properties set by exercise makers. Properties may only be Primitives.
Render Output	A Block (Definition) may contain Render Outputs, which will be stored after evaluation and used to render the exercise for the user. For example, a Block could return an HTML file that lists the passed and failed test cases to the exercise solver after his submission. The user will only be shown the latest Render Output for each Block Instance, and the render outputs of Statically Rendered Blocks are shared across all exercise solvers.
Statically Rendered Block	As mentioned in the definition of Evaluation Triggers, if the Block contains no Flow Inputs (Properties are allowed), it will be evaluated only if properties change. Therefore, Render Outputs for such “static” Blocks are the same for all

Term	Stands for
	users, allowing the application to store the result without any association to a user.
Submission	An exercise solver updates exercise files and submits them for evaluation.
User File	A file, exlicitly part of an exercise, which can be edited by an exercise solver.

Table 10: The domain-specific terms used in the system

11.2. Abbreviations

Abbr.	Stands for
ADR	Architectural Decision Record
API	Application Programming Interface
Abbr.	Abbreviation
CORS	Cross-Origin Resource Sharing
DLL	Dynamic Link Library (.NET)
DOM	Document Object Model
EF	Entity Framework
IDE	Integrated Development Environment
IoC	Inversion of Control
MSAL	Microsoft Authentication Library
ORM	Object-Relational Mapping *or* Object Relational Mapper
SSH	Secure Shell
SSO	Single Sign-On
UI	User Interface
UML	Unified Modeling Language
de.	Deutsch (*German*)
e.g.	Exempli Gratia (*for example*)
i.e.	Id Est (*that is*)

Table 11: The abbreviations used in this document or in the system

[19] These are C# aliases for `System.Boolean`, `System.String`, `System.Int32` and `System.Decimal`. Additional types (e.g. Collections, Doubles, etc.) are planned.

Project Documentation

12. Assignment

12.1. Context

The subsequent sections are based on the original assignment of this Bachelor's thesis, which can be found in the appendix under Section I (written in German).

Gamification describes the use of game-like elements in a non-gaming environment. This allows certain applications to be made more motivational. This Bachelor's thesis builds upon a previous project, in which the current practice in the organization of lectures and, in particular, their exercises was analyzed. The project has shown that a more holistic approach needs to be pursued in order to motivate students to do their exercises more consistently, which also includes organizational aspects rather than just gamification (Messmer & Fischler, 2023). The existing results are to be implemented, expanded and evaluated as part of this Bachelor's thesis.

12.2. Concrete Assignment

The Bachelor's thesis should address the following aspects. Deviations from the described assignments are possible on agreement during the course of the thesis.

1. The requirements and needs of future users are to be analyzed on the basis of existing applications.
2. Existing applications with comparable functionalities are to be considered in order to differentiate them from the objective of this thesis. Relevant sources from scientific literature are to be analyzed and included in the solution identification.
3. Requirements are to be formulated on which a comprehensive technology selection can be made. Free and open source technologies should be preferred.
4. A human-centered approach is to be pursued in the design of all user interface components. This includes, for example, the development and evaluation of conceptualizations in cooperation with real users.
5. The chosen approach is to be prototypically implemented. The software architecture and its outcomes must be documented in detail. An implementation is to be tested with potential users in a realistic environment.
6. The chosen approach is to be critically reflected upon. Concrete suggestions for improving the process and the results are to be discussed.

Additionally, a suitable project management method including work packages and an appropriate number of milestones must be documented and regularly updated during the project. Working hours must be tracked at a work package level.

12.3. Formalities

This Bachelor's thesis will be rewarded with 12 ECTS upon successful completion. The expected workload is 360 hours distributed over 17 weeks (~21h/week). The Bachelor's thesis starts on **Monday, February 19, 2024** and ends on **Friday, June 14, 2024 at 17:00**. Meetings with the supervisor are held bi-weekly unless otherwise agreed.

All results of this Bachelor's thesis (e.g. source code, documentation, etc.) must be made available to the project participants for further use. The documentation is expected to meet common quality standards for scientific papers. All references to external sources must be cited in the bibliography using the APA 7 style. All written texts must follow the Guideline for Gender-Sensitive Wording of the Swiss Confederation (Bundeskanzlei BK, 2023) where applicable.

13. Project Management

13.1. Goals

As described in the assignment in Section 12, the purpose of this Bachelor's thesis is to implement, expand and evaluate the results that were researched in the 2023 semester thesis written by two members of this project. As such, our main goal is to implement the functional requirements defined in the section "Minimal Viable Product" of the semester thesis (Messmer & Fischler, 2023, pp. 40-42).

13.1.1. Must-Have Requirements

Figure 20 and Figure 21 show the must-have requirements of this Bachelor's thesis as defined in the preceding semester thesis. For each requirement, it is indicated whether it has been implemented (✓), partially implemented (~) or not implemented (✗), with additional information provided below.

IID	Requirement	
112	The application contains a course selection.	✓
↳ **111**	The course selection shows all courses a user belongs to.	~ 1)
113	The application contains an exercise selection.	✓
↳ **114**	The exercise selection shows all exercises of a course.	✓
↳ **115**	Each exercise has a progress bar.	✓
116	The application contains an exercise view.	✓
↳ **117**	The exercise view shows the contents of an exercise.	✓
↳ **118**	An exercise can contain an online editor for coding tasks.	✓ 2)
↳ **119**	An exercise can contain a description.	✓ 2)
↳ **120**	An exercise allows the submission of a solved coding task.	✓
123	The application allows working locally using Git.	✓
↳ **124**	A course can be cloned as a Git repository.	~ 3)
↳ **125**	An exercise can be solved on a separate branch.	~ 4)
126	The application allows CRUD on courses and exercises.	✓
↳ **127**	A new/empty course can be created.	✓
↳ **128**	A new/empty exercise can be created.	✓
↳ **129**	An exercise can be edited.	✓
↳ **130**	An exercise can be restructured in its grouping.	✗ 5)
↳ **131**	An exercise can be restructured in its order.	✗ 5)
133	The application allows importing courses and exercises.	✓
↳ **134**	An existing course can be imported from GitLab.	✓
↳ **135**	An existing exercise can be imported from an archive or folder.	✗ 6)

Figure 20: The must-have requirements of this Bachelor's thesis (1/2)

↳	**136** The title of a course is determined heuristically.	✘ 7)
↳	**137** The weekly structure of a course is determined heuristically.	✘ 7)
↳	**138** The description of an exercise is determined heuristically.	✘ 7)
↳	**139** The coding files of an exercise are determined heuristically.	✘ 7)
↳	**140** The order of an exercise is determined heuristically.	✘ 7)
	144 The application supports OAuth integration.	✔
↳	**145** A user can log in using OAuth.	✔
↳	**146** The role of a user is automatically determined using OAuth.	✔
↳	**147** A course can only be edited by its owner.	~ 8)

Figure 21: The must-have requirements of this Bachelor's thesis (2/2)

Details & Rationale

1. All courses are currently shown in the course selection, since per-course access management has not been set up yet (R3).
2. This is possible with the plugin system.
3. All exercises are currently cloned individually. However, with the chosen technology (Gitolite), such an implementation would be possible.
4. Each exercise currently exists on its own branch within the same repository. However, with Gitolite, additional branches would be possible.
5. This feature has been disregarded due to time constraints and prioritization.
6. It is currently only possible to import courses (i.e. course files) from GitLab. Since course files are managed in a dedicated repository within our application (Gitolite), it would be possible to allow folders or archives if the need arises.
7. All features in relation to the heuristic import of courses have been deprioritized to make room for the plugin system, which we considered to be more beneficial and architecturally significant for the system.
8. Although the owner of a course is stored within the application, it is currently possible for all exercise makers to edit all courses (R3).

13.1.2. Should-Have Requirements

We have decided to not implement any of the should-have requirements defined in the preceding semester thesis (Messmer & Fischler, 2023, p. 42). This was done in order to devote more time to the qualitative aspects of both the implementation and documentation of the must-have requirements.

13.2. Approach

As our team consists only of three members, we decided to keep our project management approach simple in order to reduce administrative overhead. Our chosen approach is broadly inspired by iterative and agile process frameworks such as Scrum (Schwaber & Sutherland, 2020) and the Agile Manifesto (Beck et al., 2001). We also follow the principles of the user-centered design process (International Organization for Standardization, 2019).

We use Miro to manage and plan our project. To ensure that we are on track with our upcoming milestones, internal team meetings are held at least once a week (*weekly*), usually on Wednesdays. For each milestone, we hold a short review and retrospective. In addition, bi-weekly meetings are held with the thesis advisor and thesis expert to ensure that the project is on track. To avoid formalities, we do not impose any meeting guidelines in terms of both form or time.

13.3. Planning

13.3.1. Project Plan

Our project plan has been carried over and adapted from our experiences in the semester thesis. As seen in Figure 22, the project is divided into four blocks of three weeks for the user-centered iterations and six blocks of two weeks for the technical iterations. At the end of each user-centered iteration, a milestone is located. The thesis is flanked by the kickoff (week 1-2) and the submission (week 15-17). During those weeks, we are focused on planning, documenting and consolidating the project.

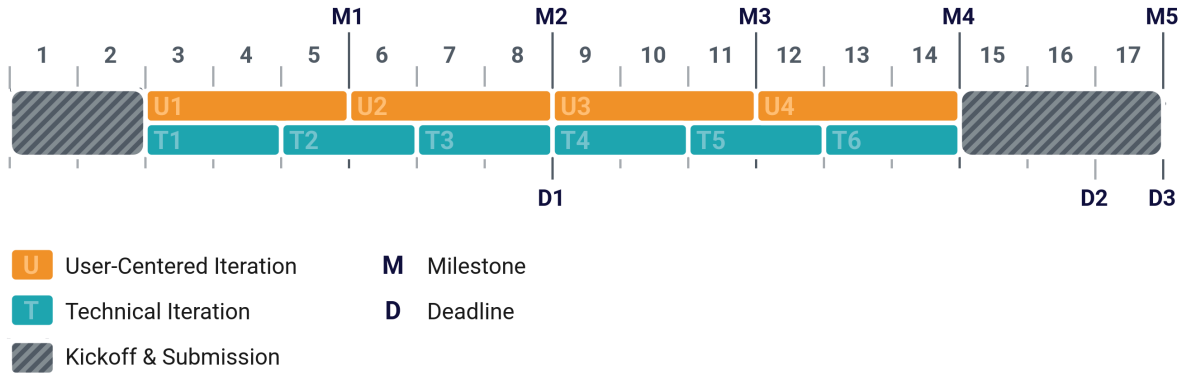


Figure 22: An overview of the iterations and milestones within this thesis

13.3.2. Deadlines

There are 3 main deadlines that are relevant for this Bachelor's thesis:

D1 The mid-term presentation of the thesis (*Fri. April 12*)
D2 The submission of the abstract (*Mon. June 10*) and poster (*Wed. June 12*)
D3 The final submission of the thesis (*Fri. June 14*)

Since the final presentation (on Thursday, July 4) of the thesis does not fall within the official time frame of this project, it has been omitted from our project plan.

13.3.3. Milestones

All our milestones are roughly planned at the beginning of the Bachelor's thesis and then concretized at the end of each preceding iteration. At the end of a milestone, a review and retrospective takes place in order to validate the completeness of our defined goals. The progress of each milestone is communicated to the thesis advisor and the thesis expert at the bi-weekly meetings.

Table 12 shows the milestones we have defined for our Bachelor's thesis, including important notes and changes that we discussed in our reviews during the course of the project. For each defined goal, we have indicated whether it has been completed (✓), partially completed (~) or not completed (✗) in the respective iteration.

#	Details	
M1	**Goals (Initial)** • U126 (Course CRUD) • U133 (Import)	**Goals (End of Kickoff)** • U151 (Course CRUD) ✓ • U155 (Week CRUD) ✓ • U160 (Exercise CRUD) ✓ • U164 (Plugins) ~
	Review & Retrospective • U126 has been split into U151, U155 and U160 • U133 has been dropped in favor of U164 • U164 is still missing its exercise solver parts • Good progress has been made	
M2	**Goals (Initial)** • U112 (Dashboard View) • U113 (Course View) • U116 (Exercise View)	**Goals (End of M1)** • U164 (Plugins) ✓ • U165 (Flows) ✓ • U166 (Exercise Solver View) ✓ • Refactor for U168 ✓
	Review & Retrospective • U112, U113 and U116 have been grouped into U166. • U164 and U165 are now working. • Some refactoring in relation to user-specific requests has been made in preparation for U168.	
M3	**Goals (Initial)** • U123 (Git) • U144 (OAuth)	**Goals (End of M2)** • U167 (Git) ~ • U168 (OAuth) ~ • Conduct user tests ✓
	Review & Retrospective • U123 is now included in U167. • U144 is now included in U168.	

<table>
<tr><th>#</th><th colspan="2">Details</th></tr>
<tr><td></td><td colspan="2">• U167 is more complicated than anticipated, which is why we require more time for its implementation.
• U168 could not be finished due to ongoing problems with the GitLab pipeline and long response times from the support team.
• Decent feedback received from the user tests.</td></tr>
<tr><td rowspan="2">M4</td><td>Goals (Initial)
• Finish the must-have requirements</td><td>Goals (End of M3)
• U167 (Git) ✓
• U168 (OAuth) ✓
• Implement some user feedback ✓
• Finish the must-have requirements ✓</td></tr>
<tr><td colspan="2">Review & Retrospective
• We were able to finish both U167 and U168 in this iteration.
• We were able to include some of the feedback received in the user tests.
• We went through all must-have requirements that were defined at the end of the SA/beginning of the BA and checked if we fulfilled them.
• We believe that we have adequately addressed all important must-have requirements and are satisfied with the results of our work.
• We have initiated a code freeze and are now focusing on the documentation.</td></tr>
<tr><td rowspan="2">M5</td><td>Goals (Initial)
• Finish the documentation
• Finish the abstract
• Finish the poster</td><td>Goals (End of M4)
• Finish the documentation ✓
• Finish the abstract ✓
• Finish the poster ✓</td></tr>
<tr><td colspan="2">Review & Retrospective
• We were able to submit all required documents.
• We have successfully finished our Bachelor’s thesis. ᕕ(ᐛ)ᕗᕕ(ᐛ·)ᕗ</td></tr>
</table>

Table 12: Details about the milestones of this Bachelor’s thesis

14. Retrospective

14.1. Context

Motivated by our firsthand experiences with the challenges in the current implementation of exercises at the OST Department of Computer Science, we committed ourselves to developing *Codable*. In this retrospective, we want to look back on our work and reflect on our approach and results. We assess our initial requirements, highlight our effective practices, and identify opportunities for improvement.

14.2. Requirements Assessment

In our opinion, we believe that we have successfully implemented the Minimal Viable Product (MVP) as defined in the preceding semester thesis (see Section 13.1). We also think that we have adequately addressed the aspects under Section 12.2.

In its current state, *Codable* allows the creation and management of exercises by lecturers and assistants. Exercises are structured into courses and can be granularly configured both in terms of their content (i.e. what the exercise should contain) as well as their logic (i.e. how the exercise should be evaluated). At the same time, the application provides a uniform interface for students to solve and submit exercises either locally with Git or via the browser.

This means that *Codable* could, for example, be used for the Object-Oriented Programming 2 course in the coming semesters. If it is well received, further development may take place to enhance its current features, which we would appreciate.

During the Bachelor's thesis, we realised that certain tasks had to be reprioritised and new requirements had to be added. For example, that the plugin system was more architecturally significant to the system than the import heuristics for courses. These changes and decisions are documented in Section 13.

14.3. Organisational Management

14.3.1. Teamwork & Meetings

Reflecting on our teamwork, our decision to meet in person once a week in Rapperswil proved to be effective in facilitating communication and collaboration. These meetings allowed us to share our progress, support each other and agree on the next steps. The dedicated Bachelor's thesis room at OST was practical, providing an environment conducive to concentration and equipped with essential infrastructure such as monitors, flipcharts, and coffee nearby :).

Our bi-weekly meetings with Prof. Dr. Frieder Loch and Dr. Juliane Fischer proved to be an effective means of addressing questions and receiving feedback. We appreciated their dedication throughout the project.

14.3.2. Project Planning

In our opinion, our project planning process was successful. We used milestones to effectively track progress without spending too much time on detailed planning or

diagrams. While a more flexible project planning approach may not be suitable for every project, it proved beneficial for our Bachelor's thesis.

The code freeze three weeks before the submission deadline was an advantage, as it allowed us to focus on the documentation, abstract, and the poster - tasks that required more effort than anticipated.

Using GitLab for issue tracking proved to be a pragmatic solution, as it centralised the code, documentation and the issues in one place. However, there were some challenges. For example, we struggled to define when an issue was considered closed, especially when dealing with issues and sub-issues.

For future projects, it would be beneficial to define that if all sub-issues are completed, the issue is considered resolved, with the disadvantage of having to specify all sub-issues. Alternatively, all sub-issues can be closed, but the issue may still remain open. As a result, the sub-issues do not provide an accurate indication of the progress of the issue.

14.3.3. External Dependencies

Due to our limited access, we relied on OST's support for some of the deployment and authentication processes. It was crucial to address these external dependencies early on, as they were time-consuming to manage.

In addition, the OST GitLab runners were a challenge as they were frequently unavailable, resulting in failed pipeline executions.

14.4. Conclusion

Our comprehensive user needs assessment and conceptual planning, initiated during the semester thesis, effectively structured the development process, minimizing unexpected challenges and unplanned issues. This thorough groundwork enabled us to create a user-centred application that, in our opinion, even in its current state already improves some of the organisational and qualitative problems of exercises in the Department of Computer Science.

We believe that this user-centred approach has proven to be much more beneficial than implementing an application based solely on stakeholder requirements or "our own opinions". As such, we are confident that *Codable* will be able to improve both student engagement and interest in exercises, even without the gamification mechanism originally envisioned, assuming that adequate resources are provided to ensure the sustainable growth of the system.

15. Outlook

15.1. Context

In the final part of our Bachelor's thesis, we will explore possible future directions for *Codable*, based on the insights and experiences we have gained throughout our project. We will look at short term goals, long term opportunities, and further development ideas that could influence the growth of *Codable*.

15.2. Short Term Goals

It is intended that *Codable* will be used productively at OST, most likely for the Object-Oriented Programming 2 course. This implementation would provide insight into the performance of the application in an academic environment. We hope to collect feedback to identify strengths, uncover existing problems and identify missing features that could improve the user experience of *Codable*.

15.3. Long Term Opportunities

If *Codable* is well received in its first implementation in a course, it may encourage more exercise makers to start using it. In addition, we could improve the system by increasing the variety of blocks available, such as multiple choice, and refining the heuristic import. These improvements would simplify the initial steps for exercise makers looking to migrate their exercises, making the transition more user friendly and less time consuming.

Furthermore, the flexibility of the plugin system allows exercise makers to create custom blocks if the existing ones do not meet their specific needs. This capability ensures that the platform can adapt and evolve to meet different requirements.

15.4. Further Development

In Table 13 we have identified a number of potential enhancements that could improve the functionality of *Codable*.

Analytics Dashboard	A comprehensive dashboard allows exercise makers to actively monitor completion rates and gather student feedback on exercises within *Codable*. This can help to identify areas where students frequently struggle and where exercise instructions are unclear. Exercise makers can use this information to improve their exercises and make more informed decisions about their courses.
Feedback Mechanisms	Exercise solvers can ask questions directly within each exercise. Others with the same question can read the answers provided, reducing the workload of exercise makers. Exercise makers can also give direct feedback on solved exercises, which can be used for mandatory assignments (de. *Testate*).

Template	A feature that provides templates that consist of multiple blocks and their corresponding flow would allow exercise makers to configure exercises more efficiently, as exercises often have the same structure for evaluation. Exercise makers can use a template and easily modify individual properties such as files and expected values.
Import Heuristics	The ability to import existing courses into our application would reduce the workload required for migration. Although this feature has been deprioritised as described in Section 13, it remains essential to streamline the migration process.
Exam Mode[20]	The ability to switch to a separate "exam mode", in which exercise solvers complete a set of exercises that follow the structure of the final exam, could significantly improve the learning process for exams. This could be achieved by allowing exercise makers to label certain questions as "exam questions" and then defining how their exam is usually structured (e.g. by configuring which topics with which difficulties come in which order). Our application could then generate a practice exam that can be solved in preparation for the real exam.
Repetition Features[20]	Different features for the repetition of exercises may help exercise solvers to learn topics in a more sustainable way. This could take the form of "repetition exercises" that are automatically generated after a few weeks, based on exercises that a exercise solver has previously struggled with. We could also allow exercise solvers to mark difficult exercises themselves in order to find them easier at a later time.
AI Integration[20]	To support exercise solvers in their exercises, an AI model trained for explaining certain concepts or detecting certain mistakes could be implemented in our application. This could allow exercise solvers to find solutions to their problems faster while at the same time reducing the workload of exercise makers with regard to answering questions.

Table 13: Ideas for the further development of *Codable*

15.5. Gamification

Now that *Codable* has solved the fundamental problems of exercise creation, management and evaluation, future development can focus on gamification features. By aligning these elements, such as leaderboards, rewards and quests, with the positive motivators identified in Section 8.3, we believe that we can increase student participation and enthusiasm even further.

[20]These ideas were already defined in the semester thesis. (Messmer & Fischler, 2023, pp. 47-48)

16. Bibliography

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Principles behind the Agile Manifesto*. https://agilemanifesto.org/principles.html

Brown, S. (2018). *The C4 model for visualising software architecture*. https://c4model.com/

Bundeskanzlei BK. (2023). *Leitfaden zum geschlechtergerechten Formulieren*. https://www.bk.admin.ch/bk/de/home/dokumentation/sprachen/hilfsmittel-textredaktion/leitfaden-zum-geschlechtergerechten-formulieren.html#download_als_pdf__content_bk_de_home_dokumentation_sprachen_hilfsmittel-textredaktion_leitfaden-zum-geschlechtergerechten-formulieren_jcr_content_par_tabs

Chamarty, S. *virtual refs*. Retrieved June 10, 2024, from https://gitolite.com/gitolite/vref.html

Chou, Y.-k. (2015). *Actionable Gamification: Beyond Points, Badges and Leaderboards*. CreateSpace Independent Publishing Platform. https://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/

Doran, G. T. (1981). *There's a S.M.A.R.T. way to write management's goals and objectives*. Management Review. https://community.mis.temple.edu/mis0855002fall2015/files/2015/10/S.M.A.R.T-Way-Management-Review.pdf

Erişen, D. (2023a). *Vanilla JavaScript single-page application (SPA) using MSAL.js to authorize users for calling a protected web API on Azure AD* [Computer software]. https://github.com/Azure-Samples/ms-identity-javascript-tutorial/tree/main/3-Authorization-II/1-call-api

Erişen, D. (2023b). *Vanilla JavaScript single-page application using MSAL.js to authenticate users to call Microsoft Graph* [Computer software]. https://github.com/Azure-Samples/ms-identity-javascript-tutorial/tree/main/2-Authorization-I/1-call-graph

Fakhroutdinov, K. (2015b, June). *Deployment Diagrams Overview*. https://www.uml-diagrams.org/deployment-diagrams-overview.html

Fakhroutdinov, K. (2015a, June). *UML Sequence Diagrams*. https://www.uml-diagrams.org/sequence-diagrams.html

Git Contributors. (n.d.-b). *10.2 Git Internals - Git Objects*. Retrieved June 10, 2024, from https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

Git Contributors. (n.d.-a). *4.1 Git on the Server - The Protocols*. Retrieved June 10, 2024, from https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols

International Organization for Standardization. (2019). *Human-centred design for interactive systems (ISO Standard No. 9241-210:2019)*. https://www.iso.org/standard/77520.html

ISO/IEC. (2023). *Product quality model (ISO/IEC Standard No. 25010:2023)*. https://www.iso.org/standard/78176.html

Jordan, N. (2021). *Geometa Lab at IFS* [Computer software]. https://github.com/geometalab/osmaxx/blob/develop/docker-compose-deploy.yml

Laubheimer, P. (2022). *Information Architecture: Study Guide*. https://www.nngroup.com/articles/ia-study-guide/

Loch, F., & Stolze, M. (2022, May). *Das Thema Accessibility in der Informatik-Ausbildung*. https://www.netzwoche.ch/news/2022-05-19/das-thema-accessibility-in-der-informatik-ausbildung

MDN Contributors. (2024b, February 9). *Web Components*. https://developer.mozilla.org/en-US/docs/Web/API/Web_components

MDN Contributors. (2024a, May). *Using shadow DOM*. https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM

Messmer, L., & Fischler, M. (2023). *Development of a Gamified Application for Programming Education*.

Microsoft. (2023). *A web API that calls web APIs: Code configuration*. https://learn.microsoft.com/en-us/entra/identity-platform/scenario-web-api-call-api-app-configuration?tabs=aspnetcore

Microsoft. (2023). *Authentication and authorization basics*. https://learn.microsoft.com/en-us/graph/auth/auth-concepts

Microsoft. (2024). *Database Providers*. https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli

Renzel, K., & Keller, W. (1997). *Client/Server Architectures for Business Information Systems A Pattern Language*.

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Starke, G., & Hruschka, P. (2024, May 2). *arc42*. https://arc42.org/

Strachan, D. *Daves Compendium of Level Select Screens*. Retrieved May 24, 2024, from http://www.davetech.co.uk/gamedevlevelselect

Tidwell, J., Brewer, C., & Valencia, A. (2020). *Designing interfaces : patterns for effective interaction design* (9781492051916; Third edition.). O'Reilly Media, Inc.

Universität Zürich. *Lernen mit Lernpfaden*. Retrieved November 22, 2023, from https://teachingtools.uzh.ch/de/tools/lernen-mit-lernpfaden

Zimmermann, O. (2022, September 9). *Artifact/Template: Architectural Decision Record (Y-Statement)*. https://socadk.github.io/design-practice-repository/artifact-templates/DPR-ArchitecturalDecisionRecordYForm.html

17. Listings

17.1. Figures

17.2. Tables

17.3. Architectural Decision Records (ADRs)

17.4. Code Snippets

Appendix

I Original Assignment

Entwicklung einer gamifizierten Anwendung für die Programmierausbildung (2)

Beteiligte Personen

Diese Arbeit wird verfasst von
Mathias Fischler; mathias.fischler@ost.ch
Lukas Messmer; lukas.messmer@ost.ch
Svenja Sutter, svenja,sutter@ost.ch

Betreuer dieser Arbeit
Prof. Dr.-Ing. Frieder Loch; frieder.loch@ost.ch

Expertin
Dr.-Ing. Juliane Fischer

Problembeschreibung

Gamification beschreibt den Einsatz von Spielelementen in einem spielfremden Umfeld. So können bestimmte Anwendungen motivierender gestaltet werden. Das Projekt baut auf einer Vorgängerarbeit auf, in der die aktuelle Praxis in der Organisation von Vorlesungen und insbesondere Übungsveranstaltungen erhoben wurde. In der Arbeit hat sich gezeigt, dass ein ganzheitlicherer Ansatz verfolgt werden muss, der auch die Organisation mit einbezieht. Die bestehenden Ergebnisse sollen im Rahmen dieser Arbeit umgesetzt, erweitert und evaluiert werden.

Formulierung eines konkreten Auftrags

Die Arbeit soll die folgenden Aspekte adressieren. Nach Absprache kann im Verlauf der Arbeit vom beschriebenen Auftrag abgewichen werden.

- **Analyse der bestehenden Anwendung.** Auf Basis einer bestehenden Anwendung sollen Anforderungen und Bedarfe der zukünftigen Nutzerinnen und Nutzer analysiert werden.
- **Recherche und Abgrenzung.** Es werden bestehende Anwendungen mit einer vergleichbaren Funktionalität betrachtet, um diese vom Ziel der Arbeit abzugrenzen. Relevante Quellen aus der wissenschaftlichen Literatur sind zu analysieren und bei der Lösungsfindung einzubeziehen.
- **Technologieauswahl.** Es werden Anforderungen formuliert, auf deren Basis eine nachvollziehbare Technologieauswahl durchgeführt wird. Grundsätzlich sollen freie und quelloffene Technologien bevorzugt werden.
- **Menschzentrierte Analyse.** Bei der Gestaltung aller Komponenten der Benutzerschnittstelle wird ein menschzentrierter Ansatz verfolgt. Dies beinhaltet z.B. die Entwicklung und Evaluierung von Konzepten mit realen Nutzern.
- **Prototypische Implementierung und Evaluation.** Der gewählte Ansatz wird prototypisch implementiert. Die Softwarearchitektur und das Ergebnis werden ausführlich dokumentiert. Es soll eine Implementierung mit potentiellen Usern in einer realistischen Umgebung getestet werden.
- **Kritische Reflektion.** Das gewählte Vorgehen wird kritisch reflektiert. Es werden konkrete Verbesserungsvorschläge am Vorgehen und am Ergebnis diskutiert.

Darüber hinaus ist eine geeignete Projektmanagementmethode auszuwählen und zu beschreiben. Die Beschreibung von Arbeitspaketen und einer angemessenen Anzahl von Meilensteinen ist obligatorisch. Die Planung ist zu dokumentieren und regelmässig im Projekt fortzuschreiben. Die Arbeitszeiten werden auf der Ebene der Arbeitspakete erfasst.

Umfang und Form der erwarteten Resultate

Die Ergebnisse der Arbeit (Quellcode der Software inkl. Dokumentation, sowie der schriftliche Projektbericht) werden den Projektbeteiligten zur weiteren Nutzung zur Verfügung gestellt.

Bei der Erstellung des Berichts werden die üblichen Qualitätsstandards für die Erstellung wissenschaftlicher Arbeiten angewendet. So müssen z.B. alle Quellen (z.B. für Definitionen) konkret und spezifisch nachgewiesen werden. Dies gilt auch für alle Abbildungen, sofern sie nicht selbst erstellt wurden. Quellennachweise erfolgen im Literaturverzeichnis im APA-Stil.

Bei der Erstellung des Berichts sollen die Empfehlungen des Bundes zur geschlechtergerechten Sprache einbezogen.

Anfangs- und Abgabetermin

- Start der Bearbeitung: **Montag, 19. Februar 2024**
- Abgabe: **Freitag, 14. Juni 2024 (17:00 Uhr)**

Zulässige Hilfsmittel und weitere Betreuung

Alle verwendeten Hilfsmittel werden in der Arbeit aufgeführt. Die Betreuung erfolgt durch die genannte Betreuungsperson. Es werden wöchentliche Beratungstermine vereinbart und von den Studierenden protokolliert.

II Requests & Requirements

II.A Functional Requirements

The following table shows the requirements of the system generated from GitLab issues on the 14.06.2024. All requirements are organized into one of 6 epics and may be further divided into one or more sub-requirements (sub-requirements in the sense that the requirement is only fulfilled if all sub-requirements are fulfilled). Additionally, grayed-out requirements are considered to be in the backlog. For the most up-to-date list of requirements, please refer to the internal GitLab repository.

IID	Requirement	State
163	The application allows creating and modifying courses.	
133	The application allows heuristic import of courses and exercises.	Postponed
136	The title of a course is determined heuristically.	Postponed
137	The weekly structure of a course is determined heuristically.	Postponed
138	The description of an exercise is determined heuristically.	Postponed
139	The coding files of an exercise are determined heuristically.	Postponed
140	The order of an exercise is determined heuristically.	Postponed
141	The solution of an exercise is determined heuristically.	
142	The test cases of an exercise are determined heuristically.	
151	The application allows CRUD on courses.	
127	A new/empty course can be created.	Done
134	The files of a course can be imported from GitLab.	Done
135	An existing course can be imported from an archive/folder.	Postponed
152	An existing course can be edited.	Done
153	An existing course can be deleted.	
154	The administrative information of a course can be imported from Adunis.	Done
162	An existing course can be copied to create a new course.	
155	The application allows CRUD on weeks.	
156	A new/empty week can be added to a course.	Done
157	An existing week can be edited.	Done
158	An existing week can be deleted.	Done
159	An existing week can be restructured in its order.	
160	The application allows CRUD on exercises.	

128	A new/empty exercise can be added to a week.	Done
129	An existing exercise can be edited.	Done
131	An existing exercise can be restructured in its order.	
161	An existing exercise can be deleted.	Done
164	The application allows adding custom exercise contents (plugins).	
118	The application provides a standard plugin with an online editor for coding tasks.	
119	The application provides a standard plugin for adding exercise descriptions.	
132	The application provides a standard plugin for running unit tests	
169	The application allows exercise makers to programmatically extend the exercise contents (i.e. plugin system).	Done
170	Plugins can receive files modified by exercise solvers.	Done
175	Plugins can have inputs (i.e. properties) provided by exercise makers.	Done
165	The application allows configuring exercise logic via a flow system.	
171	Plugins can be used to calculate the completion of an exercise.	Done
172	Plugins can be sequentially combined for more sophisticated evaluations.	Done
173	The evaluation of an exercise is run each time the exercise solver submits file changes.	Done
174	The evaluation of an exercise can be triggered manually (multiple times).	Postponed
166	The application allows working on exercises online via browser.	
111	The course selection shows all courses a user belongs to.	Postponed
112	The application contains a course selection.	Done
113	The application contains an exercise selection.	Done
114	The exercise selection shows all exercises of a course.	Done
115	Each exercise has a progress bar.	Done
116	The application contains an exercise view.	Done
117	The exercise view shows the contents of an exercise.	Done
120	An exercise can be submitted to trigger the plugin evaluation.	Done
167	The application allows working on exercises locally via Git.	
123	An exercise can be cloned as a Git repository.	Done
124	A course can be cloned as a Git repository.	Postponed
125	An exercise can be solved on a separate branch.	Postponed

168	The application allows user management using OST systems.	
145	The application supports user sign-in using OAuth.	Done
146	The role of a user is automatically determined using OAuth.	Done
147	A course can only be edited by its owner.	Postponed
148	The application can gain permissions to read GitLab repositories.	
149	Additional allowed editors can be added to a course by its owner.	
150	The access rights for a course are automatically imported from Adunis.	

II.B Feature Requests

The following table shows the feature requests of the system generated from GitLab issues on the 14.06.2024. For the most up-to-date list of feature requests, please refer to the internal GitLab repository.

IID	Requirement
5	I want to reduce the overhead for creating and managing exercises.
16	I want to easily migrate to a new solution.
3	I want to use GitLab (VCS) to manage my exercises.
14	I want to automatically synchronize access rights to my exercises (e.g. Moodle, Adunis)
7	I want to write exercises in markup languages that I am comfortable with.
8	I want to write exercises in Markdown.
10	I want to write exercises in HTML.
9	I want to write exercises in Jekyll.
11	I want to write exercises in AsciiDoc.
12	I want to write exercises in LaTeX.
72	I want to manage my exercises without using GitLab.
73	I want to manage my exercises directly inside the application.
6	I want to publish my exercises when I change them in GitLab.
15	I want to reuse exercises that have been created before (e.g. for a different course).
17	I want to automatically provision the components needed for my exercises (e.g. VMs, Hardware, etc.)
13	I want to link my exercises directly in the corresponding Moodle course.
64	I want to write exercises in WYSIWYGs (e.g. Microsoft Word).
27	I want to provide a wide variety of different exercises.
29	I want to provide practical exercises (e.g. programming tasks)

28	I want to provide theory exercises (e.g. text, images)
33	I want to provide exercises that integrate with other tools (e.g. virtual labs).
34	I want to provide exercises that span over a longer period of time (e.g. mini-projects)
76	I want to combine theoretical exercises with practical exercises.
30	I want to provide video exercises.
31	I want to provide exercises from external sources (e.g. Codewars)
32	I want to provide exercises that must be solved non-digitally (e.g. using hardware).
42	I want exercise solvers to submit their solutions for an exercise.
70	I want to evaluate the submissions of exercise solvers using custom scripts (e.g. SQL Parser).
71	I want to allow exercise solvers to run the evaluation scripts before submission.
65	I want submissions to be done in different formats (e.g. file upload, plain text, etc.)
46	I want to provide feedback directly on the submitted solution.
48	I want to provide feedback in form of text.
49	I want to provide feedback in form of an in-person discussion.
47	I want to provide feedback in form of an evaluation matrix.
50	I want to send my feedback to the exercise solvers using e-mail.
44	I want to allow exercise solvers to submit solutions as a team.
45	I want to allow exercise solvers to form teams by themselves.
43	I want to make submissions mandatory.
51	I want to view all submitted solutions of an exercise in one central location.
53	I want to allow exercise solvers to view the submissions of other exercise solvers.
52	I want to edit all submitted solutions locally in the editor I am comfortable with.
54	I want to calculate a grade based on the submitted solutions of an exercise solver.
55	I want to automatically detect plagiarism in the submitted solutions.
77	I want exercise solvers to review the submissions of other exercise solvers (e.g. peer reviews).
58	I want to provide automatic testing for an exercise.

59	I want to prevent exercise solvers from optimizing their code specifically for the test cases.
78	I want to be in control of how I solve exercises.
84	I want to solve exercises in an environment I am comfortable with (e.g. IDE).
94	I want exercises to be easy to set up.
97	I want exercises to be provided in a Git repository.
96	I want to solve exercises without having to install tools locally.
82	I want to be in control of where I solve exercises (e.g. campus, home).
79	I want to be in control of when I solve exercises.
80	I want to save exercises in order to work on them at a later time.
83	I want to access exercises from different devices (e.g. mobile).
95	I want to solve exercises in an online editor.
98	I want to view the solutions of an exercise in a browser.
81	I want to be in control of which exercises I solve.
85	I want to see the solutions of an exercise without having to solve it.
86	I want to have an overview of my exercises.
89	I want to know which exercises belong to which lecture (i.e. week).
90	I want to know which exercises belong to the current lecture.
91	I want to know which exercises belong to which topic.
88	I want to know which exercises are mandatory.
87	I want to view all contents of a lecture in one central location.
92	I want to know which exercises have mandatory attendance.
93	I want to solve exercises that are similar to the exam.
100	I want to receive feedback for my exercise submissions.
107	I want my solutions to be validated automatically.
101	I want to view feedback together with my submission.
108	I want to know why my solution is right or wrong.
102	I want to receive feedback in the form of text.
103	I want to receive feedback directly on the relevant lines of code.
35	I want to structure my exercises chronologically and thematically.
23	I want to incentivize exercise solvers to solve the provided exercises (e.g. with prices).
18	I want to provide exercises in collaboration with other exercise makers.
40	I want to provide my exercises to exercise solvers who have no knowledge of Git.

104	I want to solve exercises in a reasonable amount of time.
62	I want to allow exercise solvers to access help autonomously when problems arise.
22	I want to know how many exercise solvers are solving the provided exercises.
68	I want to allow exercise solvers to use the programming environment they are comfortable with (e.g. IDE)
60	I want to allow exercise solvers to test their code manually (e.g. using the main method)
61	I want to explain code to the exercise solvers using artificial intelligence.
69	I want to force exercise solvers to setup a local programming environment.
105	I want to solve exercises multiple times (e.g. for repetition).
106	I want to remind myself which exercises I would like to repeat.
19	I want to provide exercises in different settings.
20	I want to provide exercises for university courses (e.g. FH).
21	I want to provide exercises for courses of further education (e.g. CAS).
63	I want to allow exercise solvers to ask questions about exercises anonymously.
36	I want to include excerpts from different sources (e.g. code-snippets of official documentations)
37	I want to receive content contributions from exercise solvers.
38	I want exercise solvers to be able to rate the content contributions of other exercise solvers.
41	I want to show code examples in my browser.
56	I want to provide discovery tours to explain the code base of an exercise.
57	I want discovery tours to be expandable and maintainable (e.g. JSON under VC)
66	I want to simplify the setup of my exercises for the exercise solvers.
67	I want to provide setup templates for my exercises (e.g. Docker).
99	I want to solve exercises that use game-like elements (e.g. Kahoot).
109	I want exercise makers to enforce certain tools and standards.
110	I want to solve exercises with a more dynamic scope (e.g. mini-projects).

III Plugin Examples

III.A A Block Returning Plain HTML

The `BlockProperty` attribute configures a injection of a property value, configured by an exercise maker. In this case, the `IStaticallyReferencedFile` is a special interface (see Section 8.2.3.3) that will allow exercise makers to link a file that is part of a course to read the content during evaluation.

This block, if assigned to an exercise, will be evaluated every time the property is changed, as it contains no flow inputs (and no outputs as well). It is not dependent on any changes made by the user to any file. For example, a block such as this could be used to provide an exercise description that was already stored in an HTML file before migrating to *Codable*.

```
1  public class ForwardingHtmlBlock(
2    [BlockProperty] IStaticallyReferencedFile file)
3    : IBlock<RenderOutput.Html>
4  {
5      public Task<RenderOutput.Html> Evaluate()
6          => Task.FromResult(new RenderOutput.Html(file.ReadAllText()));
7  }
```

III.B A Block to Validate User Content

This block gets the latest state of all files that are editable by the exercise solver (see Section 8.2.3.3 for clarification regarding `IEnumerable<IFile>` and `IUserEditableFile`). In addition, it provides two properties to be edited by an exercise maker:

- `WhichFile`: The configured value defines the file to be checked by this block
- `ExpectedContent`: The content that the user's content will be compared to.

Additionally, it returns a custom type: `FileContentCheckResult`. This type has been annotated correctly to provide an output to the flow system. It could also contain render ouptuts, e.g. in the case of running a test suite, the HTML report.

```
public class FileContentCheckBlock(
    [FlowInput("Files")] IEnumerable<IFile> files,
    [BlockProperty("WhichFile")] IUserEditableFile file,
    [BlockProperty("ExpectedContent")] string expectedContent)
    : IBlock<FileContentCheckResult>
{
    public async Task<FileContentCheckResult> Evaluate()
        => new FileContentCheckResult(
            await (files
            .SingleOrDefault(f
              => f.RepositoryRelativePath == file.RootRelativePath)?
            .ReadAllTextAsync() ?? Task.FromResult<string>(null!))
            == expectedContent);
}
```

For the return type definition, a C# feature called *primary constructor* for records is being used. Because of this, to annonate the implied property correctly, the attribute needs to be given a explicit target.

```
public record FileContentCheckResult(
  [property: FlowOutput("ContentMatches")] bool Value);
```

For clarity, this would be the semantic equivalent without a primary constructor:

```
public record FileContentCheckResult
{
  public FileContentCheckResult(bool value)
  {
    Value = value;
  }

  [FlowOutput("ContentMatches")]
  public bool Value { get; init; }
}
```

III.C A Block Enabling Editing in the Browser

To add an editor, our provided `<monaco-editor>` custom component can be used as decribed in Section 8.2.3.2.1. It can simply be used as HTML tag inside of an HTML render output.

```
public class EditorBlock([BlockProperty] IUserEditableFile file)
  : IBlock<RenderOutput.Html>
{
  public Task<RenderOutput.Html> Evaluate()
    => Task.FromResult(
      new RenderOutput.Html(
        // lang=html
        $"""
        <div>
        <monaco-editor
          rootRelativePath="{
            HttpUtility.HtmlEncode(file.RootRelativePath)}" />
        </div>
        """));
}
```

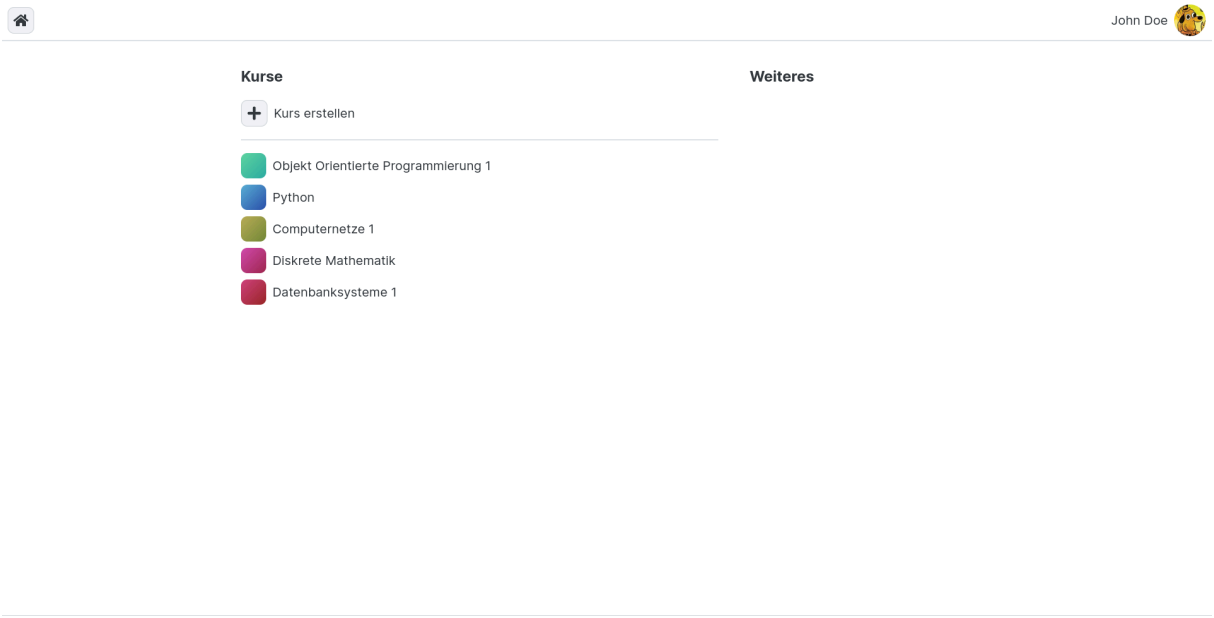
III.D A Block Setting Completion of an Exercise

The following block showcases flow inputs and outputs, as well as the use of a single primitive as return value. If a block only provides one output, it can be used directly as the return type of the `Evaluate` function, and annotated with an attribute using the explicit `return:` attribute target to help the compiler.

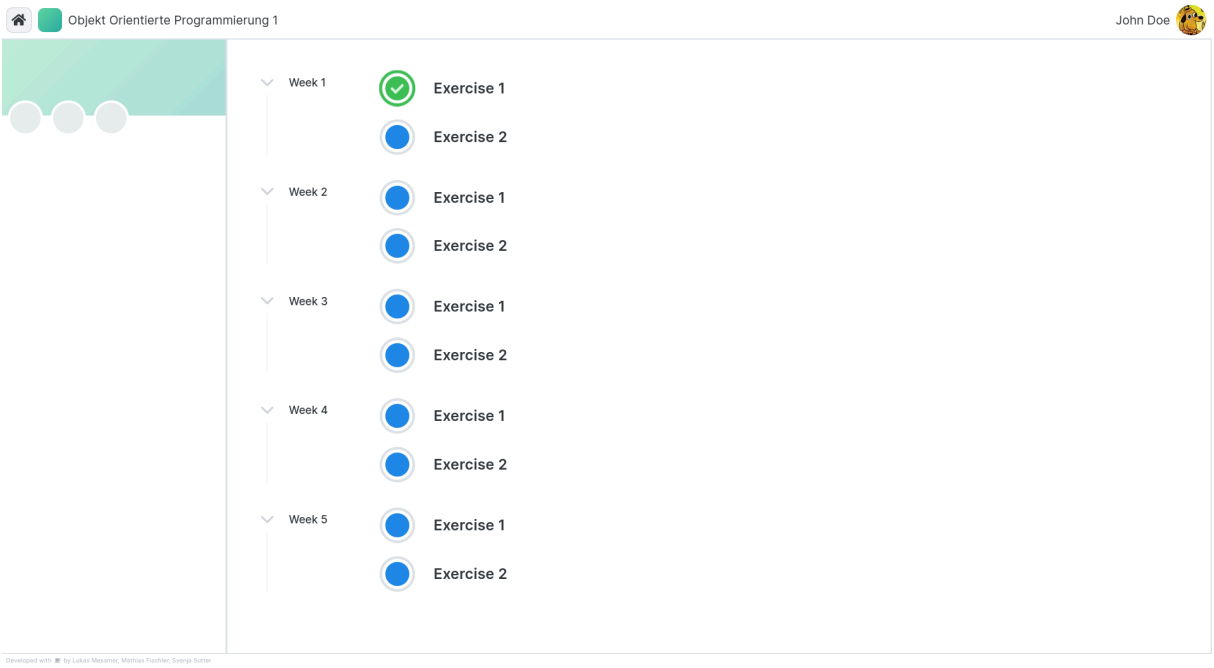
A block such as this can be used in combination with a block returning a boolean value (e.g. success/failure, such as the one in Section III.B) to set the exercise completion. Completion is a pre-defined flow input of an exercise, that the output of this block can be connected to. The completion is stored in the database, and shown in the user interface.

```
public class BoolToPercentageCompletionBlock([FlowInput] bool value)
  : IBlock<decimal>
{
    [return: FlowOutput("Completion")]
    public Task<decimal> Evaluate()
        => Task.FromResult(value ? 100m : 0m);
}
```

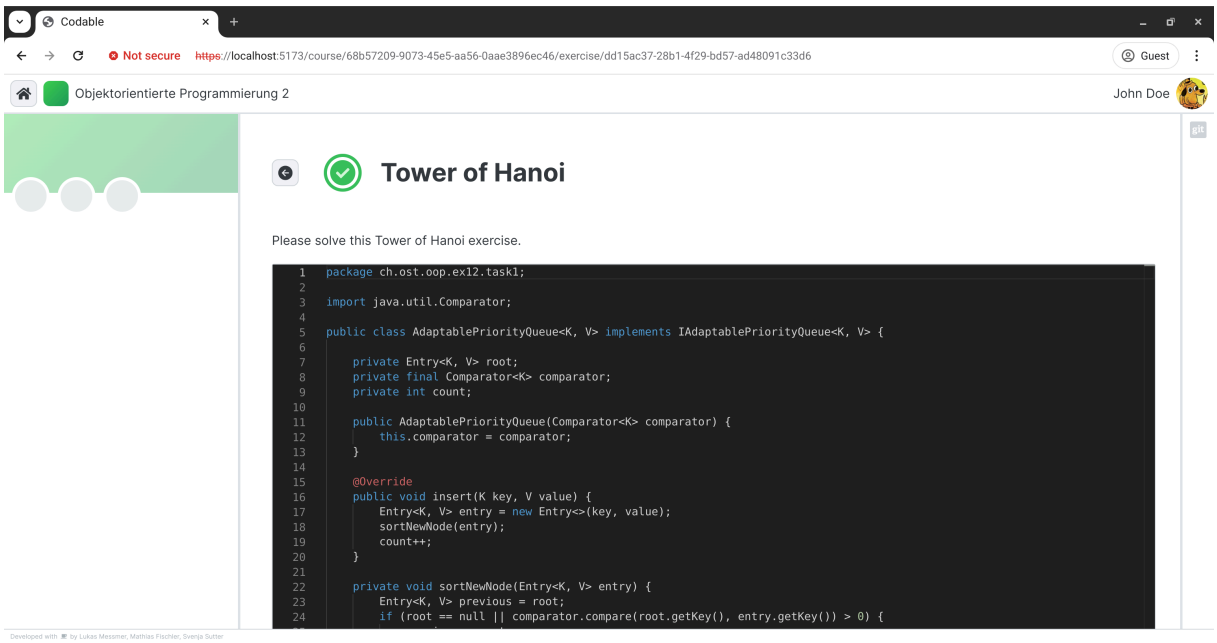
IV Screenshots of the UI



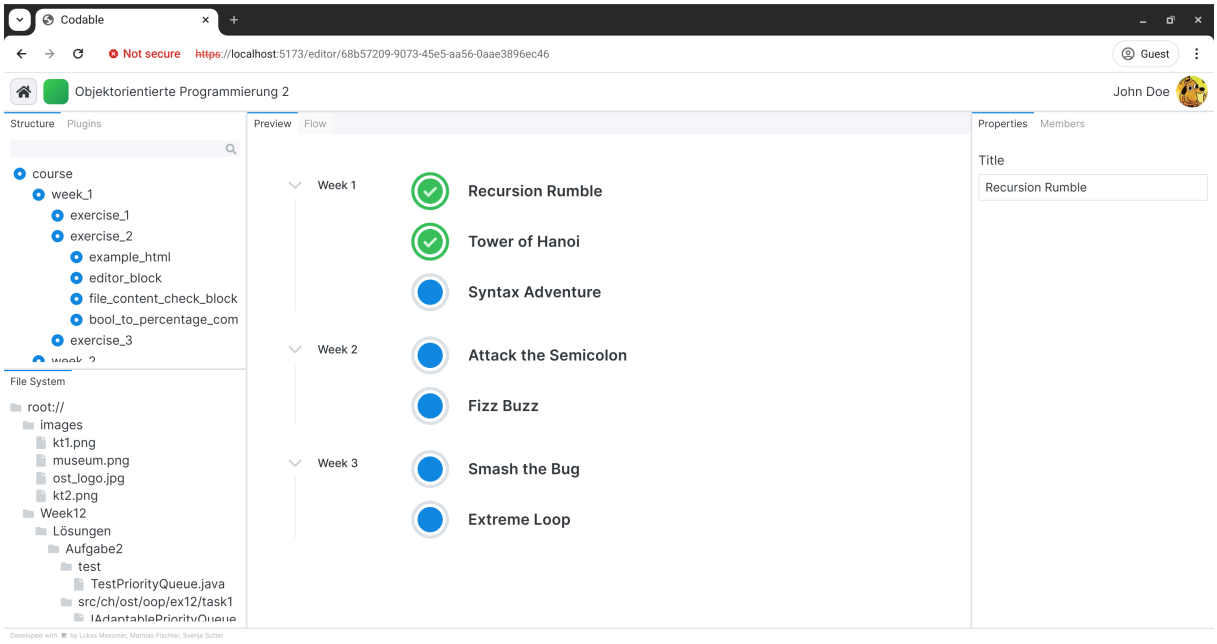
Codable course overview for exercise makers



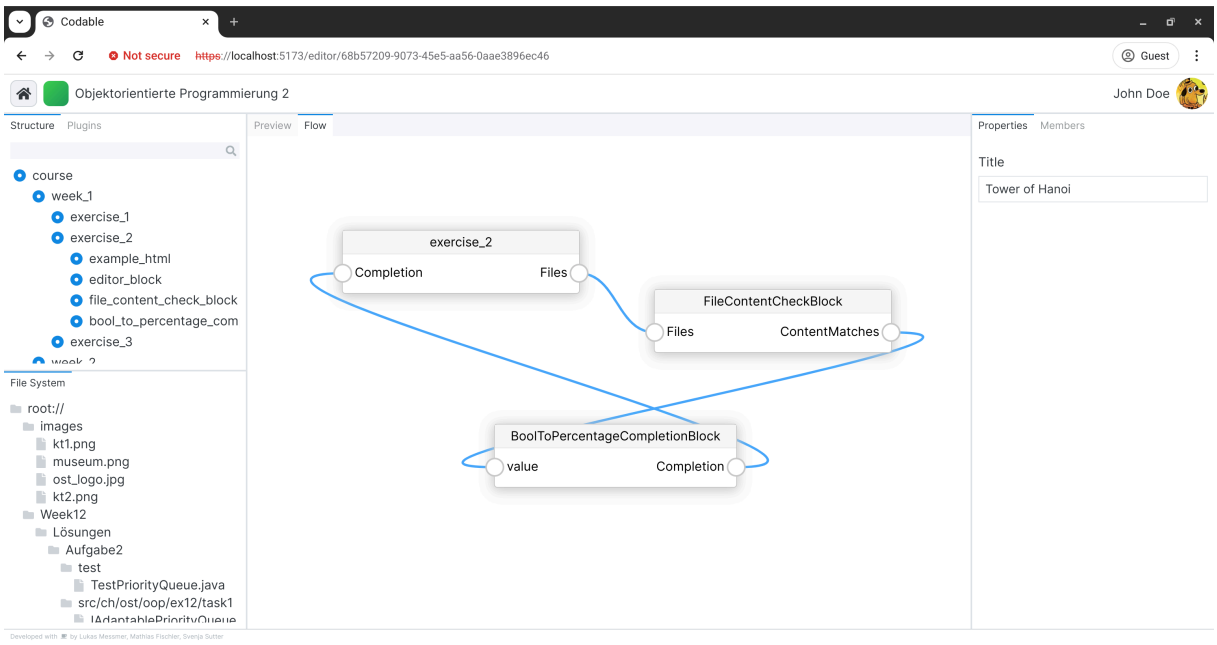
Codable weeks overview for exercise solvers



Codable exercise for exercise solvers



Codable editor (showing preview) for exercise makers



Codable editor (showing flow) for exercise makers

V Usability Testing

V.A Wissensziele

Exercise Maker

1. EM kann sich intuitiv durch die Software navigieren.
2. EM kann einen neuen Kurs anlegen.
3. EM kann einen Kurs im Editor öffnen und versteht die Funktionen der verschiedenen Fensterbereiche.
4. EM versteht den Zusammenhang von Nodes und Properties. Der Name eines Properties ist aussagekräftig genug, sodass der EM die Auswirkung davon versteht.
5. EM kann einen einfachen Flow definieren und versteht den Nutzen der Preview und des Flows.
6. EM versteht das Plugin-System einschliesslich der Funktionsweise vorhandener Blocks anhand des Namens.
7. EM kann Blöcke über das Plugin-System hinzufügen.

Exercise Solver

1. ES kann sich intuitiv durch die Software navigieren.
2. Es kann die Aufgaben der aktuellen Woche finden.
3. ES ist in der Lage eine Aufgabe online zu lösen und die Änderungen zu speichern.
4. Es erkennt den Status einer Aufgabe.

V.B Szenarien

V.B.a Exercise Maker

#	Name	Szenario	WZ	Weiteres
1	Kurserstellung	Bald startet das neue Semester. Sie unterrichten das Modul ”OOP2” und sollen dies nun erfassen. Ihre Unterlagen befinden sich aktuell im folgenden Github Repository: https://gitlab.ost.ch/oo/oop2/oop2_exercises.git.	EM1, EM2	Ist der Prozess der Erstellung eines neuen Kurses verständlich oder werden noch spezifischere Erklärungen für die verschiedenen Optionen benötigt? Versucht der EM Tastenkombinationen zu verwenden, welche noch nicht implementiert sind?
2	Kursstruktur und Editor	Fügen Sie nun dem Kurs eine neue Aufgabe “Attack the Semicolon” hinzu.	EM3, EM4	Findet der EM den Editor? Versteht er den Zusammenhang zwischen den Nodes und den Properties? Versucht der EM Tastenkombinationen zu verwenden, welche noch nicht implementiert sind?
3	Properties anpassen	Für das neue Semester wurde eine strukturelle Änderung vorgenommen. Die Modulkürzel beginnen jetzt mit M-.	EM3, EM4	
4	Zusammenhang Blocks und Flow Editor	In der vorhin erstellten Aufgabe “Attack the Semicolon” sollen die Studierenden die Datei “FileCopy.java” aus Woche 1 > Vorlagen > Aufgabe 1 > src/ch/ost/oop/ex1/task1 bearbeiten können. Anschliessend	EM4, EM5, EM6	Kann der EM die zur Verfügung gestellten Blocks anhand des Namens verwenden und im Flow Manager entsprechend konfigurieren? Versteht der EM, dass der

#	Name	Szenario	WZ	Weiteres
		legen Sie fest, dass die Aufgabe gelöst ist, wenn die Studierenden den Inhalt dieser Datei mit dem Text “done” ersetzen.		Flow Editor immer den Kontext einer Aufgabe definiert?
5	Plugin einrichten	Ihr Assistent hat einen neuen Block “Multiple-Choice” entwickelt. Fügen Sie Ihrem Kurs eine Beispielaufgabe hinzu, in der Sie diesen Block verwenden.	EM7	Versteht der EM wie er das Dll hochladen und verwenden kann?

V.B.b Exercise Solver

#	Name	Szenario	WZ	Weiteres
1	Kursübersicht	Heute hat die dritte Semesterwoche begonnen. Nach der Vorlesung “OOP2” folgt nun die Übungsstunde. Kannst du die Aufgaben für diese Woche finden? Weisst du, welche Aufgaben aus den vergangenen Wochen du bereits gelöst hast?	ES1, ES2, ES4	Benötigt es eine Markierung der aktuellen Woche? Ist der Fortschritt einer Aufgabe anhand des Progressbars erkennbar?
2	Aufgabe lösen	Du hast die Aufgabe der aktuellen Woche gefunden. Viel Erfolg beim Lösen.	ES3	Wird erkannt, wann die Aufgabe gespeichert wird? Ist der Prozess der Evaluation einer Aufgabe klar?
3	Aufgabenstatus	Wo kannst du nun erkennen, ob du die Aufgabe erfolgreich gelöst hast?	ES4	Ist der Fortschritt einer Aufgabe zu erkennen?

V.C Protokolle & Testauswertung

V.C.a Exercise Maker

Dozent & Studiengangsleiter Informatik

#	Auswertung	Wissensziel erreicht	Wissensziel nicht erreicht
1	Zuerst wollte die Testperson den Namen im Strukturbereich auf der linken Seite anpassen. Es wurde nicht direkt erkannt, dass sich die Eigenschaften auf der rechten Seite ändern. Dies ist vor allem bei einem breiten Monitor schwer zu erkennen.	EM1, EM2	
2	Die Aufgabe konnte via Kontextmenü hinzugefügt werden.	EM3, EM4	
3	Zuerst wollte die Testperson den Namen im Strukturbereich auf der linken Seite anpassen. Es wurde nicht direkt erkannt, dass sich die Eigenschaften auf der rechten Seite ändern. Dies ist vor allem bei einem breiten Monitor schwer zu erkennen.	EM3	EM4
4	Schnell wurde ein Editorblock hinzugefügt und die Datei per Drag & Drop zugewiesen. Es dauerte jedoch eine Weile, bis der Flow-Tab gefunden wurde. Es war auch nicht klar, dass die Elemente im Flow-Tab aus der linken Struktur stammten. Erst nachdem ich die Blockbeschreibungen als Hilfe gezeigt habe, konnte die Aufgabe gelöst werden. Die Testperson bestätigte jedoch, dass das System Sinn macht, wenn man es einmal verstanden hat.	EM4	EM5, EM6
5	Der neue Block konnte einfach hinzugefügt werden.	EM7	

Wissenschaftlicher Mitarbeiter

#	Auswertung	Wissensziel erreicht	Wissensziel nicht erreicht
1	Kurs konnte ohne Probleme erstellt werden.	EM1, EM2	
2	Das Kontextmenü zum Hinzufügen einer Aufgabe wurde erst nach mehrmaligem Klicken gefunden. Der Zusammenhang zwischen Nodes und Properties wurde nicht direkt erkannt, da auf der linken Seite z.B. der Kurs ausgewählt wird und auf der rechten Seite die Attribute erscheinen.	EM3	EM4
3	Nachdem klar war, wo die Eigenschaften angepasst werden können, war dies kein Problem mehr.	EM3, EM4	
4	Der Block Editor wurde korrekt ausgewählt, aber das Drag and Drop zum Füllen der Property war nicht intuitiv. Die Blöcke im FLow überlagerten sich beim ersten Mal, was nicht verständlich war. Ausserdem war nicht klar, warum der BoolToPercentageCompletionBlock benötigt wird. Durch die Beschreibung der Blöcke konnte der Flow korrekt zusammengestellt werden.	EM4, EM5	EM6
5	Nachdem der Tab Plugin gefunden wurde, war das Hinzufügen intuitiv.	EM7	

V.C.b Testevaluation

Im Usability-Test wurden die Wissensziele EM4, EM5 und EM6 nicht vollständig erreicht. Mit folgenden Massnahmen könnten die Ziele erreicht werden:

EM4:

- Kurzes Einführungstutorial für den Editor würde dem EM verdeutlichen, dass durch das Anklicken von Elementen, wie z.B. des Kurs Nodes, die entsprechenden Eigenschaften auf der rechten Seite angepasst werden.
- Die Zuweisung einer Datei zu einer Eigenschaft könnte neben der Drag & Drop-Funktionalität auch über eine Dropdown-Liste erfolgen. Damit wäre für alle EMs klar, wie die Datei zugewiesen werden kann. Eine andere Möglichkeit wäre, dies in das Tutorial zur Einführung in den Editor zu integrieren.

EM5:

- Auch für den Flow einer Aufgabe, würde ein Einführungstutorial helfen, die Funktionsweise besser zu verstehen.
- Derzeit wird der Flow einer Aufgabe in einem Kreis definiert, was für einige EMs etwas unverständlich sein kann, da das Lösen einer Aufgabe als sequentieller Prozess betrachtet wird. Daher wäre es sinnvoll, dies in einem zukünftigen Release als sequentiellen Graphen zu implementieren.

EM6:

- Eine Hilfe- und Informationsseite (/wiki siehe Figure 6) mit der Beschreibung der verfügbaren Blöcke würde den EMs helfen, die Blöcke zu verwenden, wenn der Name nicht verständlich genug ist.

V.C.c Integration in Applikation

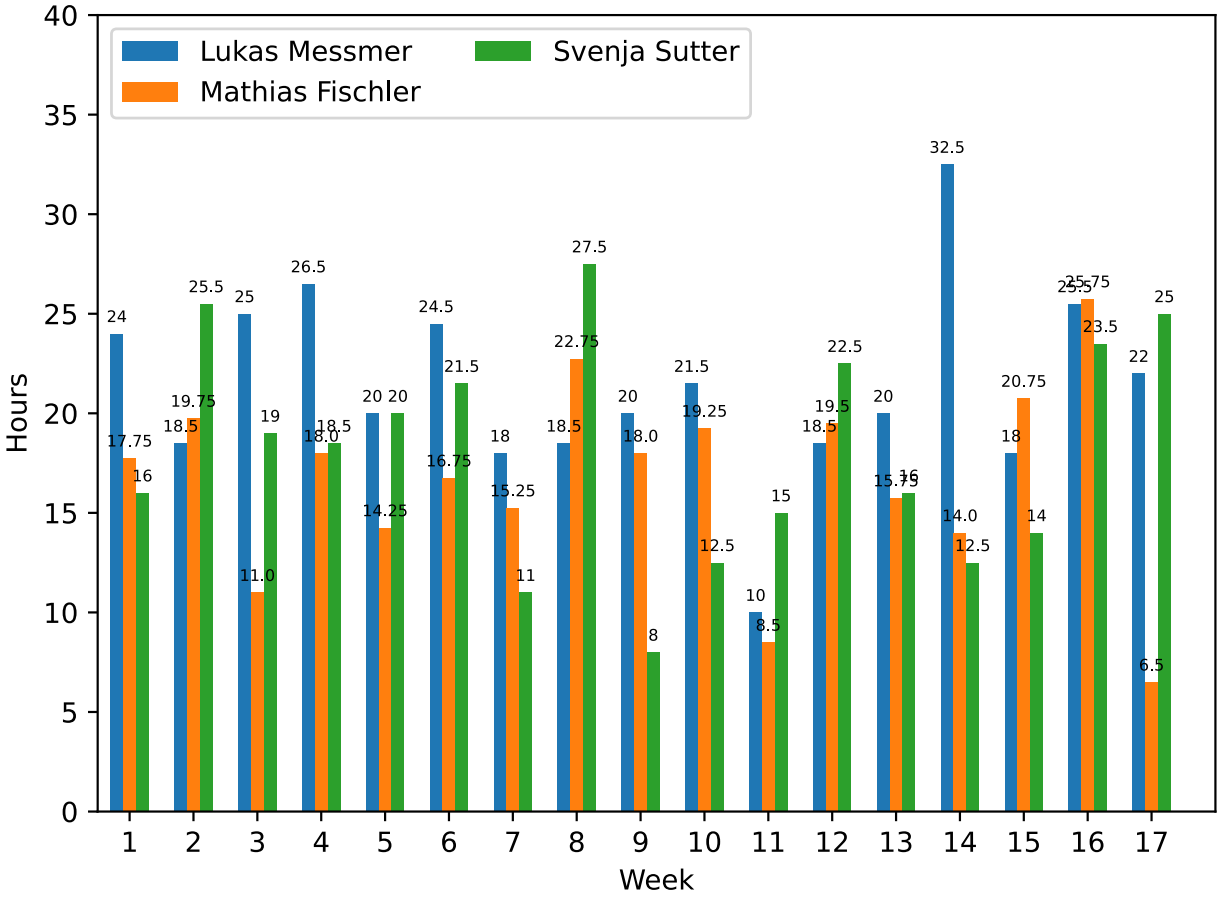
Um EM4 und EM5 vollständig zu erreichen, wäre ein einführendes Tutorial hilfreich. Die Erstellung eines solchen Tutorials ist zu diesem Zeitpunkt noch nicht sinnvoll, da wahrscheinlich Anpassungen vorgenommen werden, bevor *Codable* für ein Modul verwendet wird. EM6 wurde, wie oben referenziert, in die Dokumentation aufgenommen.

Folgende Rückmeldungen aus den Usability Tests konnten wir in *Codable* integrieren:

- Der Dateipfad einer zugewiesenen Datei zu einem Property kann nicht überprüft werden.
- Wenn keine Aufgabe gelöst ist und der Fortschritt bei allen Aufgaben bei 0% liegt, wird nicht erkannt, dass es sich um eine Progressbar handelt. Eine Prozentanzeige als Tooltip wäre hilfreich.
- Wenn der Flow angepasst wird, aber der Exercise Solver die Aufgabe bereits gelöst hat, wird die Auswertung nicht gestartet und somit nicht erkannt, dass die Aufgabe bereits gelöst wurde. Der Evaluationsstatus einer Aufgabe soll angezeigt werden.
- Die Möglichkeit, Strukturelemente umzubenennen, wird nicht benötigt und ist nur verwirrend.
- Die Aufgaben sollten aufsteigend nummeriert werden.
- Aktionen (z.B. beim Plugin-Upload) sollten dem Benutzer durch Benachrichtigungen bestätigt werden.

VI Time Tracking

This time tracking graph is automatically generated from GitLab time-tracking data. Additionally, 30 minutes are automatically added to this data for our advisor meetings every other week, starting week 2. There are also manual entries, if work was not clearly linked to an issue or merge request.



The time spent by the team members each week