VIA University College

Learnify - Project Report

Semester Project 3

Group 3

Students

Guillermo Sanchez Martinez (355442)

Piotr Wiktor Junosz (355502)

Halil Ibrahim Aygun (355770)

Alexandru Savin (354790)

Eduard Fekete (355323)

Supervisors

Joseph Chukwudi Okika (JOOK)

Jakob Trigger Knop (JKNR)

Character Count: 67439

Word Count: 9214

Software Technology Engineering

3rd Semester

December 18, 2025

Contents

1 Abstract

This work addresses the requirement for scalable and adaptable digital educational infrastructures through the development of Learnify, a distributed software system designed to ensure seamless content delivery and learning experience.

The primary objective was the construction of a resilient, multi-server solution that leverages the distinct advantages of a polyglot microservices architecture to enhance availability, data integrity, and system responsiveness. The architectural design enforces strict interoperability between Java and C# components, utilizing gRPC for high-throughput intra-service communication and HTTP for external client accessibility.

Data persistence is managed via a PostgreSQL database, while security concerns inherent to distributed systems are mitigated through the implementation of JWT authentication and Argon2 salted password hashing.

The development process followed an iterative methodology, where functional requirements were derived from User Stories established through specialized analysis and stakeholder interviews. The resulting product successfully manages concurrent user sessions within a distributed environment.

Verification procedures confirmed that the system meets essential requirements, demonstrating successful component interoperability, and robust data protection. The project concludes with a fully deployable distributed system, validating the architecture as a secure and stable foundation for the future scalability of educational technology.

2 Introduction

The acquisition of new knowledge is an essential part of human life and evolution. Functioning in society necessitates communication, which in turn requires a foundational level of knowledge (Habermas, 1984). Although mandatory education became a universal global standard during the late 20th and early 21st centuries (UNESCO, 2000), significant disparities remain - approximately 40% of the global population still lacks access to education in a language they understand (PTI, 2025).

The aim of this project is the development of a system which would be able to provide learning opportunities with a specific focus on accessibility, efficiency, and the optimization of learning processes. Beyond this, the system also aimed to ensure security, data integrity, and deployability of the solution among other things.

While the pursuit of knowledge has been a cornerstone of human development for millennia, the incorporation of digital technologies into education is in perpetual evolution (Siemens, 2005). A widely accepted model for learning with digital technologies has not been identified, mainly because the exponential increases in computing power and volumes of online information constantly redefine how users approach knowledge acquisition, processing, and retention (Haleem et al., 2022).

The approach of this project is to develop a distributed educational platform implemented using a polyglot architecture, utilizing a database for data persistence, and adapting a hybrid communication strategy that includes technologies such as gRPC and HTTP.

3 Main Section

The main section of this document is organized in the direction from the high-level analysis of the problem towards the specific implementation, testing and other relevant aspects of the solution created. This direction does not represent the chronological order of the project development and thus some aspects of the problem might not have a solution designed, implemented, tested, or deployed yet.

This overview represents merely a snapshot of the project development across each phase, and thus does not present a full solution or even analysis of the problem domain.

3.1 Analysis

The fundamental domain knowledge was at first derived from the analyzed problem domain via literature review, and research. Because of the data-driven nature of this project, the analysis focused on stakeholder interactions since the beginning to ensure proper understanding and to test the assumptions made. With more progress made on the solution, the analysis was evolutionarily refined to reflect the new understanding of the problem domain in the specificities of the solution space created by Learnify.

The most abstract and crucial aspect of the analysis was defining the system actors. The actors were defined to be:

- Learners
- Teachers
- Admins

The most debated aspect of the definition was the relationship between the roles, and particularly how teachers and admins relate to it. It was established that teachers and admins are a type of a learner, and this was confirmed throughout the project most importantly because:

- both teachers and admins were expected to be skilled users of the platform (Appendix 10.1, Interview_261125.pdf) and were supposed to be educated on it (Appendix 10.1, Interview_121025.pdf)
- both teachers and admins were understood as learners within the leaderboard setting and were expected to be equal participants in it (Appendix 10.1, Interview_121025.pdf)

3.1.1 Actor Descriptions

3.1.1.1 Learner Learners strive for knowledge acquisition. They want to be motivated to learn and they should be allowed to have a structured way of learning new information. They want to be able to learn various different topics at their own pace. They are looking for a gamified experience.

3.1.1.2 Teacher Teachers are trusted Learners, who also want to share their knowledge with others. They want to be able to manage courses easily and have a structured way of doing it.

3.1.1.3 Administrator (Admin) Admins are trusted Learners, who have the right to manage the platform. They should be able to manage all learners, and platform settings.

3.1.2 Requirements

3.1.2.1 Functional requirements The functional requirements are structured as user stories to better capture the perspective of the actor and to clarify permissions and intentions behind each requirement. This way the user stories served as the fundamental source of truth and a guideline for understanding the problem and being able to design a solution that would address the problem preserving the idea behind the intention of the actor.

ID	User Story
USL1	As a Learner, I want to register for an account so that I can access the platform.
USL2	As a Learner, I want to log in so that I can access the platform from my account.

ID	User Story
USL3	As a Learner, I want to see in which courses I am enrolled in, so that I can continue where I left off.
USL4	As a Learner, I want to continue learning where I left off, so that I don't have to start over every time.
USL5	As a Learner, I want to see all available courses, so that I can explore and choose what I want to learn.
USL6	As a Learner, I want to filter courses, so that I can find specific content quickly.
USL7	As a Learner, I want to unenroll from a course, so that I can stop learning a course I no longer want to finish.
USL8	As a Learner, I want to view the Leaderboard, so that I can compare my progress with other learners.
USL9	As a Learner, I want to view my Profile, so that I can see my personal account details.
USL10	As a Learner, I want to test my knowledge within the course, so that I know I understood the topic and I am not bored.
UST1	As a Teacher, I want to submit a course draft, so that I can find out if my course idea is relevant for the platform.
UST2	As a Teacher, I want to manage course content, so that I can correct or improve previous work.
UST3	As a Teacher, I want to edit course information, so that I can correct mistakes.
USA1	As an Admin, I want to see all drafts, so that I know what drafts are waiting for approval.
USA2	As an Admin, I want to approve course drafts, so that the teacher knows they can work on such course.
USA3	As an Admin, I want to add course categories and languages, so that the platform can easily adapt to new content.
USA4	As an Admin, I want to manage users' roles, so that I can manage what access is given to the platform and to what degree.
USA5	As an Admin, I want to disapprove course drafts, so that the teacher knows such course is not needed at the moment.

Table 1: Functional Requirements (Appendix 2.1 Requirements)

The user stories are sorted based on the actors to which they correspond, not according to the chronological order in which they were added/discovered. The chronological order is also not perfectly reflected on the IDs, as these were not always static for the same user story (when managing them, they would be adjusted)

3.1.2.2 Non-functional requirements

1. The system must be polyglot
2. User passwords must be securely stored at rest
3. The system must be deployable
4. The system must be color-blind friendly
5. The system must be distributed

3.1.3 Use cases and their related requirements

In order to address the user stories, use cases of the system were developed, which further clarified the requirements and provided a basis for understanding the system behaviour (giving basis for dynamic rather than static analysis).

The use cases developed are shown in a table below:

ID	Use Case
UC1	Register
UC2	Log in
UC3	Manage Personal Learning
UC4	Browse and Search Catalog
UC5	Complete Learning Step
UC6	View User Profile
UC7	View Leaderboard
UC8	Create Course Draft
UC9	Edit Course Content
UC10	Manage System Settings
UC11	Review Course Drafts
UC12	Manage User Roles

Table 2: Use Cases (Appendix 2.2 Use Cases)

The table below shows how the use cases are related to the user stories

User Story	Use Cases Addressing It
USL1	UC1
USL2	UC2
USL3	UC3
USL4	UC3, UC5
USL5	UC4
USL6	UC4
USL7	UC3
USL8	UC7
USL9	UC6
USL10	UC5
UST1	UC8
UST2	UC9
UST3	UC9
USA1	UC11
USA2	UC11
USA3	UC10
USA4	UC12
USA5	UC11

Table 3: Use Cases and their related requirements (Appendix 2.2 Use Cases)

3.1.4 Use case diagram (UCD)

To depict how the use cases were related to the system actors, a use case diagram was created as shown below:

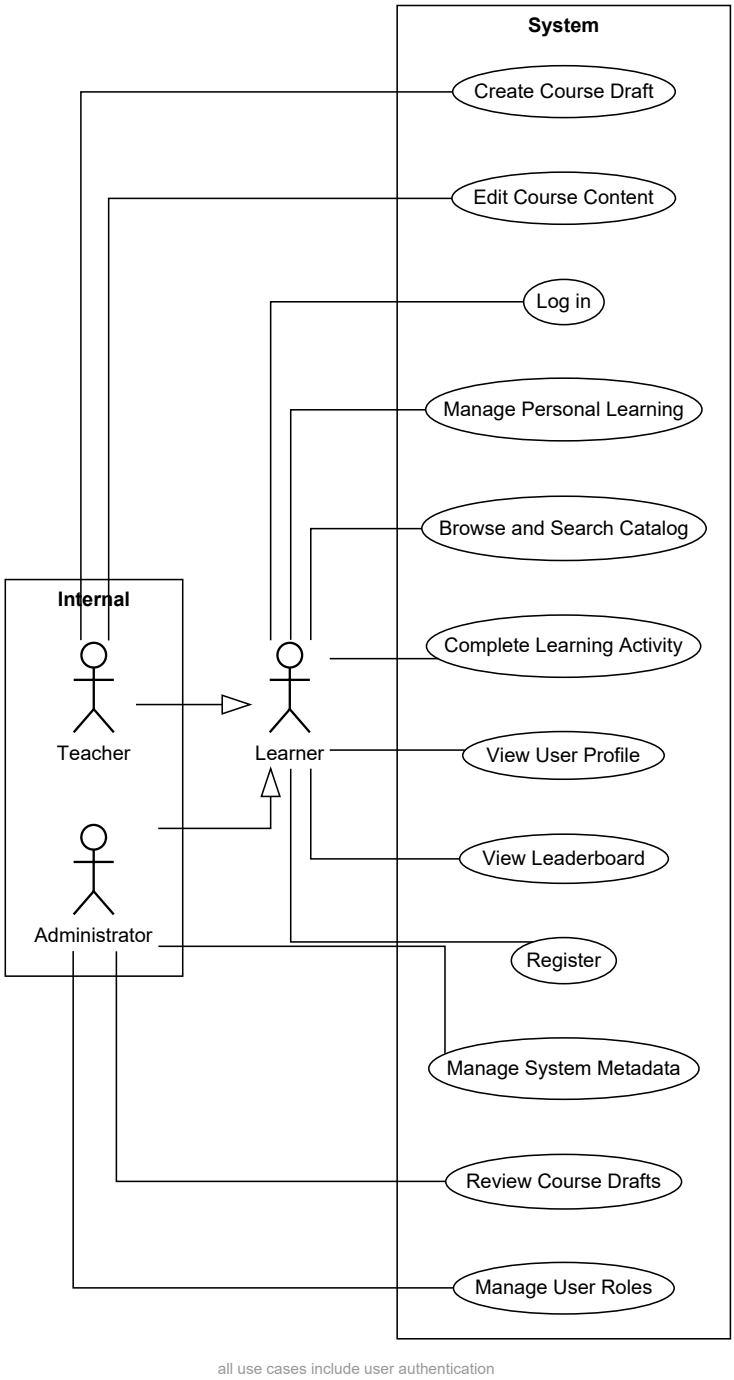

Figure 1: Use Case Diagram (Appendix 2.2 Use Cases)

As can be seen, the UCD also introduced the internal boundary for teachers and admins - specifying that these actors are not simply learners with privileges but there is a boundary to be crossed when becoming a teacher or an admin. The UCD also specifies the system boundary, which in the case of Learnify covers all the use cases developed.

3.1.5 Use case descriptions

In order to fully describe the use cases, use case descriptions were created; an example below shows such use case description, specifically for the UC5 - Complete Learning Step use case:

ID	UC5
Use Case	Complete Learning Step
Summary	A student engages with course content by completing exercises to test their knowledge.
Actor	User
Precondition	The User is enrolled in a course and is viewing a learning step.
Postcondition	Scenario A: The User answers correctly and proceeds. Scenario B: The User answers incorrectly and receives feedback.
Base Sequence	1. The System presents a learning step. 2. The User provides a response to the activity. 3. The User submits the response. 4. The System evaluates the response. 5. The System provides positive feedback. [ALT1] 6. The User proceeds to the next unit.
Alternate Sequence	[ALT1] Incorrect Answer: 5a. The System determines the response is incorrect. 5b. The System provides corrective feedback. 5c. The User attempts the activity again (Go to step 2).

Figure 2: Complete Learning Step Use Case Description (Appendix 2.2 Use Cases)

As seen above, the use case descriptions provided a structured way of understanding how the system should behave and gave a strong basis for the test cases.

All the use case descriptions were made in the same format with:

- Use Case ID and Name
- Summary
- Actor(s)
- Preconditions
- Postconditions
- Base Sequence
- Alternative Sequences

It was also determined that when the system fails to perform its action, the user should be notified of the

error and the inability to proceed. This was noted on the side of the Use Case Descriptions as a general alternative sequence.

3.1.6 System sequence diagrams (SSD)

System Sequence Diagrams (SSDs) were developed to illustrate the interaction between the system actors and the system as a black box. By focusing on the input and output events, the SSDs helped in identifying the necessary system operations and the data that needs to be exchanged to fulfill each use case.

An SSD was created for each of the 12 use cases, ensuring that the dynamic behavior of the system is fully captured from an external perspective.

The figure below shows the SSD for UC5 - Complete Learning Step, which highlights the iterative nature of the learning process and the system's role in providing feedback.

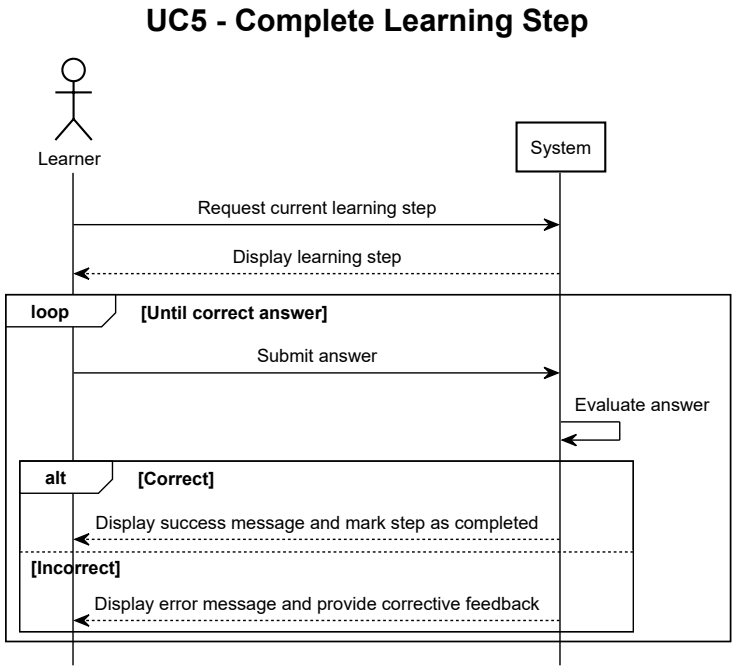

Figure 3: Complete Learning Step SSD (Appendix 2.3.2 System Sequence Diagrams)

The system sequence diagrams were kept simple and focused on the core interaction. Nevertheless, for UC9 and UC11 it was decided to include alternate paths while not including anything else than the main success scenarios.

3.1.7 Activity diagrams

The development of activity diagrams was crucial in understanding the dynamic behaviour and the interplay of several use cases and domain entities. While use case descriptions provide a structured textual representation, activity diagrams allow for a visual understanding of the logical flow, decision points, and the interaction between the user and the system's core components.

A set of activity diagrams was developed to cover the most critical workflows of the Learnify platform, including user onboarding, course discovery, content creation, and the learning process itself.

The activity diagram below illustrates the core workflow of a Learner interacting with the platform. It demonstrates the interplay between UC3 (Manage Personal Learning), UC5 (Complete Learning Step), and

UC7 (View Leaderboard).

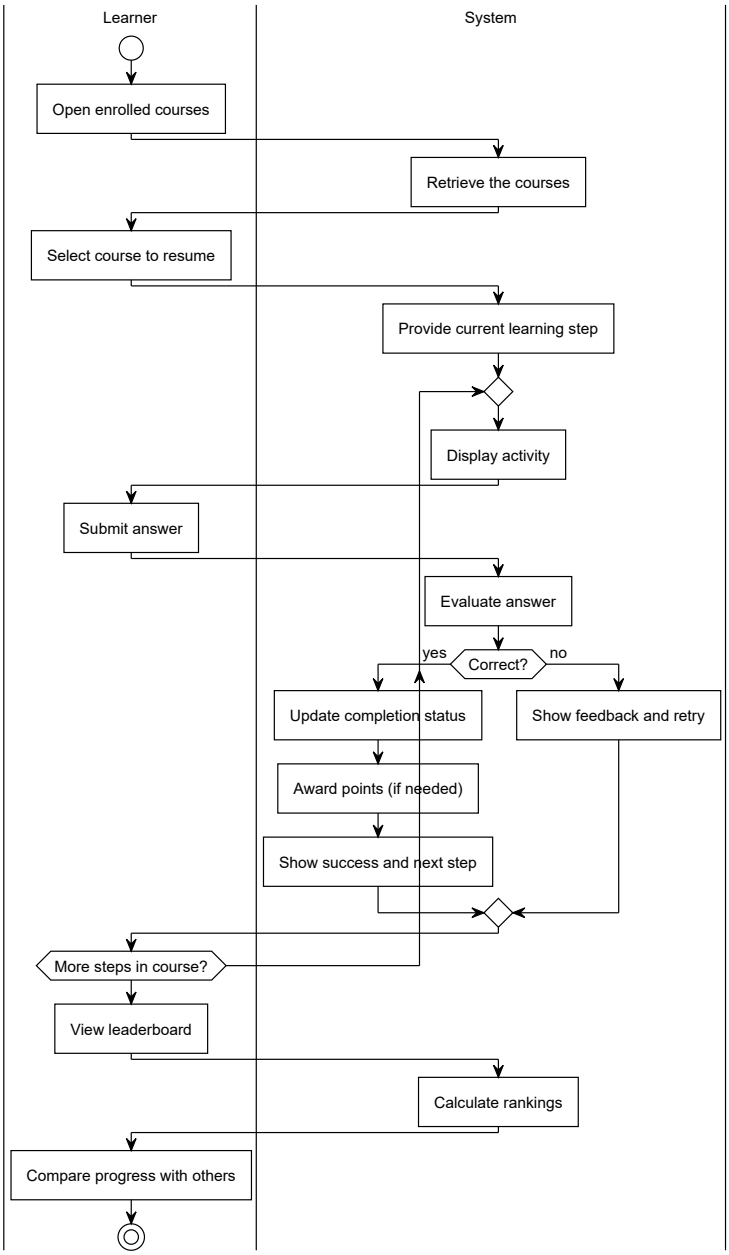

Figure 4: Learning and Achievement Activity Diagram (Appendix 2.3.1 Activity Diagrams)

This diagram is arguably one of the most important aspects of the application as it demonstrates something similar to a core loop of the system - a typical path a user takes within a session.

By modeling these workflows, the analysis phase ensured that the system's dynamic behavior aligns with the identified user stories and the relationships defined in the domain model.

3.1.8 Test Cases

Following the definition of the system's dynamic behavior through use cases, SSDs and activity diagrams, a set of high-level test cases was derived to formalize the acceptance criteria for the Learnify system. These test

cases were constructed directly from the use case descriptions, specifically targeting the preconditions, base sequences, and alternative sequences defined in the previous sections.

The objective of defining these test cases during the analysis phase - rather than the testing phase - was to ensure understanding of system's functionality and to aid the definition of done.

The table with all test cases can be found in Appendix 2.4 Tests - the table provided there also contains the test case results as these test cases were used during the testing phase as well.

3.1.9 Domain model

The domain model was constructed for this project to better understand the problem domain and to aid communication among stakeholders. The crucial aspect of developing the domain model was identifying the relationships between different kinds of users, in particular the roles and responsibilities of Learners, Teachers, and Administrators; which had to be combined with the security aspect of the system as well as had to align with the shared understanding of the stakeholders.

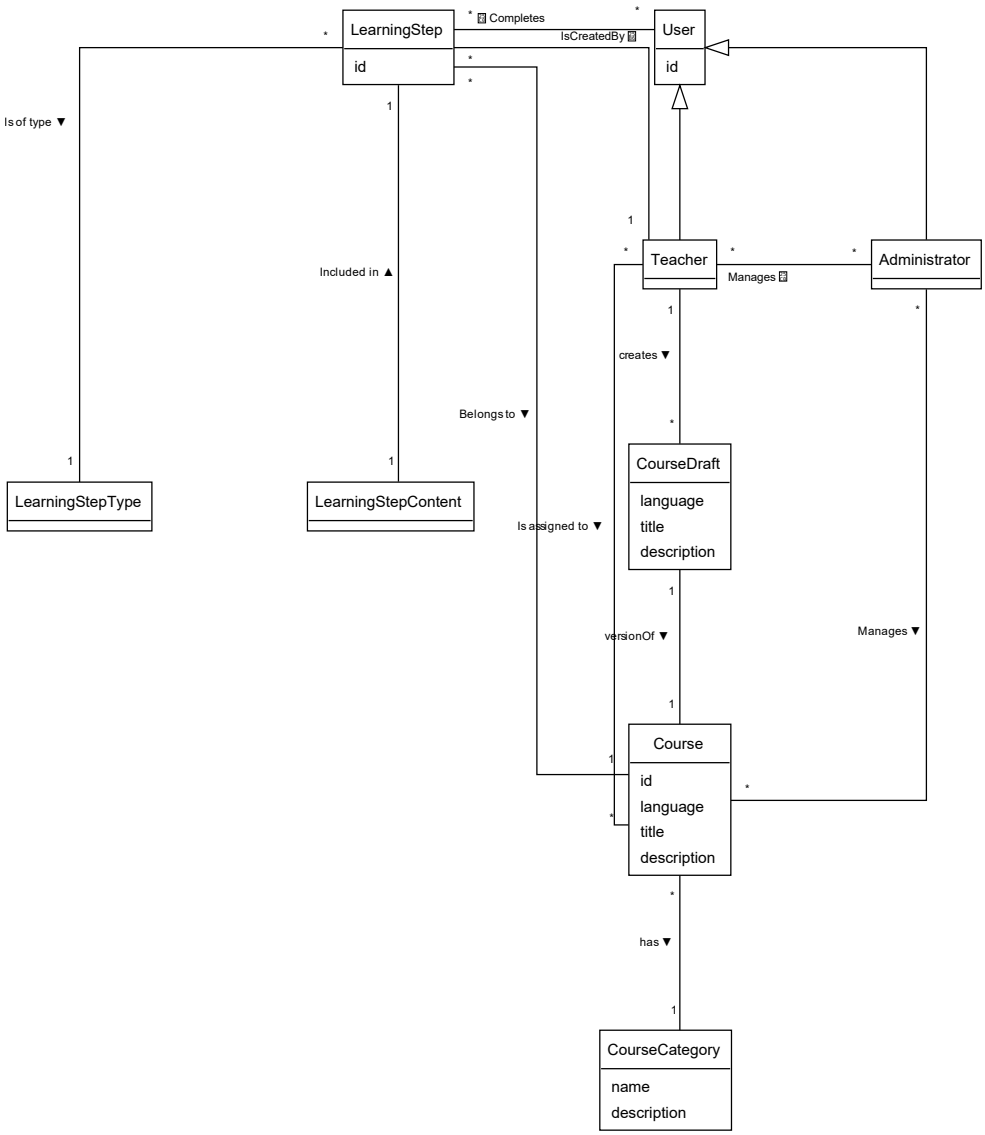

Figure 5: Domain Model (Appendix 2.3 Diagrams)

The figure above shows the domain model for Learnify. It can be seen that the crucial aspect of fully describing the different system roles is understanding what entities exist and how they relate to each other.

The inheritance from the User entity reflects the fact that all system users have some common basic attributes, rights and behaviours that can be generalized.

The philosophy behind the attribute selection and abstraction into separate entities was to provide full flexibility for future development of the system, and ensuring that no restrictions are imposed for no reason. For example, Learning Steps are abstracted into three different entities to fully support any kind of idea of a learning step as seen both in the domain and from stakeholder interactions.

At the same time, the domain model was kept in its simplest form possible in terms of more abstract entity concepts. As can be seen on the domain model, the core aspects of the system are:

- Users
- Courses
- Learning Steps

And it could be further argued that Learning Steps only exist as a part of Courses, therefore the Domain Model is centered around the idea of Users learning from Courses, which did not change from the initial vision of the system.

The inclusion of stakeholders as entities within the domain model mostly arose from the need of defining and understanding attributes and relationships of Users.

3.1.10 Security Requirements

The security requirements for the system were developed as a part of the threat modelling process. The security objectives were as follows:

- **Confidentiality:** Protect user passwords and personal data from unauthorized disclosure.
- **Integrity:** Ensuring that data can not be altered or tampered with by unauthorized parties.
- **Availability:** Ensure the system remains accessible during high traffic or denial-of-service attempts.
- **Accountability:** Actions must be uniquely traceable to a specific entity.
- **Authenticity:** Verify that data inputs and users are genuine.

These objectives were developed based on the CIA triad and expanded to fit the needs of the system (Appendix 7.1 Threat Model).

Similarly to other parts of the analysis phase, stakeholder interactions shaped the system's security requirements. In particular, even testing the prototype (before having the core system functional) showed concerns about the authority of Administrators and Teachers, and needs for accountability for actions.

3.2 Design

3.2.1 System design

The aim of the designing phase was to establish a clear vision and guide for implementing the solution. Design phase created the largest gap between the initial problem definition and the approach taken to solve it by transforming certain concepts into an implementable or more flexible form (e.g. user roles).

3.2.1.1 Wireframes Wireframes were used along other rough sketches to design the components of the user interface without the hassle of dealing with the final styling. Because of the chosen methodologies, wireframes provided a strong basis for creating HTML skeletons of various razor pages.

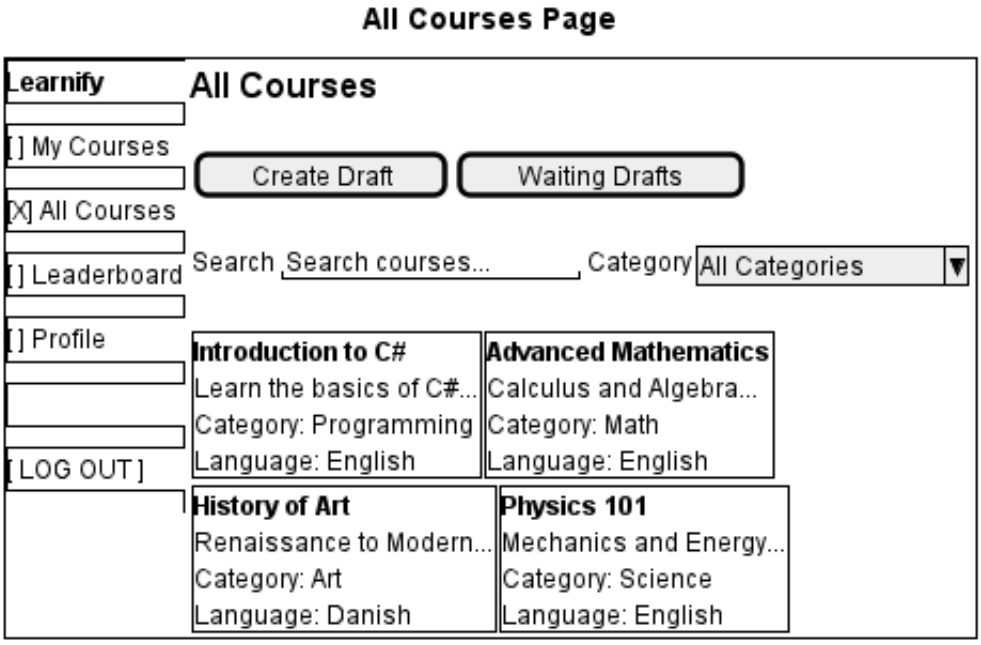

Figure 6: All Courses Page Wireframe (Appendix 9.1 Wireframes)

The figure above depicts one of the wireframes created for the system - all courses page or as described in the analysis - "The Course Catalogue".

The interface also includes features which are showed only to users with specific roles (Teacher, Admin); it can be seen that there are two action buttons named "Create Draft" which appears only for users with a Teacher role and "Waiting Drafts" which appears only for users who are an Administrator. These buttons give Teachers and Administrators quick access to content creation and moderation tool while keeping the learner view free from unnecessary elements.

The fidelity of the wireframes was kept low but the transformation into the chosen technology PlantUML (Roques, n.d.) provided a more visually accurate representation with less details provided from the creation inputs; ultimately resulting in wireframes that appear of higher fidelity.

3.2.2 Architectural overview

The architecture is a distributed, three-tier solution that aims for strict separation of concerns, scalability, and polyglot interoperability. The architecture facilitates interaction between a C# frontend, a C# logic middleware, and a Java data persistence layer.

3.2.2.1 Client The client-side application is built using Blazor in C# .NET. This layer hosts the web interface accessible to all system actors: learners, teachers, and admins. It is responsible solely for UI rendering and user input handling, delegating all business logic to the backend services.

The solution utilizes Blazor Server, which decouples the UI rendering from the client. This means that a significant part of the interactions is done on the blazor server rather than directly on the client machine.

3.2.2.2 Logic Tier The Logic Server, implemented in C# (ASP.NET Core), and acts as the central orchestrator. It exposes a RESTful API via HTTP(s) to the client, ensuring broad compatibility and standard web communication. This layer handles authentication, authorization (RBAC), and feature-specific business rules. It serves as a protocol bridge, translating external HTTP requests into internal gRPC calls for the data layer.

3.2.2.3 Data Tier The Data Server is implemented in Java (Spring Boot), fulfilling the project's polyglot requirement. Communication between the Logic and Data servers is conducted via gRPC - Google's Remote Procedure Call framework. This choice leverages Protocol Buffers (Protobuf) for binary serialization, resulting in lower latency and higher throughput compared to text-based JSON over HTTP.

3.2.2.4 Data Persistence At the foundation of the architecture is a PostgreSQL database. The Java Data Server manages all database interactions, ensuring that the Logic and Client layers remain agnostic to the underlying storage mechanics.

3.2.2.5 Architectural Diagram and Justification The figure below depicts the final architecture of the system as described above:

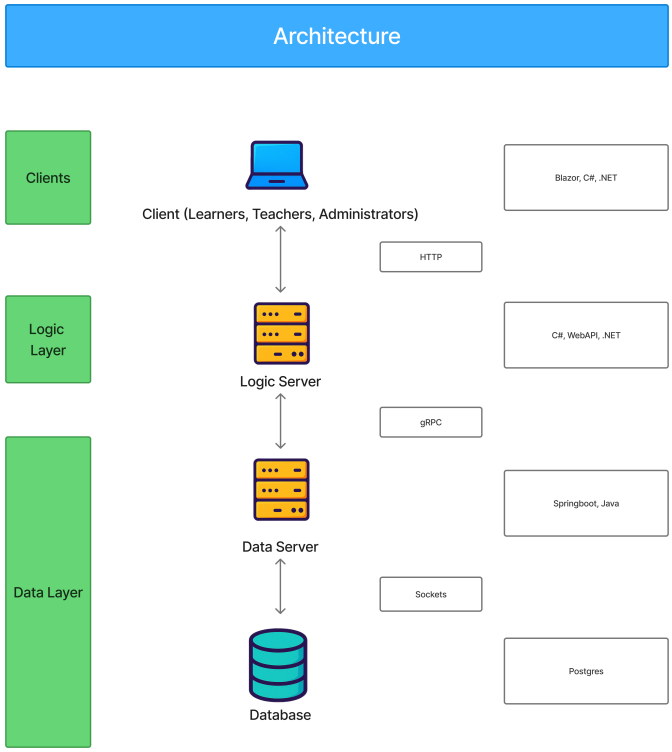

Figure 7: Architectural Overview (Appendix 11.1 Architecture)

The mentioned technologies, frameworks, and tools were chosen both to fulfill the requirements but also prioritized flexibility and ease of development.

3.2.3 Class diagram design

There is a class diagram for each of the servers, demonstrating their independence. These are the Client App, Logic Server and Data Server.

3.2.3.1 Client App Class Diagram This server is responsible for displaying the system to the client and to help the client navigate through our system, giving freedom to the user to use the system as they please, using high-level methods.

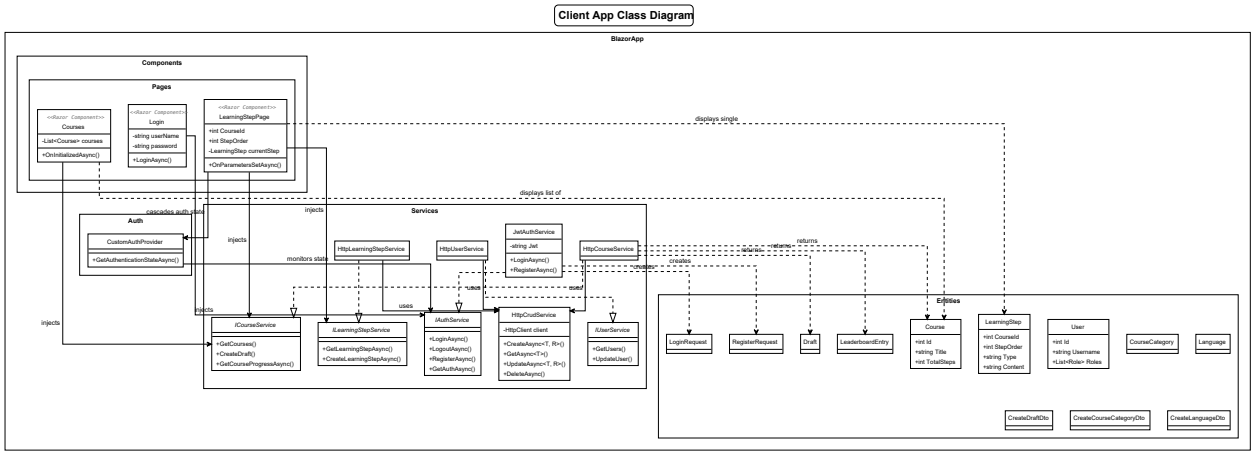

Figure 8: Client App Class Diagram

3.2.3.2 Logic Server Class Diagram In this server the logic of the system is defined through the controllers, allowing the client server to perform the actions requested by the client.

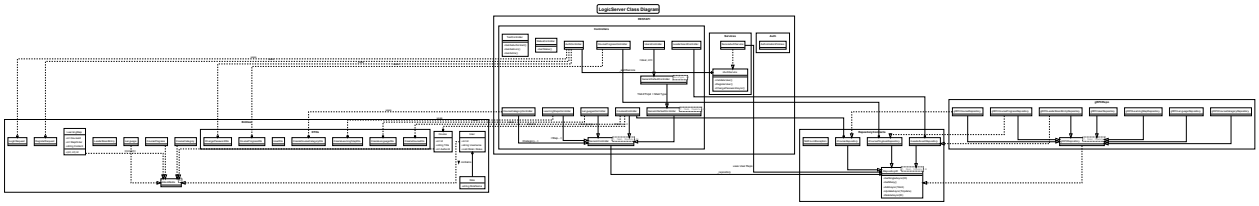

Figure 9: Logic Server Class Diagram

The diagram bellow shows the different entities needed in the logic server. All entities implement the IIdentifiable interface, achieving an abstract system where entities are easily replaceable.

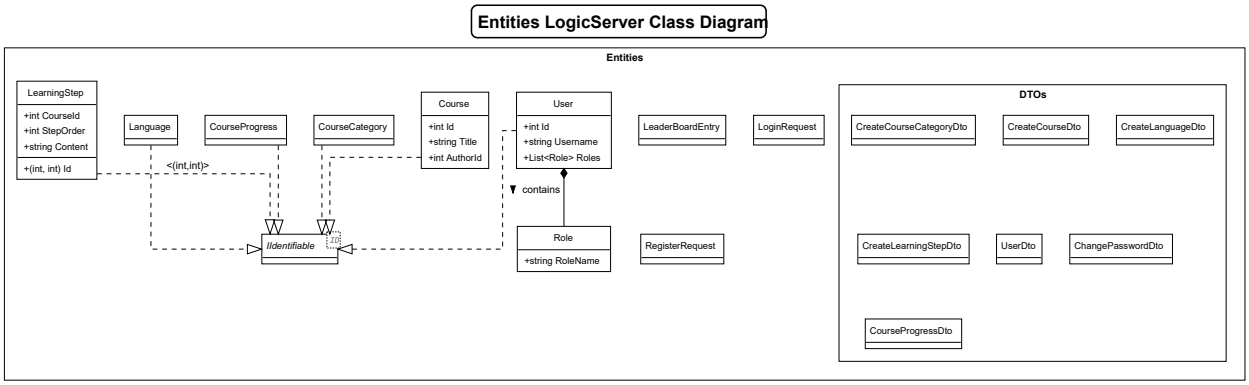

Figure 10: Entities Logic Server

The following diagram is a summarized easy-to-read Logic Server Class Diagram.

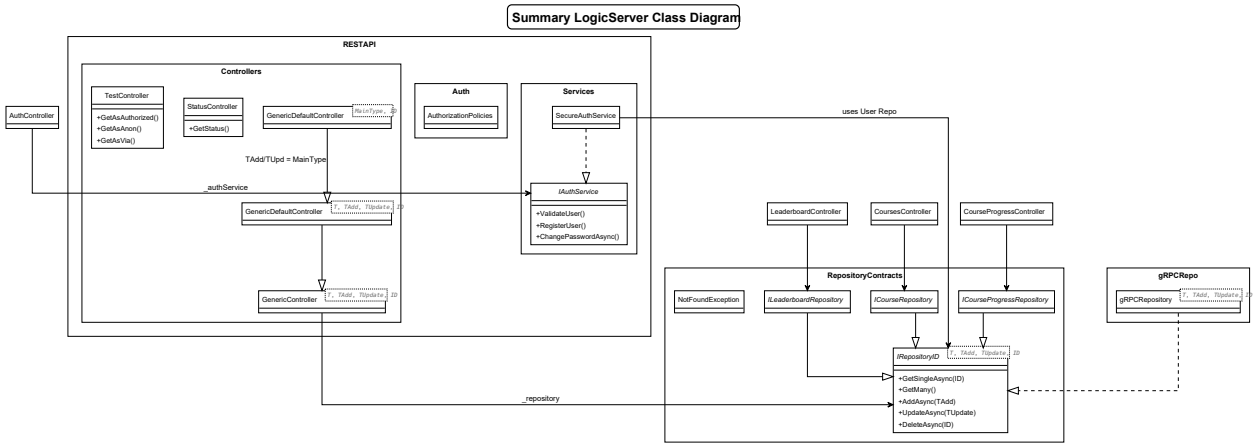

Figure 11: Summary Logic Server Class Diagram

3.2.3.3 Data Server Class Diagram This server main responsibility is to manage the database by adding, fetching, modifying and deleting the entities, ensuring that the logic server requests are completed successfully.

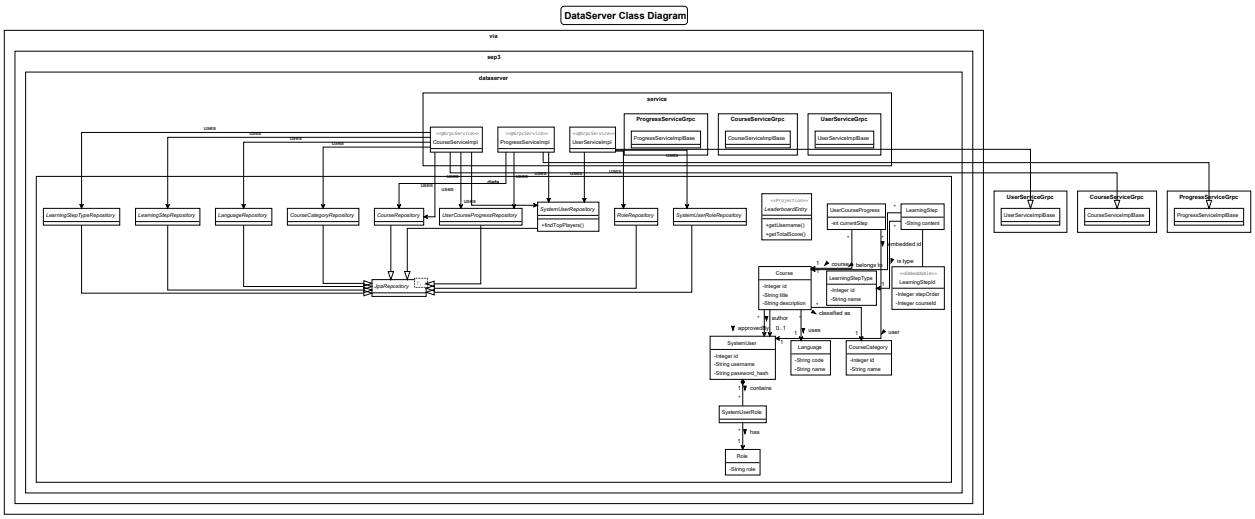

Figure 12: Data Server Class Diagram

In the next diagram the entities of the data server are exposed. It looks similar to the GRD since the system uses JPA, ensuring that the relationships in the data server's entities are transfered to the database,

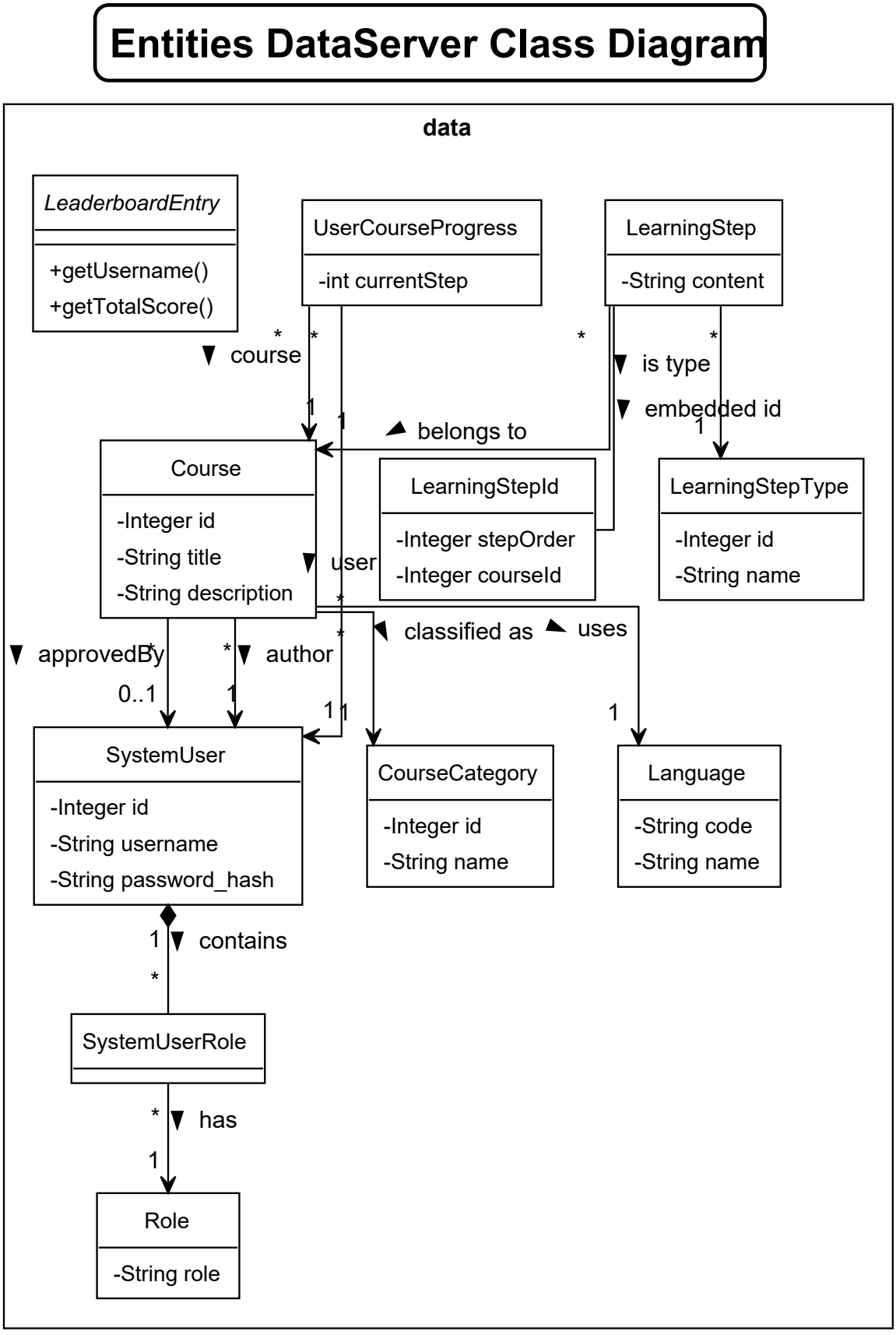

Figure 13: Entities Data Server

The following diagram displays a summary of the data server class diagram. Services and repositories can be seen on it.

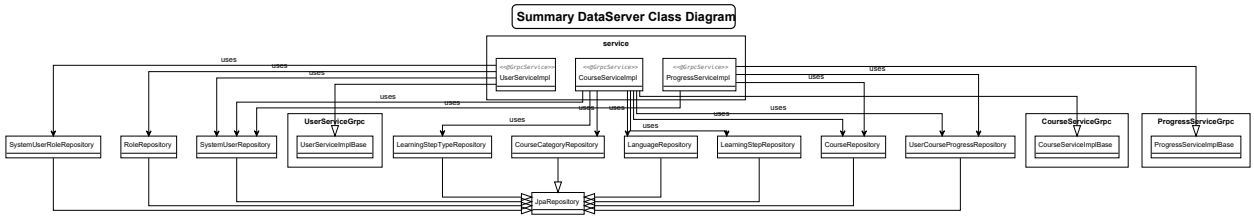

Figure 14: Summary Data Server

3.2.4 Communication protocol design

The system implements a multi-tiered architecture that utilizes distinct communication protocols for external and internal interactions. The sequence diagram below illustrates the end-to-end communication flow, demonstrating how the Client, Logic Server, and Data Server interact to process a request.

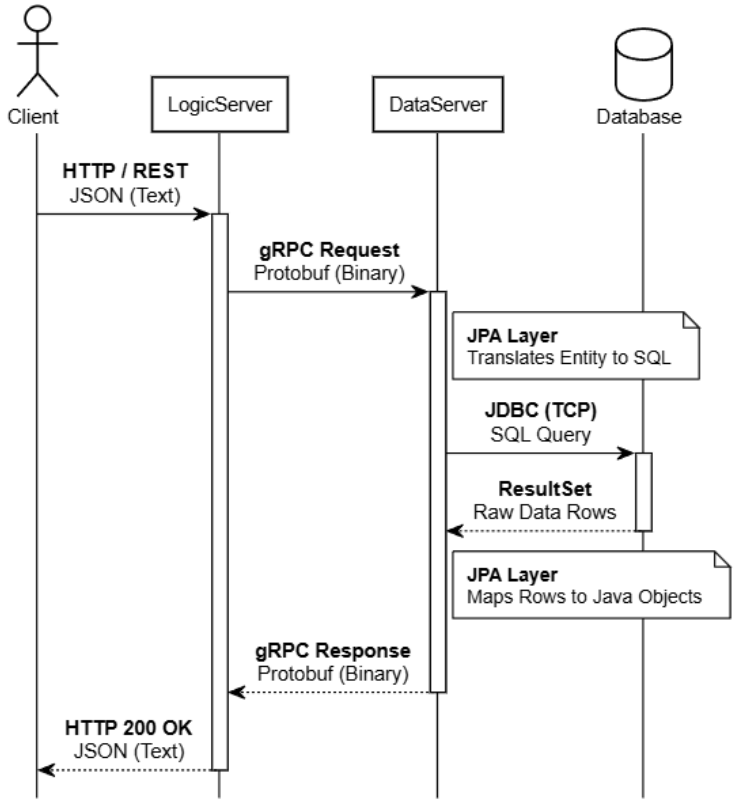

Figure 15: Application Layer Sequence Diagram

3.2.5 Security Design

Taking into consideration the earlier defined threat analysis and risk assessment, the security design focuses on the technologies used to defend against identified threats. System maintains various strategies of protecting the data at rest and while being transmitted.

3.2.5.0.1 Authentication and Authorization To address the threats of Spoofing and Elevation of Privilege, the System is based on stateless authentication architecture.

- Authentication: The System delegates auth operations to an AuthController. Once credentials are successfully validated, the System sends a JWT to the client. The System relies on JwtSecurityTokenHandler to sign the token using the HmacSha256Signature algorithm and a pre-configured secret key loaded though server level application settings.
- Authorization (RBAC): Access control is granted via ADDJwtBearer authentication method. The System inspects the claims of incoming requests to restrict access based on user roles (Learner, Teacher, Admin), effectively mitigating threats and unauthorized access.

3.2.5.0.2 Data Protection and Integrity To address Confidentiality and Integrity requirements, following strategies have been adopted to protect data throughout its lifecycle.

- Data at Rest: The system follows the principle of data minimization, ensuring no personal information is stored beyond the necessary authentication credentials (usernames and passwords). To reduce the impact of potential database leaks, passwords are never stored as plain text. The system utilizes the Argon2 algorithm. This hash algorithm provide resistance against brute-force attacks and can be considered the current state-of-the-art in password hashing (Password Hashing Competition, 2015).
- Data during Transit: To protect data against Man-in-the-Middle Attacks, protection strategies differ based on network exposure: Client to Server: The Logic Server enforces Transport Layer Security (TLS) via the app - UseHttpsRedirection() middleware. This ensures that user credentials are encrypted when being transferred over the public internet. Logic to Data: Communication between the Logic and Data servers occurs via gRPC. While this traffic remains unencrypted, the data is serialized in binary Protobuf format. This unencrypted state is considered acceptable for the current project scope as it assumes strict network isolation.

3.2.5.0.3 Input Validation To reduce the possibility of injection attacks the system validates users input. - The System's SecureAuthService is designed to handle user inputs and throw errors when invalid inputs such as mismatched passwords are detected, so that bad data is rejected before entering database.

- The system uses strict gRPC message typing (e.g., the AddUserRequest) to guarantee data structure. This ensures that injection attacks relying on malformed data structures or unexpected fields are impossible, as they are rejected by the protocol's strict binary validation before reaching the application logic.

3.2.5.1 Interface Definition (gRPC & Protobuf) Internal communication between the Logic Server and the Data Server is managed via gRPC. The data structures and service contracts are defined using Protocol Buffers (Protobuf), ensuring strict typing.

The next code snippet demonstrates the definition of the message structures (Requests and Responses) used within the system.

```
message User {
  int32 id = 1;
  string username = 2;
  string password = 3;
  repeated Role roles = 4;
}

message Role {
  string role = 1;
}

message GetUsersRequest {}
```

```
message GetUsersResponse {
  repeated User users = 1;
}
```

The next code snippet illustrates the service definition, detailing the RPC methods, their required parameters, and return types.

```
service UserService {
  rpc GetUsers(GetUsersRequest) returns (GetUsersResponse);
  rpc GetUser(GetUserRequest) returns (User);
  rpc AddUser(AddUserRequest) returns (AddUserResponse);
  rpc UpdateUser(UpdateUserRequest) returns (User);
}
```

3.2.5.2 API Specification The Logic Server exposes a RESTful API to external clients using standard HTTP/1.1 protocols. This design streamlines client integration by using standard HTTP verbs (GET, POST, PUT, DELETE) and status codes.

For example, authentication is handled via the /auth/login endpoint. By sending a POST request to http://domain:port/auth/login with the correct credentials, a client can successfully authenticate and connect to the system.

3.2.5.3 Protocol Justification A hybrid protocol approach was chosen to balance user experience with system performance, in specific, HTTP was chosen for the logic server and gRPC for the data server:

· External Communication (HTTP/JSON): HTTP was used with JSON for client-server interaction because of its universality and readability. JSON is natively supported by web browsers and mobile clients, making the system easy to debug and integrate. While the text-based format introduces some overhead, the trade-off favors the ease of development and broad compatibility required at the client layer.

· Internal Communication (gRPC/Protobuf): For communication between the Logic and Data servers, gRPC was selected over REST. Unlike the text-based JSON, gRPC uses Protocol Buffers to serialize data into a binary format. This results in significantly smaller payload sizes and faster serialization/deserialization times. Furthermore, gRPC operates over HTTP/2, allowing for multiplexing and lower latency, which is critical for high-throughput internal traffic.

3.2.6 Database design

3.2.6.1 Enhanced Entity Relationship Diagram As means of bridging the gap from the problem domain in general and the Learnify system in specific, an EER diagram was created as can be seen on the figure below:

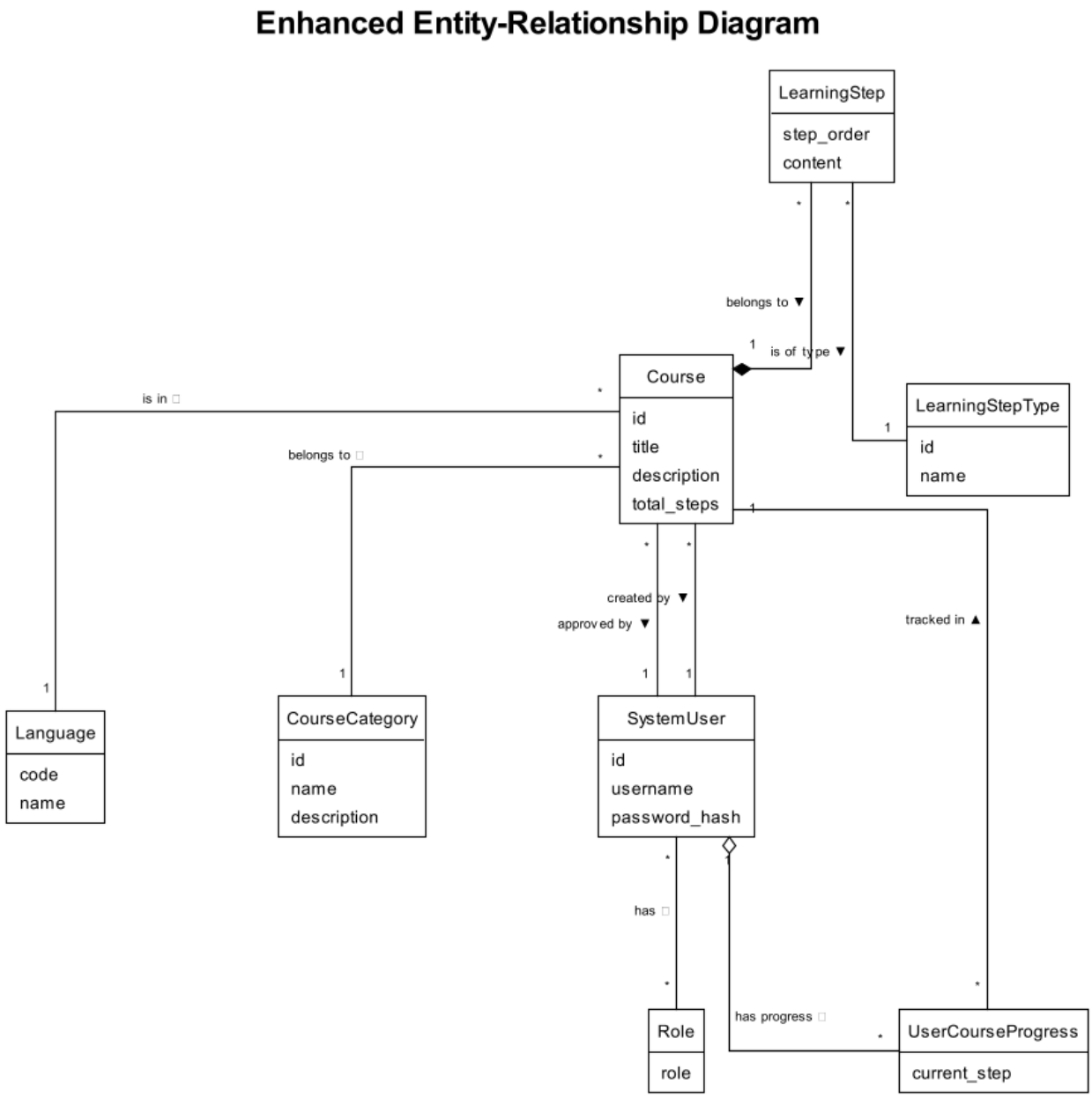

Figure 16: Enhanced Entity Relationship Diagram (Appendix 2.3 Diagrams)

The EER developed does not significantly differ from the domain model as both diagrams are conceptual and could in theory be used interchangeably. However, the EER diagram further reflects the decisions made during analysis, which most notably reflected on the way how roles are handled.

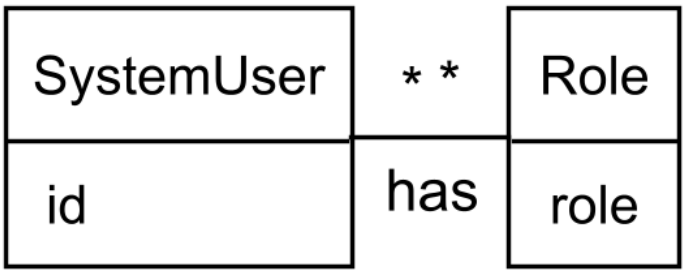

Figure 17: SystemUser to Role Relationship

The figure above is a part of the domain model (Appendix 2.3 Diagrams) and focuses on the relationship between the User and their Roles. This relationship in contrast to inheritance based models provides a flexible and strict way of handling user roles - their permissions and access to the system. Most importantly it does not hide the complexities of inheritance into a seemingly simple abstraction and prevents the potential issues that could arise from mindless inheritance hierarchies.

Certain relationships could in theory be kept as many-to-many, however the need to attach attributes to these relationships led to the preference of a separate entity to hold the attribute thus separating the concerns and providing a more flexible design. This can be seen on, for example, the UserCourseProgress entity as shown below:

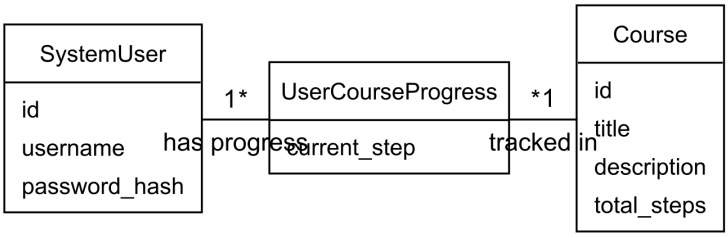

Figure 18: SystemUser to Role Relationship

3.2.6.2 Relational Schema Contrary to the conceptual modelling, the logical modelling required adherence to the rules and specificities of the relational model. Because of the designed use of standard relational database (PostgreSQL), the mapping of the EER needed to determine all the necessary resolutions of relationships, strong and weak entities, and the establishment of integrity keys.

The mapping of the EER diagram resulted in a relational schema and the to it related global relations diagram.

The mapping resulted in the relation schema as shown on the figure below:

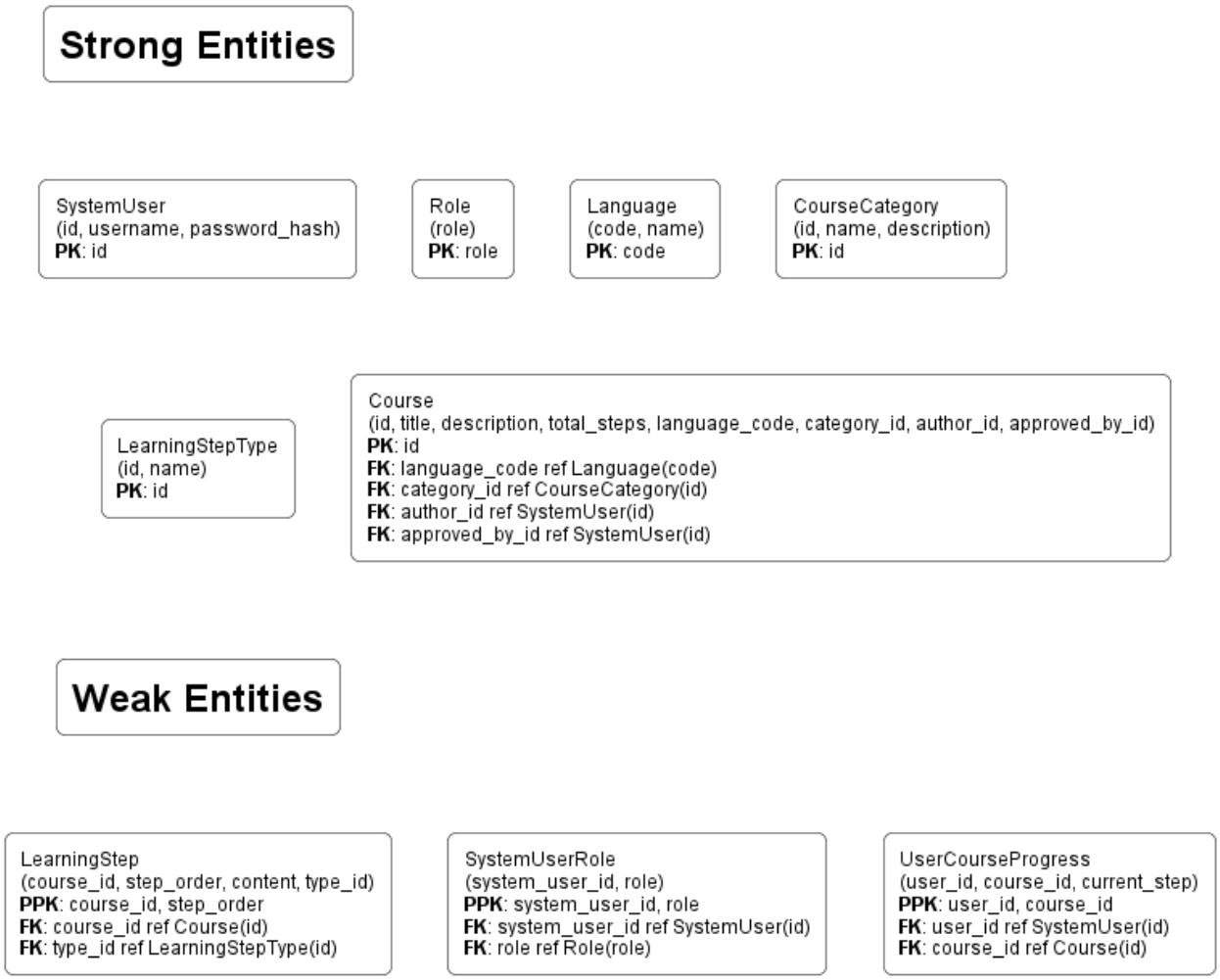

Figure 19: Relational Schema (Appendix 4.1 Relation Schema)

As can be seen below on the part of Global Relational Diagram (Appendix 2.3 Diagrams), the many-to-many relationships got resolved into new relations - SystemUserRole, and UserCourseProgress.

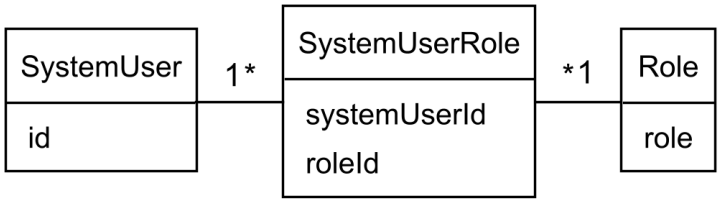

Figure 20: Caption

3.2.6.3 Global Relations Diagram (GR/GRD) Relational schema and Global Relations Diagram are technically identical. In the project Learnify, the GRD was the main source of truth for later implementation of the database structure and discussing the data persistence design.

The final GR diagram is shown on the figure below:

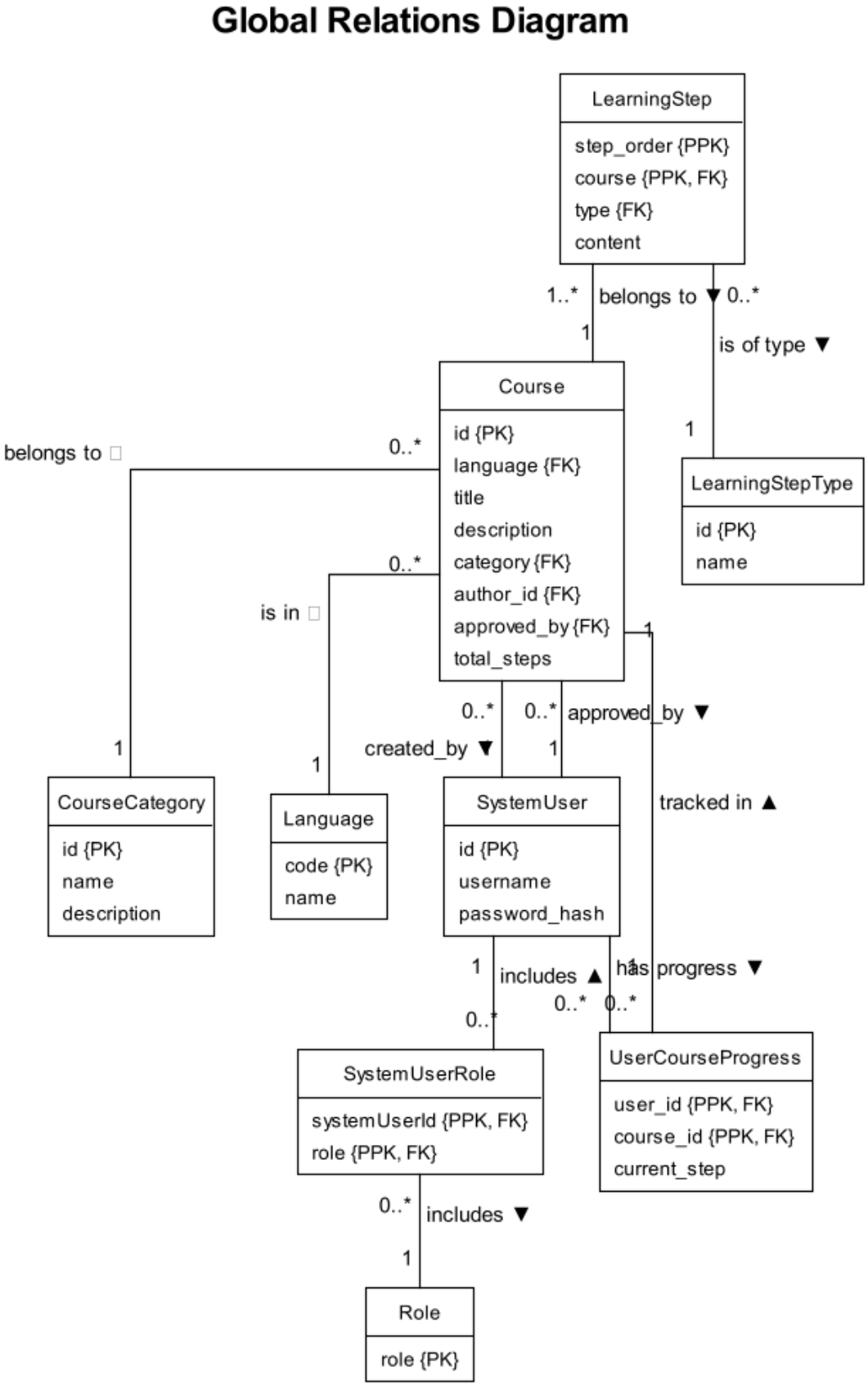

Figure 21: Global Relational Diagram (Appendix 2.3 Diagrams)

It can be seen that the GRD reflects the same concepts as the relational schema, however, in this case the positioning of the elements provides a more natural step going from the EER and the domain model; although, the positioning did not fully preserve the conceptual relationships as seen in the EER diagram.

3.3 Implementation

The implementation of the system followed the designed architecture and communication protocols with a focus on setting up the core of the architecture first as one continuous vertical slice.

At first, a database schema was created in PostgreSQL, a Springboot project was created for the Data Server, and two .NET solutions were created for the Logic Server and the Client Application.

This stage did not include any actual logic but rather provided a skeleton of the system. One of the decisions taken at this stage was to maintain separate solutions for the Logic Server and Client Application. Despite the initial idea of implementing a shared solution, it was decided that the feature of C# anonymous types would suffice for most of the purposes of data transfer objects (DTOs) and that the added complexity of a shared solution would not be justified.

The implementation of the skeleton was followed by the implementation of a vertical slice which focused on fetching all courses from the database.

The individual components of the vertical slice can be seen below on the part of Domain Model (Appendix 2.3 Diagrams):

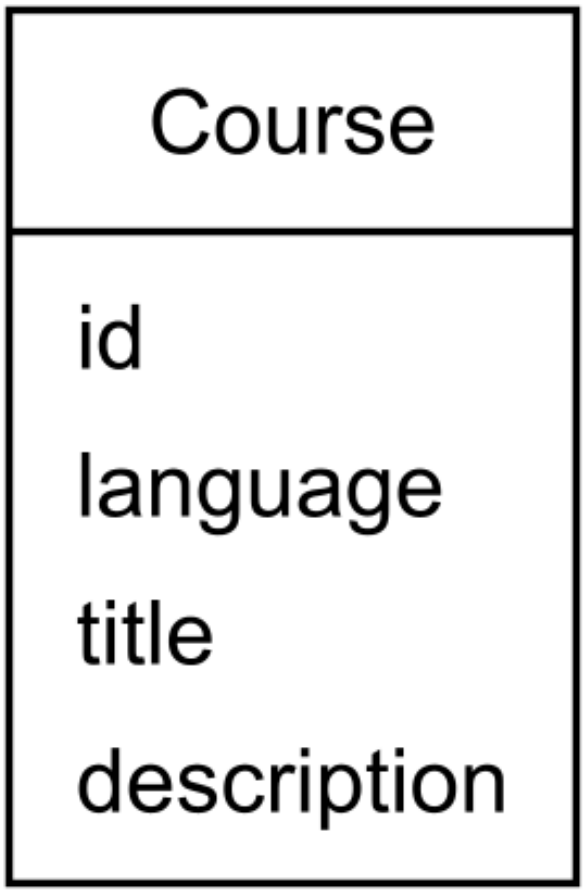

Figure 22: Caption

At a later stage, the original database setup was split into pure DDL script and a dummy data initial setup crucial for testing stages mentioned later in this document.

3.3.1 Methods and tools

Going further into implementation details, Java and C# programming languages were chosen to meet the requirements of multilanguage system. For storing information PostgreSQL database has been implemented. C# Blazor .NET was used for the frontend implementation because its component-based system gave a possibility for fast GUI development without needing an additional JavaScript code. On the Logic Server, ASP.NET Core provided an intuitive environment for managing REST endpoints and gRPC services, facilitating high-speed communication. It was achieved by using shared Data Transfer Objects (DTOs) as well as anonymous types and validation logic which kept data model changes synchronized between client and server systems. Java programming language was selected to run the Data Server because it met this semester's polyglot requirements and demonstrated how .NET and Java systems can work together using gRPC protocol, which is explained further in the integration logic paragraph below.

3.3.2 Servers Implementation

Data Server implemented using Java was intended to not have any code related to the main logic of the system. Its main responsibility was to handle operations of the services which were either taking data from the database or updating the database. The framework that was used in order to make the implementation process cleaner and more efficient was Spring boot. Spring Data JPA was chosen when it comes to handling data persistent with PostgreSQL due to the possibility of working with Java objects instead of raw SQL queries.

The Logic Server operated as the system's Web API which functioned as the core processing unit of the platform by using the C# ASP.NET Core framework. The Java server took care of database management while the Logic Server executed all business operations which enable the system to function properly. The system operated as a middleman between the Blazor client and the Data Server because it handled requests which followed system rules before sending data to the Data Server. It is worth mentioning that security was of high concern on this server. The system used JWT (JSON Web Tokens) to handle user authentication which restricted access to particular features based on user authorization. All the main logic was kept here which made it simple to control features such as course enrollments and leaderboard system or course draft approval workflow. Using C# for this layer was a great fit because it worked perfectly with the Blazor client on another server, allowing the code to be kept organized and easy to build on.

When it comes to Client Server, as mentioned before, Blazor C# was chosen. It gave a structured template to work on the frontend using reusable components and integrating logic using C# programming language instead of JavaScript. Client side was responsible for sending HTTP request to the WebApi through user friendly, GUI.

3.3.3 Integration Logic:

To showcase the path from GUI through the servers, the database, and back as well as communication between the servers, the following figures demonstrate the necessary logs and implementations of such functionality. The code can be found in (Appendix 3.1 Source Code).

3.3.3.1 Login Feature When a user who already has an existing account tries to log in, they input their credentials into the text field and clicks the login button.

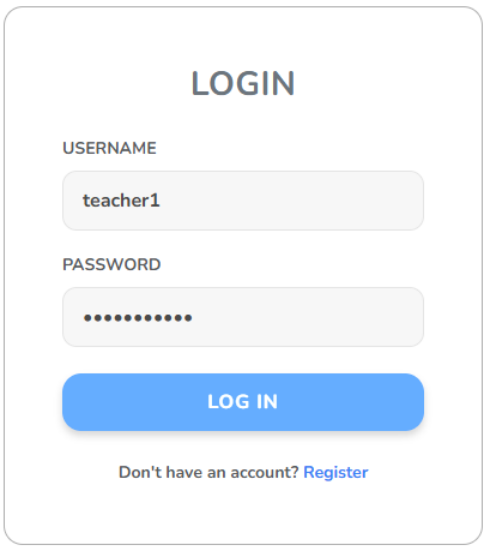

Figure 23: Click login as teacher

After clicking the login button, the client side server sends a request to the Logic Server. The following json

represents the HTTP request sent:

```
[Client] Sending HTTP POST /auth/login:
{
  "username": "teacher1",
  "password": "password123"
}
```

The Logic Server receives this request and initiates the validation process.

Logic Server (AuthController.cs):

```
[HttpPost("login")]
public async Task<ActionResult> Login([FromBody] LoginRequest request)
{
    try
    {
        User foundUser = await authService.ValidateUser(request.Username, request.Password);
        string token = GenerateJwt(foundUser);
        return Ok(token);
    }
    catch (Exception e)
    {
        return BadRequest(e.Message);
    }
}
```

Code Snippet 1: AuthController (Appendix 3.1 Source Code)

The `AuthController` delegates the validation to the `SecureAuthService`.

Logic Server (SecureAuthService.cs):

```
public async Task<User> ValidateUser(string username, string password)
{
    User? existingUser = await GetUserByUsernameAsync(username) ?? throw new Exception("User not found")

    if (!Argon2.Verify(existingUser.Password, password))
        throw new Exception("Password mismatch");

    return existingUser;
}

private async Task<User?> GetUserByUsernameAsync(string userName)
{
    IEnumerable<User> users = await Task.Run(() => userRepository.GetMany());
    foreach (User user in users)
    {
        if (user.Username.Equals(userName))
        {
            return user;
        }
    }
    return null;
}
```

Code Snippet 2: SecureAuthService (Appendix 3.1 Source Code)

The `SecureAuthService` retrieves users via the `gRPCUserRepository`, which communicates with the Data Server.

Logic Server (gRPCUserRepository.cs):

```
public override IQueryable<User> GetMany()
{
    var resp = UserServiceClient.GetUsers(new GetUsersRequest());
    var users = resp.Users.Select(c => new User
    {
        Id = c.Id,
        Username = c.Username,
        Password = c.Password,
        Roles = c.Roles.Select(r => new Entities.Role { RoleName = r.Role_ }).ToList(),
    }).ToList();

    return users.AsQueryable();
}
```

Code Snippet 3: gRPCUserRepository (Appendix 3.1 Source Code)

The Protobuf message structure used for this communication is:

```
message User {
  int32 id = 1;
  string username = 2;
  string password = 3;
  repeated Role roles = 4;
}

message Role {
  string role = 1;
}

message GetUsersRequest {}

message GetUsersResponse {
  repeated User users = 1;
}
```

Code Snippet 4: Protocol (Appendix 3.1 Source Code)

Finally, the Data Server handles the gRPC request and retrieves the users from the database.

Data Server (UserServiceImpl.java):

```
@Override
public void getUsers(GetUsersRequest request, StreamObserver<GetUsersResponse> responseObserver) {
    try {
        List<SystemUser> users = userRepository.findAll();
        List<via.sep3.dataserver.grpc.User> grpcUsers = new ArrayList<>();

        for (SystemUser user : users) {
            grpcUsers.add(convertToGrpcUser(user));
        }
        GetUsersResponse response = GetUsersResponse.newBuilder()
                .addAllUsers(grpcUsers)
                .build();
```

```
            responseObserver.onNext(response);
            responseObserver.onCompleted();
        } catch (Exception e) {
            responseObserver.onError(e);
        }
    }
```

Code Snippet 5: UserServiceImpl (Appendix 3.1 Source Code)

3.3.3.2 Create Draft Feature Once logged in, the teacher can create a course draft. This process involves the Client sending data to the Logic Server, which then forwards it to the Data Server.

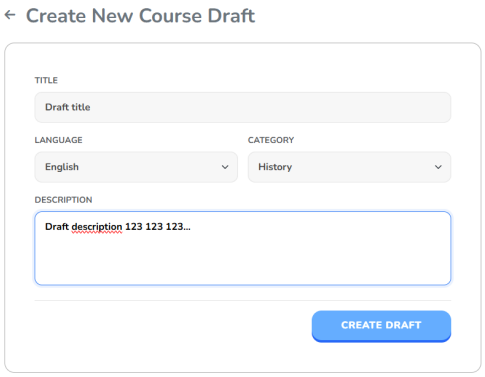

Figure 24: Click Create Draft

Client App (HttpCourseService.cs):

```
public async Task<Optional<Draft>> CreateDraft(CreateDraftDto dto)
    => await httpCrudService.CreateAsync<Draft, CreateDraftDto>("drafts", dto);
```

Code Snippet 6: HttpCourseService (Appendix 3.1 Source Code)

The Logic Server handles the request at the `/drafts` endpoint:

Logic Server (CoursesController.cs):

```
[HttpPost("/drafts")]
[Authorize("MustBeTeacher")]
public async Task<ActionResult<Course>> HttpCreateAsync([FromBody] CreateCourseDto entity)
    => await CreateAsync(entity);
```

Code Snippet 7: CoursesController (Appendix 3.1 Source Code)

The controller uses the `gRPCCourseRepository` to send the data to the Data Server.

Logic Server (gRPCCourseRepository.cs):

```
public override async Task<Course> AddAsync(CreateCourseDto entity)
{
    var request = new AddCourseRequest
    {
        Title = entity.Title ?? "",
        Description = entity.Description ?? "",
        Language = entity.Language ?? "",
        Category = entity.Category ?? "",
        AuthorId = entity.AuthorId ?? -1
```

```
    };

    var response = await CourseServiceClient.AddCourseAsync(request);

    return new Course
    {
        Id = response.Course.Id,
        Title = response.Course.Title,
        // ... mapping other fields ...
    };
}
```

Code Snippet 8: gRPCCourseRepository (Appendix 3.1 Source Code)

The gRPC messages for creating a course are defined as:

```
message AddCourseRequest {
  string title = 1;
  string description = 2;
  string language = 3;
  string category = 4;
  int32 authorId = 5;
}

message AddCourseResponse {
  Course course = 1;
}
```

Code Snippet 9: Protocol (Appendix 3.1 Source Code)

Finally, the Data Server persists the new course draft:

Data Server (CourseServiceImpl.java):

```
@Override
public void addCourse(AddCourseRequest request, StreamObserver<AddCourseResponse> responseObserver) {
    try {
        Course course = new Course();
        course.setTitle(request.getTitle());
        course.setDescription(request.getDescription());
        // ... additional field setting ...
        course = courseRepository.save(course);

        AddCourseResponse response = AddCourseResponse.newBuilder()
                .setCourse(convertToGrpcCourse(course))
                .build();

        responseObserver.onNext(response);
        responseObserver.onCompleted();
    } catch (Exception e) {
        responseObserver.onError(e);
    }
}
```

Code Snippet 10: CourseServiceImpl (Appendix 3.1 Source Code)

3.3.4 Security Implementation

Learnify bases its entire system security approach on the foundation of its Security Policy (Appendix 7.2 Security Policy). The document describes vital security measures which the system aims to employ to protect the information from unauthorized access and security breaches.

The System depends on three critical security principles which include confidentiality, integrity and availability that protect its core functions through multiple essential security measures.

The process needs each user to enter their personal login details for authentication. The policy specifies that users needs to create passwords which contain at least eight characters to safeguard their accounts. Role-Based Access Control (RBAC) provides additional protection through its access management system which grants specific permissions to Learners and Teachers and Administrators based on their designated roles.

The Logic Server performs user role verification through generated claims to authorize only permitted actions before processing any requests. The endpoints of WebApi logic server have been secured. Focus was also on achieving data security by using a well-planned system which organizes information through classification and protects it with encryption methods.

The system contains three types of data which include:

- public information that users can access through registration and login pages,
- internal information that requires authentication to view course catalogs and content, and
- sensitive information which needs encryption for user passwords.

The system uses Argon2 as a secure password hashing and salting algorithm which protects user passwords. The system operates with continuous security measures and regular software updates to maintain its network security. It functions through a firewall which grants access to particular ports that are essential for operation.

3.3.5 Deployment

A deployment diagram was created to visualize the physical deployment of the system components. The diagram illustrates how the system components are dependent on each other and how they interact once deployed. The diagram can be found as Appendix 13.1 Deployment Diagram or seen on the figure below:

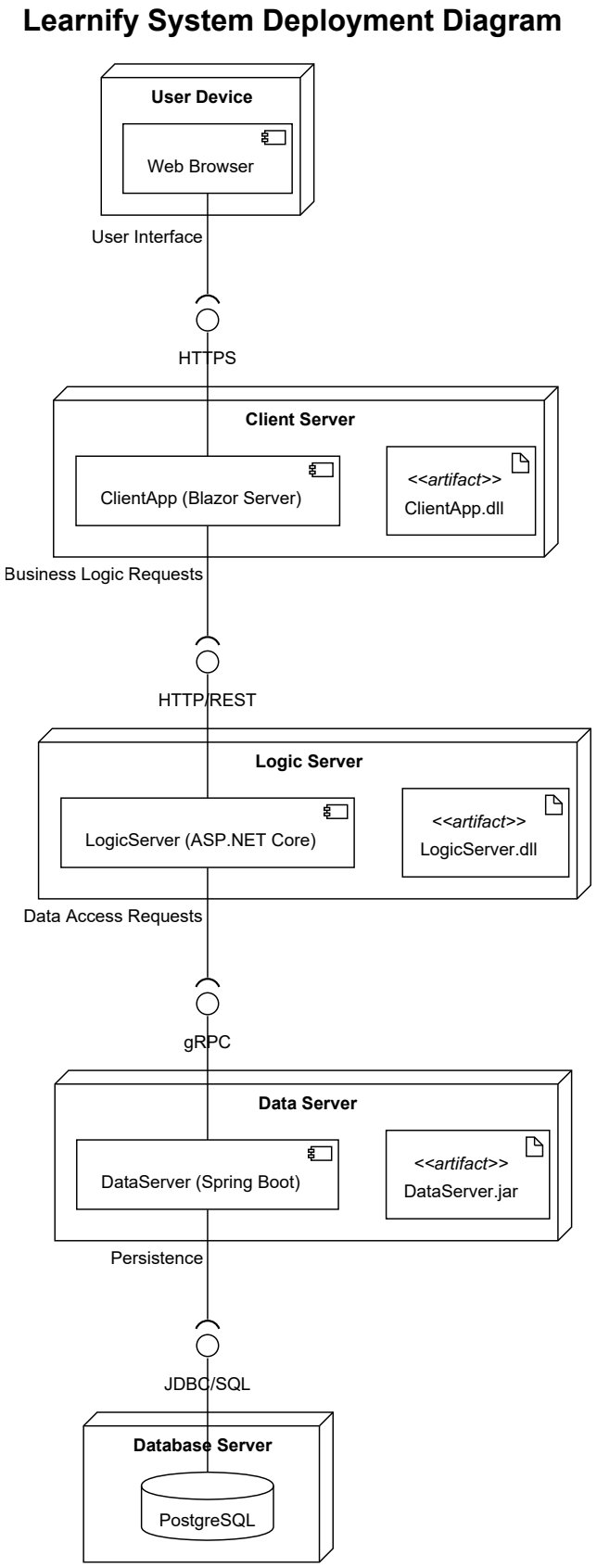

Figure 25: Deployment Diagram (Appendix 13.1 Deployment Diagram)

3.4 Testing

3.4.1 Testing Approach

Testing in this project was structured around the V-Model (Sommerville, 2016), where testing activities are planned in parallel with developments phases. Instead of treating testings as a final step, test considerations were introduced early, starting at the requirement analyses, and refined as the system design evolved. This approach ensured that development decision had a corresponding verification strategy.

The first step of the V-Model was requirement analysis, which directly maps to acceptance testing.

At this stage, testing focused on the question of the ability to determine whether the system fulfills the user requirements.

Rather then writing code-level tests, high-level acceptance criteria were defined for each use case. These criteria described:

- What user expects to achieve
- Under which conditions the system should allow or deny actions
- What outcome confirms that the requirement is fulfilled

These acceptance-oriented test ideas were later used to validate the system both manually and through automated test. In this way, the test cases were already embedded in the analysis phase.

In practice, different types of tests naturally aligned with different parts of the V-Model:

- High-level test cases and use-case validation corresponded to the upper part of the V (requirements and acceptance testing).
- Automated tests focused mainly on the lower part of the V, especially unit and integration testing.

3.4.2 Tools and frameworks

A combination of automated and manual testing tools was used throughout the project:

- xUnit – Used for writing automated unit and integration tests.
- HTTP client–based tests – Used for endpoint-level testing (Postman).
- .http files – Used early in development for manual REST endpoint testing.
- BloomRPC – Used for manually testing gRPC endpoints.
- In-memory repositories – Used to test logic without external dependencies.
- IDE tooling (Visual Studio / IntelliJ) – Used for writing, executing, and debugging tests.

3.4.3 What was tested

Testing focused primarily on core logic and system-critical behavior, rather than attempting full coverage of all UI components.

The following aspects of the system were tested:

- Business logic in the logic server
- REST and gRPC endpoints
- Authentication and authorization behavior
- Role-based access control
- Course and draft management workflows
- Integration between services

The aim was for high endpoint coverage, testing endpoints under different scenarios such as:

- Different user roles and credentials
- Authorized vs. unauthorized access
- Expected success paths

However, testing of faulty or malformed input data was less comprehensive. While access control and permissions were thoroughly tested, negative scenarios involving invalid data were sometimes under-tested. This limitation is acknowledged as an area for improvement.

3.4.4 Method-level test case documentation

Method-level testing focused on logic-heavy and high-impact methods, following a white-box testing approach. Instead of testing every method, priority was given to areas where errors would have the greatest impact on system behavior.

For critical methods, tests were designed to cover:

- Normal execution paths
- Boundary conditions
- Exceptional cases
- Interface correctness

This approach ensured that the internal behavior of key methods was thoroughly validated, not just their external outputs. Test cases were documented to show clear intent and traceability between requirements, logic, and expected outcomes.

The table of all test case results can be found in (Appendix 2.4 Tests).

3.4.5 Benefits and bug detection

Testing played an important role in maintaining system stability throughout development and provided confidence when introducing changes.

One of the main benefits of automated testing was that it enabled safe refactoring. Because core logic and most endpoints were covered by tests, the codebase could be restructured and improved without the constant risk of breaking existing functionality. This was especially valuable in later stages of the project, where refactoring became more frequent as the system matured.

In several cases, features appeared to work correctly when tested manually through the user interface, but automated tests revealed issues that were not immediately visible. These included:

- Missing validation for unexpected input
- Edge cases and boundary values not being handled correctly
- Operations failing silently without clear feedback

A continuous test–fix–verify cycle was followed during development. When a problem was discovered, it was corrected, committed through version control, and often accompanied by additional tests to prevent similar issues in the future. Although the project did not strictly follow Test-Driven Development from the start, the gradual shift toward automated testing significantly improved reliability, reduced regressions, and supported ongoing changes as the project evolved.

The red-green-refactor cycle also played its role. The implementation often followed this patterns where a failing test (red) would be written first, followed by the implementation to make it pass (green), and finally refactoring the code for clarity and maintainability while keeping the tests passing (refactor).

3.5 Result

Based on the defined use cases and their corresponding test scenarios, the system was evaluated through a combination of automated tests and manual verification. The implemented functionality covers the main workflows for learners, teachers, and administrators that were within the project scope.

The results show that most implemented use cases behave according to their expected outcomes. Core features such as user registration and login, learning activities, course creation, and administrative management were verified to function correctly.

Regarding security-related aspects, the system communicates over HTTP during development and testing. While the application configuration includes redirection to HTTPS outside of development mode, a full HTTPS setup with proper certificate handling was not established as part of this project. Additionally, no explicit enforcement of a minimum password length (such as an 8-character requirement) is implemented in the current system.

The table with all test cases can be found as (Appendix 2.4 Tests) - the table provided there also contains the test case results as these test cases were used during the testing phase as well.

4 Discussion

4.0.0.1 What Has Been Accomplished The main purpose of this project has been achieved through the creation of the "Learnify" distributed heterogeneous e-learning system which targets the problem of educational inequality.

The system operates through a three-tier architecture which combines a C# Blazor client application with a Logic Server built in C# .NET and a Data Server developed in Java. The PostgreSQL database functions as the storage system which maintains all relevant information for every service.

The system architecture uses gRPC protocol to establish fast communication between services while it employs HTTP protocol to enable client application interaction.

The system has achieved full addressment of 18 specific user stories which support the complete learning process.

The system enables users to:

- create new accounts through its registration feature.
- browse course catalogs.
- take part in courses.

The system provides users with suitable interactive learning activities that deliver instant feedback. It also enforces a rigid role-based structure which applies to all users including Learners, Teachers and Administrators. The role hierarchy establishes a governance process which requires Teachers to obtain administrator approval before their content can be published.

The security implementation utilizes industrial standards with Argon2 for Password Hashing with salting and JWT for Stateless Authentication. In addition, the User Experience has been strengthened with the incorporation of leaderboards for gamification and a User Interface for color-deficient users.

The overall requirements have been designed in accordance with the primary values laid down in the project and stakeholder feedback.

4.0.0.2 What Can Be Improved Even with the successful implementation of the fundamental system, there are some aspects that need optimization towards the reduction of the identified security risks and the improvement of scalability.

The basic application needs additional security measures to protect against advanced attacks even though it uses Argon2 password hashing and JSON Web Tokens. The Critical risk of unpatched software vulnerability exploitation requires automated scanning and effective patch management for future developments. The protection of high-risk systems against Man-in-the-Middle attacks and credential exploitation needs Transport Layer Security/Secure Sockets Layer combined with HTTP Strict Transport Security and Content Security Policies to prevent Cross-Site Scripting attacks and Multi-Factor Authentication.

The system has passed normal functional tests to meet operational requirements but its performance under severe stress remains untested through extreme load testing. The risk assessment shows that Distributed Denial of Service requires future research to simulate high user volumes for identifying system bottlenecks.

The testing also significantly failed to address certain parts of the system and unit tests like the InMemoryRepository test suite only proved their functionality once their failure started affecting other tests relying on them.

In addition to testing, compiler feedback also went unaddressed for many parts of the system despite already showing potential future issues.

Concerning user experience, improvement can be done towards the inclusivity of the platform. Although the support for color vision disabilities is a good starting point, the support needs to be extended to implement full compatibility with the Web Content Accessibility Guidelines and screen readers for users with motor disabilities.

In addition to this, the format of the learning model needs to change from the linear pattern that it currently supports to an adaptive path and also an advanced gamification system incorporating features like badges and streaks.

One of the flaws and features that were not implemented in the project related to the performance testing is also the performance itself. Even though the implementation fully allowed for simple caching and other performance optimization techniques (via the proxy pattern and other partially implemented solutions), such optimizations were never made as their need went virtually unnoticed.

Lastly, to avoid administrative delays that could arise in the future due to the expansion of users, it is advised that the manual content reviewing system should be integrated with an AI-powered content review system.

5 Conclusion and Recommendations

The primary objective of the Learnify project was to develop a scalable, heterogeneous, and distributed e-learning system designed to mitigate educational inequalities. This was successfully achieved through a three-tier architectural design that leveraged a polyglot approach, combining C# (Blazor and ASP.NET Core) with Java (Spring Boot) and a PostgreSQL database. A key technical success was the implementation of a hybrid communication protocol—utilizing REST for external client-server interaction and gRPC for high-performance internal data exchange—which ensured both accessibility and system efficiency.

The system fulfilled its core functional requirements, including user registration, course management, and an interactive learning loop. Notably, the project met its non-functional requirement for inclusivity by delivering a color-blind friendly interface. However, the full vision for system integrity was partially constrained as automated content moderation features remained unimplemented within the project timeframe, necessitating manual oversight by administrators.

Future development should prioritize the following areas:

1. Enhanced Security and Robustness: Implementation of Multi-Factor Authentication (MFA) and Transport Layer Security (TLS) across all tiers is essential. Future iterations must also undergo rigorous stress and load testing to mitigate risks associated with Distributed Denial of Service (DDoS) attacks and to ensure low latency under high user volumes.
2. AI-Powered Moderation: To resolve the bottleneck of manual content review, integrating AI-driven moderation tools would allow for scalable and immediate quality control of teacher-submitted drafts.
3. Advanced Inclusivity and Gamification: While color-blind support is present, full adherence to WCAG standards, including screen reader support and motor-disability adaptations, is required. Furthermore, expanding the current leaderboard into a comprehensive gamification suite with badges and learning streaks would further enhance user engagement.
4. Adaptive Learning Paths: Moving beyond linear course structures toward AI-driven content recommendations would better address the “digital divide” by tailoring educational experiences to individual learner needs.

In conclusion, Learnify establishes a robust foundation for a distributed learning system. By addressing the identified security, performance, and accessibility gaps, the platform has the potential to significantly improve

the technological infrastructure of educational institutions and provide a more equitable digital learning environment.

6 References

- Haleem, A., Javaid, M., Qadri, M. A., & Suman, R. (2022). Understanding the role of digital technologies in education: A review. Sustainable Operations and Computers, 3(1), 275–285. https://doi.org/10.1016/j.susoc.2022.05.004
- Password Hashing Competition. (2015). Password-Hashing.net. https://www.password-hashing.net
- Project Description.
- PTI. (2025, March 2). 40% global population doesn't have access to education in language they understand: UNESCO. Deccan Herald. https://www.deccanherald.com/world/40-global-population-doesnt-have-access-to-education-in-language-they-understand-unesco-3428194
- Samonas, S., & Coss, D. (2014). The Cia Strikes Back: Redefining Confidentiality, Integrity and Availability in Security. In Journal of Information System Security (Vol. 10, Issue 3). https://www.proso.com/dl/Samonas.pdf
- Siemens, G. (n.d.). Connectivism: A Learning Theory for the Digital Age. https://static1.squarespace.com/static/6820668
- Habermas, J. (1984). The theory of communicative action: Vol. 1. Reason and the rationalization of society (T. McCarthy, Trans.). https://teddykw2.wordpress.com/wp-content/uploads/2012/07/jurgen-habermas-theory-of-communicative-action-volume-1.pdf
- UNESCO. (2000). The Dakar framework for action: Education for all: Meeting our collective commitments. UNESCO Digital Library. https://unesdoc.unesco.org/ark:/48223/pf0000121147
- Sommerville, I. (2016). Software Engineering Tenth Edition. https://dn790001.ca.archive.org/0/items/bme-vik-konyvek/Software%20Engineering%20-%20Ian%20Sommerville.pdf
- Roques, A. (n.d.). Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams. PlantUML.com. https://plantuml.com/

7 Appendices

7.1 Appendix 2.1 Requirements

7.1.1 Requirements

Can be found as requirements.png

7.2 Appendix 2.2 Use Cases

7.2.1 Use Case Diagram

Can be found as UseCaseDiagram.svg ### Complete Learning Step Use Case Description Can be found as CompleteLearningStepUseCaseDescription.png

7.3 Appendix 2.3 Diagrams

7.3.1 2.3.1 Activity Diagrams

Can be found in the Activity Diagrams folder ### 2.3.2 System Sequence Diagrams Can be found in the System Sequence Diagrams folder ### Domain Model Can be found as DomainModel.png ### Enhanced Entity Relationship Diagram Can be found as EER.png ### Global Relational Diagram Can be found as GR.png ### Application Layer Sequence Diagram Can be found as Application-LayerSD.png

7.4 Appendix 2.4 Tests

7.4.1 Test Cases

Can be found as Tests.xlsx

7.5 Appendix 3.1 Source Code

7.5.1 AuthController.cs

Can be found as System/LogicServer/RESTAPI/Controllers/AuthController.cs ### SecureAuthService.cs Can be found as System/LogicServer/RESTAPI/Services/SecureAuthService.cs ### gRPCUserRepository.cs Can be found as System/LogicServer/gRPCRepo/gRPCUserRepository.cs ### data_protocol.proto Can be found as System/DataServer/DataServer/src/main/proto/data_protocol.proto ### UserServiceImpl.java Can be found as System/DataServer/DataServer/src/main/java/via/sep3/dataserver/service/UserServiceImpl.java ### HttpCourseService.cs Can be found as System/ClientApp/BlazorApp/Services/HttpCourseService.cs ### CoursesController.cs Can be found as System/LogicServer/RESTAPI/Controllers/CoursesController.cs ### gRPCCourseRepository.cs Can be found as System/LogicServer/gRPCRepo/gRPCCourseRepository.cs ### CourseServiceImpl.java Can be found as System/DataServer/DataServer/src/main/java/via/sep3/dataserver/service/Cou

7.6 Appendix 4.1 Relation Schema

7.6.1 Relational Schema

Can be found as RelationalSchema.png

7.7 Appendix 7.1 Threat Model

7.7.1 Threat model

Can be found as ThreatModel.pdf

7.8 Appendix 7.2 Security Policy

7.8.1 Security Policy

Can be found as SecurityPolicy.pdf

7.9 Appendix 9.1 Wireframes

7.9.1 All Courses Page Wireframe

Can be found as Wireframes-2.png

7.10 Appendix 10.1: Stakeholder Interviews

7.10.1 Interview David

Can be found as Interview_261125_1.pdf ### Interview Andrej Can be found as Interview_121025.pdf

7.11 Appendix 11.1 Architecture

7.11.1 Architectural Overview

Can be found as ArchitecturalOverview.png

7.12 Appendix 13.1 Deployment Diagram

7.12.1 Deployment Diagram

Can be found as DeploymentDiagram.svg

VIA University College

Learnify - Process Report

Semester Project 3

Group 3

Students

Guillermo Sanchez Martinez (355442)

Piotr Wiktor Junosz (355502)

Halil Ibrahim Aygun (355770)

Alexandru Savin (354790)

Eduard Fekete (355323)

Supervisors

Joseph Chukwudi Okika (JOOK)

Jakob Trigger Knop (JKNR)

Character Count: 33387

Word Count: 5379

Software Technology Engineering

3rd Semester

December 18, 2025

Contents

1 Introduction

The project of a Learning Platform was initiated with the goal of creating an accessible and user-friendly online environment for learners. This project focused on developing a distributed system in a team environment, leveraging various distinct technologies and methodologies to ensure effective collaboration and successful delivery.

---

2 Group Work

Many of our group processes stemmed as a continuation from our previous project experience, because our current group consisted of the same five members. For this reason, we continued with improving our collaboration based on E-estimate personal profiles completed in the previous semester. Knowing the fact that our profiles consisted mostly of dominant red and blue working style and our last experiences we found a way of working really efficiently by creating clear todo lists and working mostly in smaller sub-groups. The high number of "Red" profiles unfortunately meant also that we were prone to conflict.

Figure 1: Personal Profiles (Appendix 6.1 Personal Profiles)

Our team was culturally diverse as we come from: Slovakia, Poland, Moldova, Spain and Turkey. The Southern European members of our team exhibited flexible relationship-based communication which proved to be a good balance for the structured task-oriented methods used by Central and Eastern European members. In addition, our earlier experiences together enabled us to handle cultural differences better because we understood each team member's work habits which helped us predict problems and address personal challenges before they could harm the team's performance. For example, knowing who preferred straightforward communication versus who tended to procrastinate with the set tasks allowed us to motivate each other effectively while maintaining a healthy working balance.

We based the group contract (Appendix 1.1 Group Contract) for the project on our previous contract with slight changes, for example to the point system. This group contract outlined our core values, purpose, expectations, conflict resolution strategies, and accountability measures in the team. While social loafing still occurred in the group, the team tried to handle it in a different way compared to having super strict contract with a lot of penalties in it. Unfortunately, this approach did not work well, thus many serious talks took place within the team, including warnings about the necessity of supervisor interventions if performance of the member did not improve.

2.0.1 Professional Collaboration

To document our ability to independently take part in professional collaboration, we adopted industry-standard practices such as using git. We used branching with Pull Requests to review code, ensuring no single person could break the main build. More details about use of version control can be found in the UseOfVersionControl (Appendix A: UseOfVersionControl.pdf). We communicated asynchronously via Discord/Teams to respect deep work time.

An important feature of the project's process was the role of a Product Owner (PO). The PO was responsible for managing the user stories, prioritizing the backlog, and ensuring that the team was aligned with the project goals. This role was crucial in maintaining a clear vision and direction for the project.

Contrary to our previous experience, this role was set to be more settled and less rotating among team members. This was done to ensure consistency and a clear point of contact for the team regarding project requirements and priorities. The role of the PO was assigned democratically at the start of the project.

---

3 Project Initiation

The initial ideas we worked with included a Health Assistant, Bank System, and the Learning Platform. These were chosen as relevant candidates from a brainstorming session based on their potential impact and feasibility. After discussing the vision and scope of each idea, we held a democratic vote to select the final project. The Learning Platform was chosen due to its clarity of purpose and alignment with our skills and interests.

The alignment on the actual vision was more complicated than the initial idea selection. We had to ensure that everyone in the team agreed on the project's goals, target audience, and key features. This involved several discussions and compromises to reach a consensus that satisfied everyone.

One of the problems we faced during the initiation phase was deciding on the actual scope of the project. We were all juggling between thinking about it as an actual ambitious product versus a simpler prototype suitable for the course requirements. This problem was resolved by agreeing on scalable and flexible solutions that allow for both simple approaches and potential complex upgrades.

Another challenge was approaching the problem in a data-driven manner. Considering our mostly European backgrounds, we had to ensure that the project's aims were relevant and applicable on a global scale by researching, and not relying solely on our personal experiences and assumptions. This also included exploring various perspectives and shifting some of the focus on minorities and underrepresented groups in education. This aligned with our mission to improve the education, rather than just having business goals focusing on making profit off of profitable majorities.

3.1 Choice and Alignment on Framework

Shortly after the group formation, a process brainstorm session was hosted, where most details of the desired framework were discussed and the Product Owner was selected. The full export of the session can be found as Appendix 12.1 Process Brainstorm.

---

4 Project Execution

The way of working in the project was a radical shift from our previous experiences. We adopted various aspects of Agile methodologies that fit our desired workflow. This included sprints, regular meetings, pair programming, Kanban but also our own adaptations to these practices.

Our main workflow was initially communicated and written down in the (Appendix A: Project Guidelines). This document served to outline our processes and roles but also aligned us on semantics and definitions of key concepts. Even though this document served as our aim and theoretical perfection we could aim for, perfect adherence was not always possible and advantageous.

One of crucial aspects of our work was embracing asynchronicity, which might contradict with our vision of pair programming but was seen collectively as the means to achieving better productivity and work-life balance. Asynchronicity was also implemented on a level of subgroups - fx. different feature groups would collaborate within the group in real time while asynchronously reviewing and agreeing with the rest of the team on other issues.

4.1 Before Project Period

Because of daily school and work commitments, we mostly worked asynchronously in this period. Our sprints were one week long, starting and ending on Wednesdays. Each sprint would start around lunchtime as we wanted to have time to end the previous sprint collaboratively. Each sprint would start with a planning meeting where we would discuss the goals and tasks for the sprint. At the end of each sprint, we would review all features and tasks, and discuss any blockers or challenges faced during the sprint.

4.2 Project Period

Our work during the project period looked as follows:

4.2.1 Week perspective

We met every working day, weekends voluntary individually.

4.2.2 Day perspective

We preferred to work remotely as this reduced commuting time, allowed for more flexible working hours, but also prepared us for future remote work.

Each day started at 8:20 with a daily standup meeting where we discussed for each: - What was done - What will be done - Any blockers

After the standup, we would plan the day's work, which we considered one sprint in this dense period. Based on our framework, the purpose and focus of a sprint is to merely label and agree on which aspects of the Kanban board we focus on during the sprint. Contrary to Scrum, not all focus was put on putting tasks from TODO to DONE, but rather on deciding how far to push each task (allowing reviews to be done in a different sprint and similar).

Depending on the approach of each sprint, we would either be in a group call or split into channels for pair programming. With more iterations, we realized that collective calls were more effective as they allowed for quicker communication and synchronization on key issues; often problems would reappear for multiple people and having everyone listen to the solution saved time in the future.

Each day the calls lasted until around 16:30 with breaks for lunch and short breaks in between. Depending on everyone's situation, some days the work would continue later into the evening/night for those who preferred that.

4.2.3 Feature perspective

Working on a feature was usually for 1-3 people subgroups. The feature was started by discussing the definition of done, breaking down the tasks, and creating a branch for it. Each feature was practically a vertical slice of the product and the work was also often split into vertical slices (in order to reduce interpersonal dependencies). The aim of each feature team was to deliver a working feature compatible with main (which was evolving in parallel) by the end of the feature work. Each feature would have to be reviewed by someone least biased before merging to main.

Working with the team usually involved calling, sketching ideas, and drawing UML diagrams, while researching domain context and processing data from users. On a practical level, we used Visual Studio Code with Live Share for pair programming, GitHub for version control, Figma/FigJam for brainstorming and unrestricted diagramming, PlantUML inside VSCode for quick UML sketches, and Discord for screen sharing and communication.

4.3 Adherence to the Framework

It must be noted that the adherence was not perfect. One such example is periodical claiming of tasks that often happened with no reason, which resulted in a so-called "Code Ownership Trash Can" where labels were put from tasks that did not need to be reserved by anyone. This can be seen on the figure below:

Figure 2: Code Ownership Trash Can

Another important point is the splitting of the work and the adherence to Kanban in critical situations - like before deadlines. IN these situations we resorted to more chaotic working as can be seen on the figure below, depicting a cumulative clump of tasks that arised towards the end of the project:

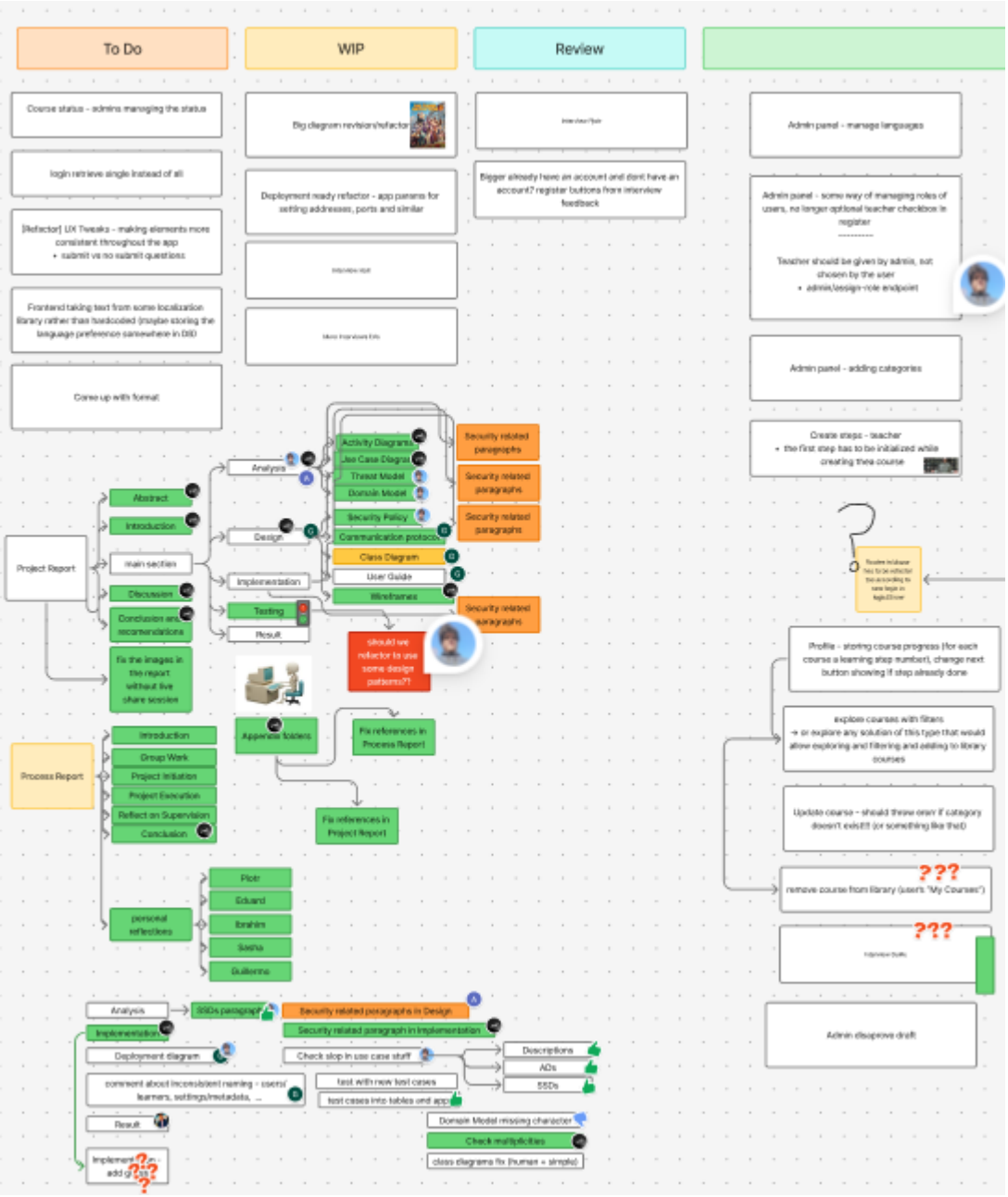

Figure 3: Tasks Before The Deadline

5 Choice of Tools and Methods

5.1 Task Decision and the PO Role

One of the most often used techniques for determining the priority of tasks was utilizing a board with Impact and Effort axes. Despite a lack of public information about such usage - in time restricted situations, the

tasks were prioritized from left to right, and in periods with time to spare - top to bottom. Overall, this method proved to be an effective planning technique that allowed the PO to quickly and relatively with no bias to prioritize tasks. A snapshot of a usage of this graph can be seen below:

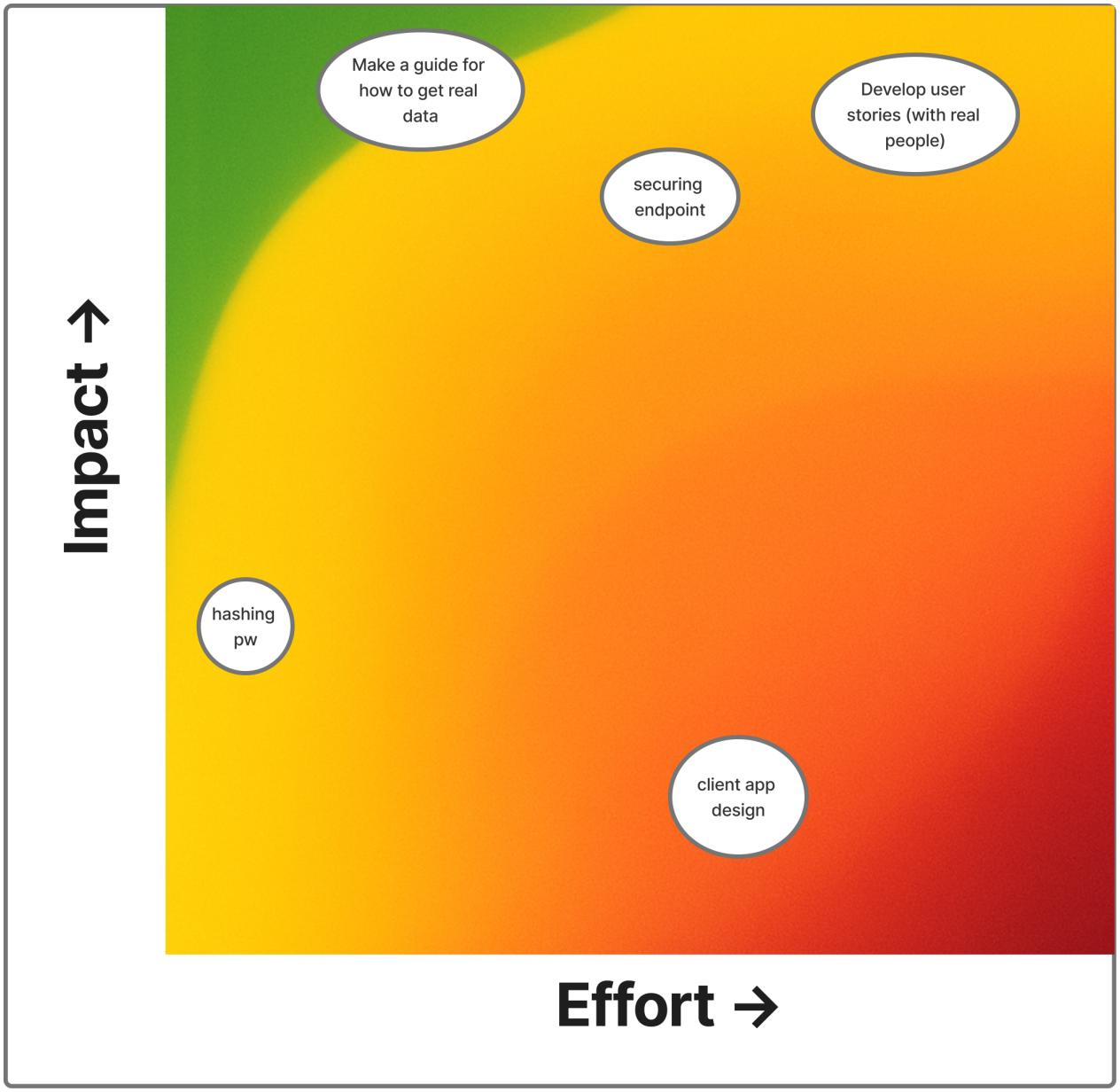

Figure 4: Impact Effort Snapshot

Creating concepts from sources like user stories, non-functional requirements and others also proved to be a powerful idea that arised from the project initiation period. This method allowed great flexibility while keeping the opinionated choices of prioritization - often arising from the data-driven nature of the project.

5.2 Tools

A powerful tool for this project compared to other semester projects proved to be Visual Studio Code by Microsoft. Through the power of extensions, actions, and powershell scripts, the whole team was able to work

collaboratively and effectively.

For documentation, a hybrid approach with markdown (Pandoc) and PlantUML was chosen. Markdown allowed for easy collaboration (both Live Share and Git) with not many conflicts while providing a powerful framework in which styling is not a concern during content creating. PlantUML allowed for quick diagramming and sketching of ideas without the need to open a separate application. This allowed for quick iterations and changes to diagrams as the project evolved.

PlantUML was also important for all UML purposes in general - text-based form of diagrams allowed for great collaborative experiences, simple exports and truly dynamic changes. The ease of use and its popularity also resulted in UML being used outside of pure engineering diagramming, for example as shown on the figure below depicting a quick guide we shared on our board in Figma:

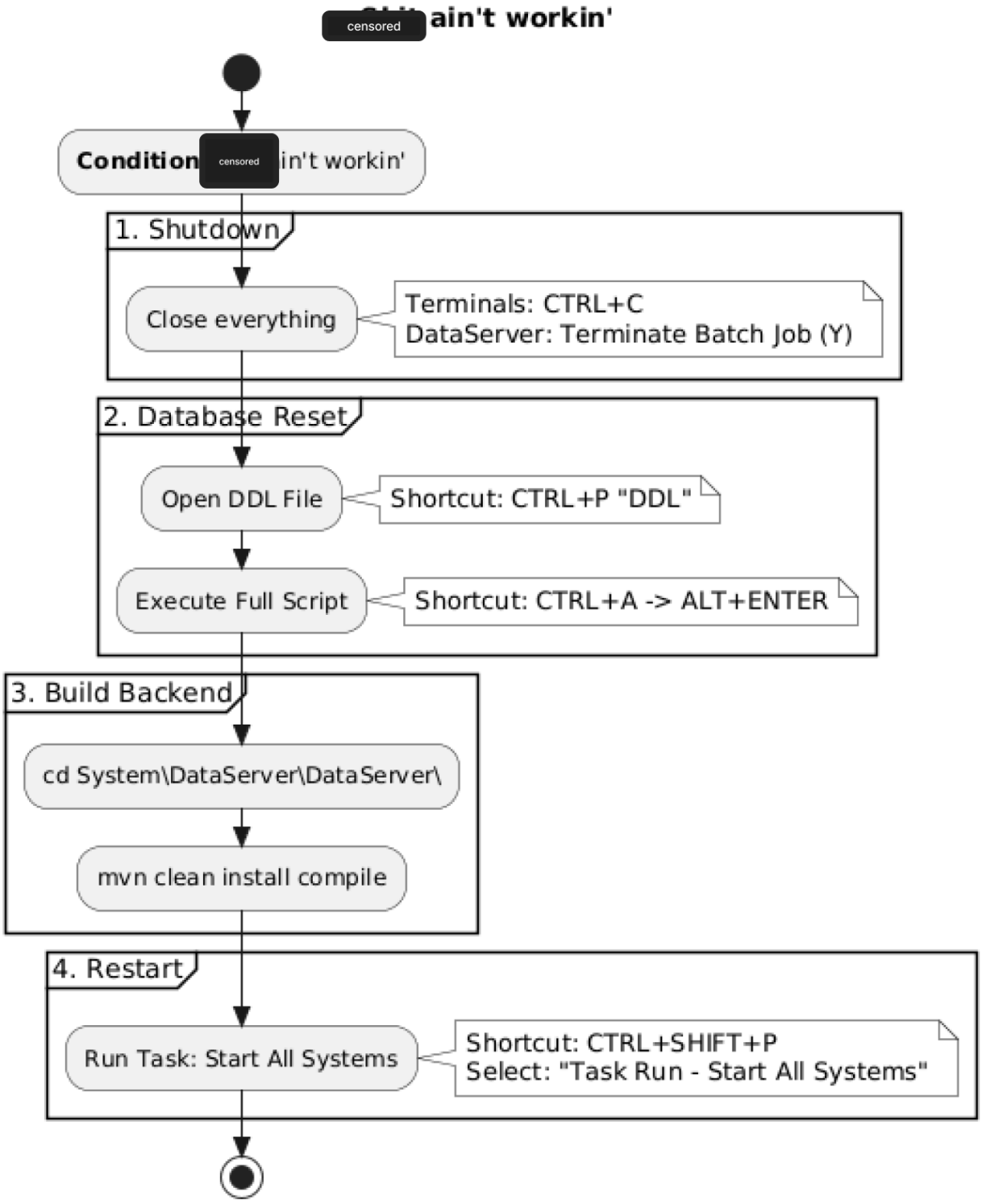

Figure 5: UML Guide

Visual Studio Code also allowed for efficient work with the database - providing the ability to connect to the database and easily execute all queries from the same environment as everything else.

On top of hosting all these features, Visual Studio Code also allowed for automating terminal sessions for running all parts of the system (building documentation, running all systems, zipping everything) and because of the language server support, also provided one IDE for managing the whole project.

Despite that, a significant part of the collaboration still relied on Figma - including a part of documentation (diagrams, user story maps, ...) and general task/iteration management. Figma proved to be a superior tool for brainstorming, quick collaboration sessions, and truly free and easy alignment sessions.

---

6 Personal Reflections

6.1 Guillermo Sánchez Martínez

During this semester we wanted to try a different agile framework since we do not know which one our future company will use ,and we would like to find the most relevant for us in order to work more efficiently. This time we decided to combine Kanban with the role of Product Owner from SCRUM. This allowed us to switch more easily from one task to another without needing to wait for the Scrum Master to assign the next task. Additionally, keeping the role of Product Owner, helped us maintaining an unified and static vision of our final system.

At the beginning of the project I thought this agile method was the best for me, but now that we have finished I would not say so. Because of not having any tasks assigned, I felt less attached to the project, and in some cases that meant taking longer to finish a tasks because it did not feel like mine. This effect of personal disattachment from a task has been studied and it has been proved that clear task ownership is directly linked to increased motivation and project engagement (Masood et al., 2021). I am still getting to know myself and this project has definitely helped with that. If I could reshape our agile framework I would have included a rule saying that a task must be taken by a person, so I could feel more attached to it and therefore work more effectively. To continue discovering myself and how do I perform in different methodologies, I am open to explore other agile frameworks in the next semester in order to find the one that suits best with me. Since in a company I will have to adapt to their prefered working methodology.

In this project there has been differences between members that worked actively and members who worked passively. Meaning that some of us were taking tasks one after another, sacrificing their time for the good of the project, while others spent less time and priorized personal time. I personally think that this group is great, but after collaborating for two projects in a row we might need a change. We already know how each of us reacts to different challenges and because of not having task ownership, we knew that if we didn't finish a task someone could take over our work and continue from there. Maybe expanding the group to 12 members next semester is the change we need.

Regarding next semester project, I am excited to work with such a big team and take the project as a job and not as an assignment. Since it might be the last big project I am going to do before landing an intership, and therefore it will be the last opportunity to learn a lot before getting into the real job market.

Concluding, the Kanban framework has been something useful to work with but I would make some modifications to it in order to perform better. I am happy with the final system we built, and I feel like this has been the first real project I have ever done, which is something i am proud of.

6.2 Piotr Junosz

This semester, our group decided to explore new methods and frameworks so that we can learn more about different types of software development processes. The idea was to maintain working in the iterative, agile approach, transiting from strictly using Scrum with Unified Process (UP) to Kanban workflow while retaining Product Owner role which was found useful in the previous semester. For me, the most significant aspect resulted from changing the way of working, more precisely planning and splitting the tasks within the group. We moved from a push-based planning model, in which the work is assigned upfront in the beginning of each sprint, to a pull-based system using our Kanban board. The efficiency of completing and splitting the tasks

improved greatly due to the fact that after each group member finished their work, they could immediately pull next tasks from "To Do" column. Those experiences can be explain by comparing push and pull theories, which confirms the success of our Kanban approach. A Push System relies on predetermined scheduling, where "work items are scheduled based on deterministic planning" (Kanban University, 2022). In comparison, Kanban uses a Pull system, where the process is controlled by available capacity. The Kanban Guide explicitly states that in a pull system, "completed work is regarded as more valuable than starting new work" (Kanban University, 2022). Our new team rule - that everyone has to commit all changes before taking a break - was designed to support the flow and improve efficiency of completing the tasks even more. Because of this policy, unnecessary waiting for a group member to finish they assigned tasks was avoided and everyone could just take over the unfinished work. Having experienced the fluency of the pull system, I now realise that self-organized pulling work based on capacity is far more efficient than the push system in the Scrum + UP setup. This successful transition showed us that there are a lot of different frameworks and methodologies to try out in the future development work. Understanding each new process method while comparing to the ones already learned is essential when it comes to choosing your own best and most efficient approach. As a next step, I would like to know more about defining clear process rules as well as learning any new agile frameworks that can improve group work efficiency even further.

6.3 Alexandru Savin

My thinking about feasible SEP3 ideas and their implementation started long before the actual project. I and my group members shared a common idea that we should develop a product in sync with the current SDGs, and although it had to be scoped and narrowed down, it is good to know it's core idea remains to promote free education possibilities and share human knowledge. During the project I brightened my understanding and got new competencies in Java, C#, and architectural patterns.

I'm think overall it was a correct and fair decision, that we chose kanban board as our main framework. Our app is of small scale and it was comfortable to keep track of where the current development progress was. We transferred the role of Project Owner of Scrum into our project and it played out well for the project, however for group members to feel more responsibility I can imagine we should have implemented the term based switch of the POs as we did in our previous semester.

Unfortunately, I did not make timesteps of our kanban board in order to analyze a cumulative flow infographic. Based on additions over time from our github inside statistics, we see, we've experienced some blockages and our workflow went not as steady as expected, nevertheless the easy pull of tasks mechanism allowed group members to overcome those obstacles. Each of our group members shared ownership of the project and could contribute anytime to adding or completing the tasks from the board.

Github version control and working on different branches felt much more useful compared to last semester. We developed a rule and sticked to it most of the time, claiming that each branch once ready, had to be peer tested locally, reviewed and only then merged into main, which also makes common sense.

I highly value the real feedback I received, because it did in fact once happen that, while one of my implementation methods pull request was being analyzed by the reviewer from security risks perspective, it proved to be a threat, which was shortly mitigated shortly after and before completing the feature and deleting the branch.

The time spend on this project gives me another chance to dive in and retrospect my workflow over the past weeks, my knowledge, my abilities to contribute to a project as an independent individual, evaluating and completing tasks on one side of the hand and as a student, a team member, on the other hand, where to be a good student is to be predictable, someone who can either be keep up with the tasks he choose to solve and be transparent about the difficulties he encounters and just be a group member other peers could rely on. All in all, I'm looking forward to the next semester project and eager to learn and practice methods to help me work and acquire knowledge more efficient.

6.4 Halil Ibrahim Aygun

During this project, I was involved in almost every stage of the system, working across different layers and at different points in time. I often handled features that needed to work end-to-end, meaning I had to understand and connect the database layer, backend logic, and client-side behavior. This gave me a much broader view of the system than focusing on a single isolated component, but it also came with challenges that required constant coordination with the rest of the team.

Our workflow was based on shared responsibility rather than strict code ownership. Multiple people could work on the same branch, and progress was discussed daily during meetings where we talked about what we had done, what we planned to do next, and how we could improve our approach. Having a shared Kanban board helped keep tasks visible and flexible, but it also meant that changes made by one person could quickly affect others. Because of this, I had to deal with merge conflicts several times, especially when my branch was waiting for review while other features were merged into main.

Handling these conflicts taught me how impactful even small changes can be in a shared codebase. I became much more aware that my decisions did not only affect my own work, but could slow down or complicate the work of others. This experience highlighted the importance of communication and timing, especially in a project where many features evolve in parallel.

At the beginning, working with a distributed system that combined multiple technologies and programming languages felt chaotic. However, by applying the principles and structure we learned during the semester, I gradually understood how the different parts fit together. This helped me move from seeing the system as many separate components to understanding it as one connected whole.

Through this project, I feel more confident working in larger teams and navigating complex systems with multiple branches and contributors. I learned how to collaborate in an environment where changes happen continuously and where coordination is just as important as writing code. If I were to work on a similar project again, I would focus even more on early communication about changes, discussing potential impacts beforehand, and supporting teammates more actively. Helping others and receiving help often led to learning new things myself, making collaboration a clear win-win.

6.5 Eduard Fekete

This semester brought the best and the worst of experiences. To start on the positive side, I believe that the way of our working was an extremely powerful blend of industry standards that was created after our team debates. I dreamt of something like this for a long time, every time listening to Uncle Bob, Kent Beck, Martin Fowler and other great guys, I just thought about how exactly it could work for me, how could I arrive at some satisfactory way of functioning that would be easy to get people into, that would not make VIA complain, and that would actually make sense.

And here it was, the nice chaos of XP with the structure of Kanban and benefits of Scrum. Although, not to idealize - many people absolutely avoided pair programming. Once when I suggested “let’s do it together, can you share screen?”, the answer was “Nah, it’s super easy, I’ll just do it and you check it afterwards”. And yes, that defeats the purpose, this wasn’t 100%. And it wasn’t super easy.

It also really got me when someone would “review their own changes”, and in an attempt to redeem “refactored the code” and funnily enough, all this without even running or contributing to the tests.

However, claiming that the process failed because of a lack of adherence and expertise puts us back to the waterfall times, where an argument of type “You should have thought about it in the Analysis phase” would be the way to justify why waterfall works rather than addressing its flaws.

And so the question arises - *what was it that failed and what was it that worked*?

In order to look why things worked, let’s look at what worked. We were able to efficiently and quickly deliver features, with not much bureaucracy and sweet sweet agility. Disputes of what and how to do were replaced with getting data, shared code ownership, and a bunch of refactoring.

What didn't work? I believe that our bus factor was at most 2, if not 1. We absolutely lacked a shared understanding of crucial aspects despite tons of accessible documentation. And there it was - "tons" of documentation maybe failed us.

The group contract is another chapter of the story itself. Karaoke never happened and was not even a proper punishment to begin with. I never liked the idea of involving supervisors or any such external party in a position of authority in the SEP conflicts because it just felt like cheating to get conflicts resolved by someone else. Now I would not go into a project without signing something saying that if someone decides to not show up absolutely for anything, and if they lie about their work, they will have real consequences besides "singing karaoke". I cannot believe my *naïvety* of thinking that easing the rules from last semester would just work like magic, I think that freedom is something more complicated than just removing rules.

And this connects to the bus factor again - this system worked better than the previous Scrum + UP one, but was it because we were more productive or because suddenly without any fairness/equality concerns of sprint planning, there was full power for a couple of people to do the work while the rest could ease? Did the lack of code ownership mean a lack of responsibility over the system? Is it in any way fair if someone has more responsibility over the system *purely* because they put more effort? And actually, did I bite myself on this because I predicted a lot of this in my head and thought that having no constraint of taking on the work would somehow only improve my efficiency without having a negative impact on the rest of the people?

To bridge the gap between speculative illusions of reality and the more refined reflection of it, let's form some facts:

- the group contract was broken often, consequences were not enforced
- the methods were more of an inspiration rather than a strict framework
- promises were broken, assigned work was not done
- transparency and honesty were often lacking (e.g. AI usage)

And with a weak proof or rather subjective anecdotal evidence, if these facts hold true, then I claim that none of the previous/other processes would have worked. I believe that a team that does not have a good sense of responsibility, honesty, and transparency can not achieve good results with any normal process; with normal including nothing that would break the law, ethics, ideally common sense as well.

Nevertheless, I would not be able to claim that I am data-driven without proper objective measurements. Therefore, I propose that if the process of judgment is fair, then the grade should reflect the quality of my work on semester project, not my subjective talks in this report. And without such judgment, I could merely conclude and reflect that my contribution was not perfect but never expected to be perfect, though of high quality, consistent, and conducted in a professional and ethical manner.

And to reflect further, I would like to say that I haven't felt such pride in a long time as when seeing some of the team members just crushing the whole SEP aspect of the project. Conducting an hour-long interview on a call because we needed data, studying how kanban works, speaking up on big issues, and even going to "prison" just to lock on a task because that's what we agreed to do are in my opinion the moments that will shape this team into something more than regular Software Engineers. I hope that people who find themselves in this understand the transformation that happened during the semester in the same way I see it.

---

7 Reflect on Supervision

In the project team, we were often aligned on ideas and approaches, which made critical analysis and reflections on our work more challenging. The role of supervision thus became a crucial element of our project process. The supervisors often challenged our assumptions and decisions, ensuring that the project was kept improving based on proper processes and data, rather than just group consensus and momentum.

Each of our supervisor meetings had a clear agenda - acquiring external feedback and clarifying doubts. We did not focus on mere presenting of our progress but rather on seeking constructive criticism and guidance. This

approach allowed us to identify potential pitfalls and areas for improvement that we might have overlooked without the supervisors.

Another aspect of the supervision was filling the technical gaps in our knowledge and project requirements. The supervisors always helped with long-term planning which could not have been done by the team alone due to the limited knowledge towards the beginning of the project.

The frequency of meetings was resulted mainly from the mandatory supervisor meetings and our own needs. Therefore, every time we were not sure about our decisions related to the project we communicated with one of the supervisor about the need to have a meeting with them. We communicated mainly through ItsLearning and e-mail.

---

8 Conclusion

The project showed how flexible structured team collaboration works through our transition from Scrum to Kanban pull-system workflow. The experience we gained helped us develop a list of best practices and common mistakes which will guide our future group projects.

The team needs open communication and respect as fundamental values together with a Product Owner who manages the vision to achieve successful teamwork. We should keep using standup meetings to exchange information while supervisors need to provide essential feedback and help with technical issues and code submissions must happen regularly to avoid creating bottlenecks.

It would be a good idea to avoid three main mistakes which include making decisions based on assumptions when data exists, enforcing processes that block work and splitting into groups that do not collaborate well with each other. The team should focus on achieving fundamental goals before attempting to tackle complex problems.

The team needs to keep their group contract active while using Kanban pull-based workflow management and feature decisions must be based on data. The team should seek outside feedback to confirm their project's direction.

---

9 References

Kanban University. (2022). The official guide to the Kanban method (Version 2). Mauvius Group Inc. https://kanban.university/wp-content/uploads/2023/04/The-Official-Kanban-Guide_A4.pdf

Pierce, J. L., Kostova, T., & Dirks, K. T. (2001). Toward a Theory of Psychological Ownership in Organizations. The Academy of Management Review, 26(2), 298–310. https://doi.org/10.2307/259124

10 Appendices

10.1 Appendix 1.1 GroupContract

10.1.1 Group Contract

Can be found as GroupContract.pdf

10.2 Appendix 7.3: Other documents

10.2.1 Project Guidelines

Can be found as ProjectGuidelines.pdf

10.3 Appendix 6.1 Personal Profiles

10.3.1 Personal Profiles

Can be found as PersonalProfiles.png

10.4 Appendix 12.1 Process Brainstorm

10.4.1 Process Brainstorm

Can be found as ProcessBrainstorm.png