INSTITUTO POLITÉCNICO DE LISBOA

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura em Engenharia de Informática e de Computadores

Assembly Odin
School Management System for Point-Based Learning

Bernardo Serra
a47539@alunos.isel.pt

Rafael Costa
a48315@alunos.isel.pt

Supervisor: João Trindade

Project and Seminar

July 2024

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Assembly Odin

47539 Bernardo Filipe Marcelo Serra

A47539@alunos.isel.pt

48315 Rafael Alves da Costa

A48315@alunos.isel.pt

Orientadores: João Trindade

joao.trindade@isel.pt

Report for Projecto e Seminário Class of Licenciatura em Engenharia Informática e de Computadores

July 2024

Abstract

Education is rapidly evolving to meet the demands of a technology-driven world. Traditional methods often fail to keep pace with the dynamic nature of modern learning environments. Innovative approaches to teaching and schooling are essential, necessitating custom software solutions tailored to specific educational needs. Such unique software must accommodate diverse and evolving pedagogical strategies, ensuring that educational institutions can provide personalized and effective learning experiences.

The Assembly School of Technologies in Lisbon is recognized for its innovative approach to technology education, emphasizing hands-on learning and skill development in areas such as coding, robotics, and design. Despite its progressive methodology, the school's manual system for managing student activities and gaming time allocation has become increasingly inefficient and error-prone. This project aims to address these issues by developing Assembly Odin, a costume made school management system designed to streamline and enhance the educational process.

Assembly Odin integrates multiple modern technologies, including ASP.NET Core, Kotlin, Spring Data, PostgreSQL, MongoDB, React, React Router, CSharp, Windows Services, PowerShell, and Docker. These technologies work in concert to automate attendance tracking, gaming time allocation, and gaming time management. A key component of the system is Heimdall, a custom-made point tracking system specifically developed to monitor and log computer usage by students, facilitating the automatic deduction of gaming time based on their activities.

By automating these tasks, Assembly Odin not only enhances operational efficiency but also reduces human error, providing a scalable solution that grows with the institution. The implementation includes a user-friendly web application for teachers to electronically mark attendance, a robust backend server for comprehensive point management, and a sophisticated monitoring system for tracking gaming time.

The result is a powerful, user-centric platform that significantly improves the management of student activities, ensuring accurate and real-time updates. This innovative solution not only streamlines administrative tasks but also fosters a more engaging and transparent learning environment, ultimately enhancing the overall educational experience at the Assembly School. The project hopes to demonstrate the technical proficiency, innovative thinking, and practical skills of its developers, showcasing their capability to solve real-world problems in imperfect situations with unique characteristics.

Resumo

A educação estava cada vez mais a evoluir rapidamente para responder às exigências de um mundo guiado pela tecnologia. Os métodos mais tradicionais de ensino, na maioria das vezes, não conseguem acompanhar a natureza dinâmica dos ambientes de aprendizagem modernos.

Por conseguinte é necessário que haja uma abordagem inovadora no ensino uma vez que é essencial para o desenvolvimento das futuras gerações, necessitando de soluções de software personalizadas adaptadas às suas necessidades específicas. Os Softwares desenvolvidos com este propósito devem acomodar diversas estratégias pedagógicas, garantindo que as instituições de ensino possam proporcionar experiências de aprendizagem personalizadas e eficazes.

A Assembly Escola de Tecnologias em Lisboa é reconhecida pela sua abordagem inovadora à educação tecnológica, enfatizando a aprendizagem prática e o desenvolvimento de competências em áreas como a programação, a robótica e o design.

Apesar de ter uma metodologia inovadora, o sistema manual da Assembly Escola de Tecnologias para gerir atividades estudantis e a alocação do tempo de jogo tornou-se cada vez mais ineficiente e propenso a erros.

Com o intuito de resolver esses problemas propomos o projeto Assembly Odin, um sistema de gestão escolar personalizado criado para otimizar e melhorar o processo educacional.

O Assembly Odin integra várias tecnologias modernas, incluindo ASP.NET Core, Kotlin, Spring Data, PostgreSQL, MongoDB, React, React Router, React Bootstrap, CSharp, Windows Services, PowerShell e Docker. Estas tecnologias funcionam em conjunto para automatizar o rastreamento de presenças, a alocação do tempo de jogo e o gasto do mesmo.

Um componente chave do sistema é o Heimdall, um sistema de rastreamento de pontos personalizado desenvolvido especificamente para monitorizar e registar o uso de computadores pelos alunos, facilitando a dedução automática do tempo de jogo com base nas suas atividades.

Ao automatizar essas tarefas o Assembly Odin, não só melhora a eficiência operacional, mas também reduz os erros humanos, proporcionando uma solução que cresce com a instituição. A implementação inclui uma aplicação web de fácil utilização para os professores marcarem a presença de forma electrónica, um servidor backend robusto para a gestão abrangente de pontos e um sistema de monitorização sofisticado para o rastreamento do tempo de jogo.

Como resultado temos uma plataforma poderosa e centrada no utilizador que melhora significativamente a gestão das atividades estudantis, garantindo atualizações precisas e em tempo real. Esta solução inovadora não só otimiza as tarefas administrativas, mas também promove um ambiente de aprendizagem mais envolvente e transparente, melhorando a experiência estudantil na Assembly Escola de Tecnologias.

O projeto procura também demonstrar proficiência técnica, pensamento inovador e as habilidades de aplicação prática dos conhecimentos dos desenvolvedores, evidenciando a sua capacidade de resolver problemas do mundo real em situações imperfeitas e com características.

Contents

CHAPTER 1

Introduction

Contents

The Assembly School in Lisbon stands out as a pioneering institution in technology education, striving to equip future professionals in Portugal with the skills necessary for a technology-driven world. Traditional educational systems often struggle to keep pace with the rapidly evolving technological landscape, but Assembly addresses this challenge by offering a unique learning methodology. This methodology enables students to tailor their education according to their interests and strengths, with a strong focus on essential tech skills such as coding, robotics, and design while rewarding them for the work and effort they put in.

A commitment to hands-on learning across various the fields of technology is at the heart of Assembly's educational methodology. Instead of conventional tests and grades, Assembly encourages practical project development and recreational activities with technology and learning. Students earn points (XP) for their effort and engagement in theoretical (TECH) and practical (VOC) units. These points can be redeemed for hours of entertainment creating a work reward system.

However, the current system for managing student activities presents several challenges that are increasingly becoming untenable for a growing institution like Assembly. Teachers have to manually mark students' attendance and activities on Excel sheets, which are then allocate points to students. Students subsequently record their gaming times on sheets of paper when requesting gaming equipment, and administrators must then transfer these records to a different Excel sheet. This manual process is not only inefficient and prone to errors, but it is also difficult to manage, lacks scalability and doesn't fit the idea of what Assembly is meant to be. As Assembly continues to expand, the limitations of this outdated system become more pronounced, underscoring the urgent need for a scalable, automated solution that can efficiently handle the institution's growing needs.

To address these issues, we propose the implementation of an automated management system named Assembly Odin. Assembly Odin is a comprehensive, multi-piece solution designed to restructure the way points and activities are managed at Assembly. It integrates several features to automate the entire process, from attendance tracking to point allocation and gaming time management. Teachers will mark student attendance electronically for theoretical classes and approve practical works, with automatic point tracking. When students use their points on gaming computers, tracking software will automatically deduct the necessary points, eliminating human error and streamlining the process. This integrated approach will be able to fix all of the issues on the current system, while also having several challenges caused by the unique aspects of the company and their educational framework.

1.1 Structure of the Document

This document has the information for both understanding the initial issue and providing a comprehensive understanding of the automated point management system for Assembly School that we have developed. We will know talk about all the chapters and what information they contain

Chapter 2: Problem Description
This chapter presents a detailed description of the problem domain, discussing the challenges faced by the current manual system and the requirements for an effective automated solution. It outlines the inefficiencies and errors in the existing system and the main pain points that require fixing.

Chapter 3: Architecture
This chapter provides an in-depth overview of the system architecture, emphasizing its key components and their interactions. It explains how the architecture addresses the objectives outlined in Chapters 1 and 2, and what were the solutions we elected to use to tackle them. The technologies in use are detailed in that section as well.

Chapter 4: Implementation
This chapter delves into the implementation details of each component of the automated point management system. It discusses their functionalities, interactions, and deployment processes. Additionally, it explores the extensibility of the platform, demonstrating how it can be customized and expanded to meet specific needs. It also contains a record of the possible points of failure of this system and what challenges will need to be eventually tackled.

Chapter 5: Tests
This chapter focuses on the testing and validation processes conducted on the system to ensure its reliability and performance. It includes details on test cases, methodologies, and results, demonstrating the system's effectiveness in meeting its objectives. Its main points are talking about stress tests and beta testing with its intended and future users.

Chapter 6: Final Remarks
This chapter presents final remarks on the project, reflecting on the accomplishments and identifying potential areas for further improvement that are meant to be tackled in a near future. It discusses the envisioned impact of the automated point management system on Assembly School and outlines future developments that could enhance the platform's

capabilities.

CHAPTER 2

Problem Description

Contents

The Assembly School in Lisbon is at the forefront of technology education, preparing students for the demands of a technology-driven world. However, the current manual system for managing student activities is becoming increasingly obsolete and unmanageable as the institution grows. This chapter provides an in-depth analysis of the challenges posed by the educational framework at Assembly, its complexities, and its features to improve the reader's understanding. It is crucial to consider this chapter, as many decisions are influenced by the underlying rules.

2.1 Underlying Educational Framework and Rules

The Assembly School operates within a unique educational framework that emphasizes both theoretical (TECH) and practical (VOC) units. This framework is guided by several underlying rules and principles that shape the educational experience and influence the management of student activities. Understanding these rules is crucial for appreciating the need for a tailored management system like Assembly Odin, both on understanding and creating it. The first step of development was gathering and compiling all of those underlying rules, and we have compiled that information in this section

2.1.1 Departments, Fields of Study, and Modules

The educational structure at Assembly School is organized into three primary departments: Design, Robotics, and Code. Each department offers various fields of study, also refereed to as paths, which cater to different interests and career goals. Within these paths, students engage with specific modules. Its also important to note that there is no formal section defined anywhere. Usually sections are caused by students that take classes at the same time each week.

Departments

Departments never change are the baseline of all education done at Assembly. Students are free to swap between existing departments or even do modules from several at the same time. It is also encouraged to hop between paths for certain fields of study, per example, game development, where both design and code are very important, so there is no logical relationship between students and departments besides what module they are currently engaged in.

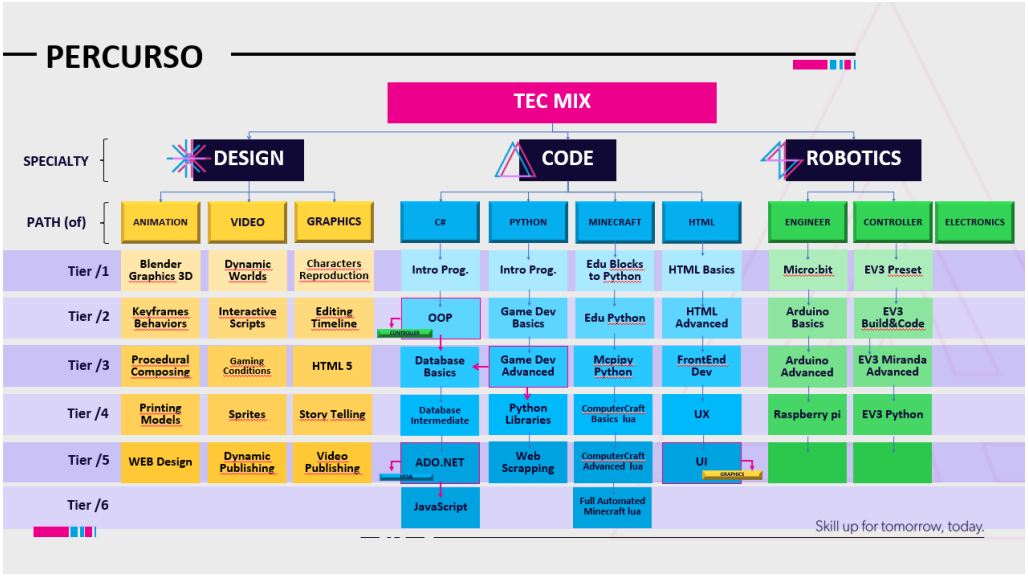

Figure 2.1: An image representing the current logic for Departments, Fields of Study, and Modules

Fields of Study (Paths)

Each department offers several fields of study, known as paths, which guide students through a structured learning journey in their chosen area. This fields of study tend to be removed updated or added at the start of every year, but can remain the same. These paths focus on developing a certain skill-set for a student, be it things like game development or animation design. To do that its expected of the students to sometimes take modules of other fields of study and departments to fully complement their current skill-set. Those cases are handled on a case by case, as there is a big emphasis on having the students go on the path they desire to learn more about, but usually involve having the whole section that is taking that field of study have classes of other modules.

Modules

Modules are a set of classes designed around 24 hours of learning (12 TECH and 12 VOC) and giving the students a certain skill at a level, like angular intermediate, or python essentials. They have a curriculum that is expected to be accomplished during their length. They are swapped often, as the curriculum for a certain field of study evolves or changes. There can be several sections taking a certain module in the same week, but never at the same time.

2.1.2 Theoretical and Practical Units

The educational framework at Assembly School divides the curriculum into theoretical (TECH) classes and practical (VOC) work. Theoretical units focus on traditional academic knowledge, while practical work emphasizes hands-on skills and real-world applications. Practical work also serves to support and consolidate the knowledge gained in theoretical classes, helping students to apply and reinforce what they have learned.

TECH Classes

TECH classes are the theoretical units that focus on imparting traditional academic knowledge. These classes work like regular classes in every sense. Their contents are determined by what module they belong to and go according to their class plan without much change. They happen once a week, for a total of 12 weeks, for an hour each time in a fixed schedule.

- **Module Participation**: Students can participate in various modules at the same time, although its rare, but as such they can have several TECH classes in a week.
- **Attendance**: Attendance is mandatory for this classes, and they can only gain points if they attend. Additionally there can be placed upon them a gaming freeze if they don't attend more than two classes in a row.
- **Point System**: Points (XP) are awarded based on attendance for TECH classes, at a rate of a point for each hour. There extra points can be awarded for TECH class challenges or good behaviour, although that's rare.
- **Evaluation**: Students are evaluated continuously through assignments projects and participation. There is no grading system, just a holistic evaluation preformed at the end of each module.

VOC Classes

VOC classes consist of a hour per week the students must spend at Assembly doing practical work assigned by their teacher with the intent of consolidating they theoretical knowledge gained in the TECH class of that week. They can do it at any time, as long as they are on premises. There always is a teacher assisting people with practical work, and he is expected to attest that a certain student did their assigned VOC class. The contents of said class can changed based on what the teachers deems necessary, so there is no standard guideline for what VOC they should be preforming.

- **Module Participation**: Students can participate in various modules at the same time, although its rare, but as such they can have several VOC classes in a week.
- **Attendance**: Attendance is mandatory for this classes, and they can only gain points if they attend. Additionally there can be placed upon them a gaming freeze if they don't attend more than two classes in a row. If a student for whatever reason misses this class in one week, he is expected to do the one he missed and the following one in the upcoming week.
- **Point System**: Points for VOC classes are assigned at a rate of a point per hour, but if a VOC is done, the minimum amount of points a student can get is one. The point is only assigned after the support teacher confirms the work was done
- **Evaluation**: There is no formal evaluation for VOC work, but there can be a review done by a teacher for student feedback

2.1.3 Point-Based Reward System

Assembly School employs a unique point-based reward system to incentivize student participation and engagement. Points are earned through various activities and can be redeemed for privileges such as gaming time or merch.

- **Point Allocation**: Points are allocated based on attendance too TECH and VOC classes and rewards for class challenges. Students can also preform school work, costum projects or help around for extra points. The usual rate of point gain is 1 per hour, bust helping around, or class rewards usually have a fixed payout.
- **Redemption of Points**: Points can be redeemed for accessing gaming equipment and merchandise. For gaming time the rate of point expenditure is 1 point per 30 minutes with a minimum of 1 point deducted for each instance of gaming.
- **Bonuses and Penalties**: Students can earn bonus points for exceptional performance and participation. Conversely, points can be deducted for violations of school rules or failure to meet participation requirements. There can also be placed a point freeze on a student if they miss too many classes in a row, either TECH or VOC.

2.1.4 Roles at Assembly School

The community at Assembly School comprises students, teachers, and administrators. Each group has distinct roles and responsibilities. It's important to note that administra-

tors can also be teachers, and both teachers and administrators do not earn or use points within the school's system.

Students

- **Participation**: They have a weekly schedule for TECH classes and need to complete one VOC class per week at least.
- **Point System**: Students earn points (XP) based on the completion of activities. Their point count is never reset after they first enroll and their balance can also go into the negatives, which will earn them a warning or punishment, depending on if it went into the negatives due to abuse of the system. Their point expenditure can also be frozen as a punishment
- **Responsibilities**: Students are responsible for telling admins they performed VOC classes, and attend TECH classes.

Teachers

- **Responsibilities**: Teachers prepare TECH and VOC classes and lecture TECH classes. They also need to track student attendance to TECH classes and can approve students VOC work. They also can create new modules and develop new fields of study. They can also lecture several modules, and be in different departments.
- **No Point Usage**: Teachers do not earn or use points within the school's point system but can give extra point to students as rewards. They are also able to request for point system punishments for students, but they need to be approved by an admin.

Administrators (Admins)

Administrators at Assembly School are also teachers who take on additional responsibilities related to the management and oversight of the educational framework. They tend to have access to the entire system and manage it. Besides doing everything a teacher does they have several more responsibilities.

- **Oversight**: Administrators oversee the implementation of the educational framework and ensure compliance with the school's rules and guidelines. Every decision that involves changes to the overarching system needs to pass through their approval.

- **Policy Enforcement**: Administrators enforce school policies related to attendance, participation, and point redemption. They monitor compliance and address any violations.

- **Data Management**: Like teachers, administrators ensure that accurate and up-to-date records of attendance, point allocations, and student performance are maintained. They are responsible for the integrity of these records.

- **No Point Usage**: Administrators do not earn or use points within the school's point system.

2.2 Current System Overview

The existing system relies heavily on manual processes. Teachers manually mark students' attendance and activities on Excel sheets. These sheets are used to allocate points (XP) for student participation in theoretical (TECH) and practical (VOC) units. Students record their gaming times on paper when requesting gaming equipment, and administrators transfer these records to a different Excel sheet. As Assembly continues to expand, the limitations of this outdated system become more pronounced.

- **Inefficiency**: Manual data entry and record-keeping are time-consuming and labor-intensive, requiring substantial administrative effort.

- **Lack of Scalability**: As Assembly School expands, the volume of data to be managed increases. The manual system cannot efficiently scale to handle larger datasets.

- **Difficulty in Management**: Coordinating and verifying multiple Excel sheets and paper records is complex and error-prone, making it difficult to maintain accurate and up-to-date records.

2.2.1 Requirements for an Automated Solution

Given the limitations and challenges of the current manual system, there is a clear need for a scalable, automated solution. This solution must:

- **Automate Data Entry**: Reduce administrative workload and minimize errors by automating attendance and point tracking.

- **Centralize Data Management**: Consolidate data into a centralized database, simplifying access, verification, reporting and allowing to add more systems like the point management one in the future.

- **Ensure Scalability**: Design the system to handle increased data volumes and user interactions as the institution grows, transforming a pain point into a positive solution that helps with scalability and management.

2.3 Review of Alternative Solutions

In the process of designing Assembly Odin, we considered several existing school management solutions that could potentially be used to manage student activities and points. However, these tools were not implemented due to the unique nature of Assembly's learning system, which requires a tailored solution.

2.3.1 Existing School Management Solutions

Description: There are numerous school management solutions available, such as PowerSchool, Infinite Campus, and Schoology. These platforms offer a wide range of features, including attendance tracking, grade management, scheduling, and communication tools.

Reasons for Not Implementing: These systems are designed for traditional educational models and may not support the unique aspects of Assembly's learning methodology, such as the point-based reward system for both theoretical (TECH) and practical (VOC) units, and its automatic management and it would take both money and a decent amount of time to create a middle-ware that would need regular updating to be able to manage both those systems and the costum point tracking aspect.

2.3.2 Other Tools for Tracking Computer Attendance

Description: Various tools can be used for tracking computer usage, such as Azure Active Directory, which provides comprehensive logs of user activities on computers, and would be a good fit, as Assembly's IT system is already managed by it.

Reasons for Not Implementing: Implementing Azure Active Directory would require additional payments and subscriptions, which were not desired by the client.

2.4 The Proposed Solution

To address these challenges, we propose Assembly Odin, solution designed for school management and automated point tracking, costume made for Assembly's use case.

2.4.1 Functional Requirements

- **Class Creation and Management**: Enable the creation of new classes, assignment of students to these classes, and the representation of modules, fields of study, and departments within the system.
- **Module, Field of Study, and Department Representation**: Accurately represent the structure of modules, fields of study, and departments within the system to ensure students understand their fields of study paths and departments.
- **Automate Data Entry**: The system must automate the process of marking attendance and tracking points (XP) for both TECH and VOC classes.
- **Attendance Tracking**: Provide an interface for teachers to electronically mark attendance for both theoretical (TECH) and practical (VOC) classes.
- **Point Allocation**: Automatically allocate points based on recorded attendance and student participation.
- **Gaming Time Management**: Implement a system that tracks gaming time usage by students and automatically deducts points based on their gaming activities.
- **Centralized Data Management**: Consolidate all data into a single, centralized database to simplify access, verification, and reporting. Ensure data is easily accessible to authorized users.
- **User Roles and Access Control**: Implement secure login and role-based access control for students, teachers, and administrators.

2.4.2 Non-Functional Requirements

- **Scalability**: Design a system that must be able to scale due to an increasing number of users and data as the institution grows.
- **Reliability**: The system should be reliable and resilient to network and hardware failures.

- **Maintainability**: The system should be easy to maintain and update, with clear documentation and modular components.
- **Usability**: Provide a user-friendly interface that is easy to navigate for students, teachers, and administrators.

Taking all this into account, the next section will talk about the agreed upon architecture for Assembly Odin and its features.

CHAPTER 3

Architecture

Contents

This chapter provides an overview of the system's components and their interactions. It outlines the capabilities of the project and presents the architecture, entities, and implementation blueprint that have been designed and developed.

3.1 Overview

The proposed solution for an automated point management system at Assembly School consists of two main components: Odin and Heimdall. Odin is a server/web application that serves as the central hub for managing student activities, points, and class information working with a backend API and a frontend SAP application. Heimdall is a custom-made point-tracking system designed to monitor and log computer usage, designed to be deployed in all of the relevant computers and feed collected data to a data store. Odin and Heimdall communicate via a Non Relational Data store where Heimdall publishes its logs to and Odin consumes, process and adds the relevant information to the main database. The usage of the frontend application and the API requires authentication via an Office 365 account, using a Single Sign-On (SSO) mechanism. This requires the use of an external authentication server provided by Microsoft.

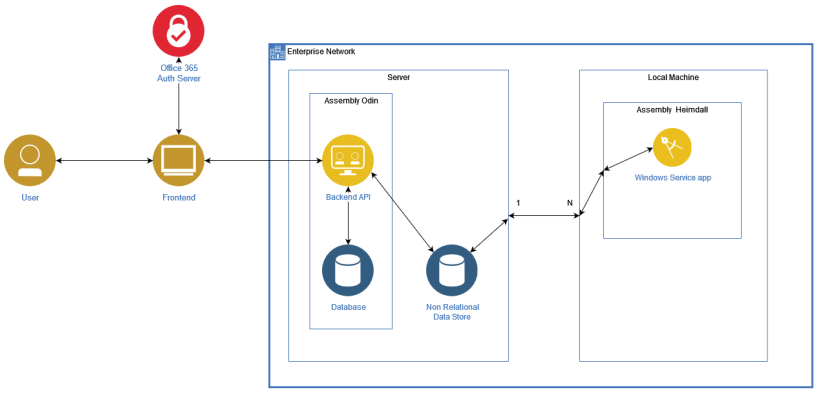

Figure 3.1: Architecture of Assembly Odin Software

Architecture Notes

We opted this two fold solution to simplify automated point deduction by creating a designated software for data collection and one for overall data processing. One of the discussed topics was creating a micro service for data processing, instead of letting Odin do it, but we deemed it over engineering, and elected to instead let the implementation of the

data processing feature be open to further expansion later down the line when it becomes relevant.

Its also relevant to mention that the implementation of a data store to store the data outputted by Heimdall applications was done with the intent of creating archaic Pub/Sub pull architecture to their communication. This once again is a simplified solution that allows the application to work normally, but can be further expanded later down the line, be it using cloud systems or a more complex implementation without requiring massive architectural changes.

Another important topic is the usage of Office 365 SSO. This was decided on the basis that every person at assembly has a office 365 account, so the usage of SSO not only makes it easier for everyone as they already have a relevant account, it also simplifies development and security.

3.2 Data Model

3.2.1 Application Storage

Data within the Odin application is meant to be stored inside a relational database system. To help explain the structure that database system should follow we elaborated a Entity-Relationship Diagram.

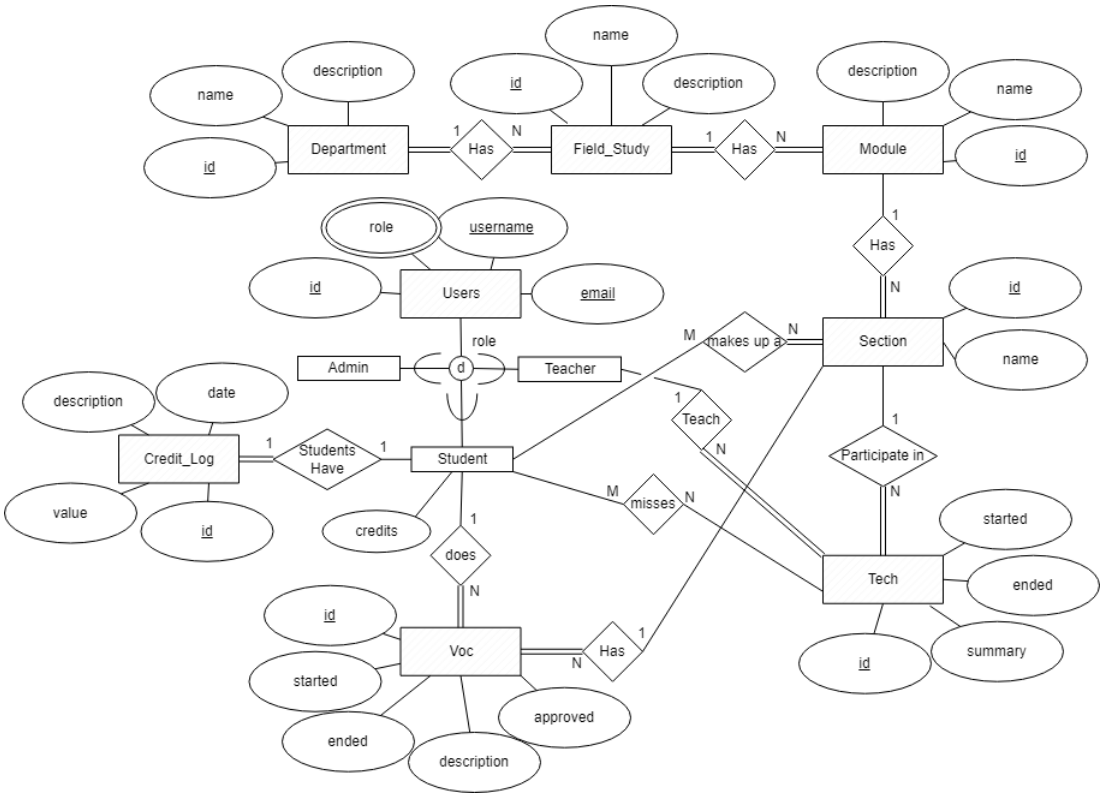

Figure 3.2: EA Data Base Model

What follows know is a comprehensive description of each of the entities, their relations and reasoning for certain design decisions

Department

A Department is a collection of fields of study in the same area of knowledge. A good example is robotics, or programming. It's identified by a unique ID and includes attributes such as name and description. Each department can have multiple fields of study, but each field of study belongs to only one department.

Field of Study

Fields of Study are study paths students can follow to acquire a certain skill set, for example, in the programming department a field of study would be frontend developer. They are associated with departments and have unique IDs, names, descriptions and their respective department. Each field of study can have multiple modules associated with it, indicating the courses or subjects available in that field.

Module

Modules are the individual courses taught within a field of study, a good example being Angular 101 for the field of study of frontend developer . Each module has a unique ID, name, description and their respective field of study. Modules are associated with sections, which represent different class groups or instances of the module being taught.

Section

A Section represents a specific group of students that's undertaking a module. As there can be several students undertaking the same or differing modules, it was decided that section would be a good name, given that it describes a section of said students that is undertaking a certain module. Each section is identified by a unique ID, name and the module that section has to do. Sections participate in tech (theoretical) classes and are made up of students. This requires the creation of a new section each time students swap modules. This is desired, with the intent of being able to save and maintain records of the students that did a certain tech class for a section with the module taken into account without falling into the mistake of when a certain student drops out of a section disturbing the records.

Users

User in the system are identified by a unique ID and have the following attributes, username, email, and role. Roles determines the user's capabilities within the application, role can be one of the following: an Admin, a Teacher, or a Student.

Student

One of the roles of User can be Student. When a user is a student he has an additional attribute for credits, and is able to participate in a section. He can also do VOC classes, and has a credit log, which track their academic credits and activities. If a users role is changed to another one, none of this data is lost, just unable to be added to.

Teacher

Teachers are another type of User in the system. They are responsible for teaching TECH classes. This role also has elevated permissions

Admin

Admins are another type of User in the system. This role has permission to do all available actions in the system

Credit Log

A Credit Log tracks the credits and activities of a students. Each credit log entry includes the attributes: ID, description, date, value and a Student. Credit logs can be added for actions such as doing a VOC or participating in a TECH class. If a role of a user changes of being a student, this logs are kept, in case it changes back, they should not be deleted, to ensure system consistency.

VOC (Practical Sessions)

Practical sessions, identified as VOC, are hands-on, practical work sessions. Each practical session has the following attributes: ID, description, start date, end date, approval status, a Student (the one who does the VOC work) and a Section (The section where the student belongs, that is relevant to the work he is doing).

Tech (Classes)

Tech represents the theoretical classes or lectures that are given by a teacher to a section of students. Each class is identified by a unique ID and includes the attributes: start date, end date, and summary. A TECH class is lectured to a Section by a Teacher, which is represented by their respective relations. By default, it is assumed that every student attends a class, but in the eventuality that they don't that is represented by the miss class relation, where students who miss a tech class are represented.

3.2.2 Assembly Heimdall Storage

Heimdall collects data by accessing security event logs, which capture login and logout events on each monitored computer. The data is formatted into JSON objects. That data is then meant to be stored in a non relational database. This is done for two main reasons:

- The first one is allowing the main application (Odin) to be able to choose to pull the resources in the exchange instead of them being forced upon him. This allows for more control of the entire system for Odin, instead of a solution where Heimdall submits the data into Odin via its API, which can have unintended side effects in the long term, per example, a situation on a larger company where many instances

of Heimdall is deployed in a very large number of computers, causing delays on the main applications other tasks, due to system overload.

- The second reason is that this allows for the data that is stored inside the logs can be changed to fit the users needs. As long as the minimum requirements for the content of the log itself is stored, more data can be added, which can be relevant. Per example one can elect to store or not the name of the machine where the log was originated, its IP address, and such information, without the need to change the relational model.

The minimum data for a event Log as needed for this architecture is defined as follows This was the minimum data that was determined to be need for the intended use of Odin to account for user computer usage time.

```
{
"name": "<Name>" | "email": "<Email>" ,
"event": "Login" | "Logout" | "Lock",
"timestamp": "XXXXX"
}
```

3.3 Heimdall Monitoring System

The Heimdall Monitoring System is designed to provide a solution for monitor, log and export User usage of Windows machines. It utilizes a combination of a executable running as a windows service and the windows event logs to function and obtain information.

3.3.1 Event Log Processing

The Heimdall system processes event logs to extract user information regarding activities. The core component, a executable, queries security event logs on Windows machines to gather events such as user logins and logouts. These events are identified by specific event IDs, which then can be converted to an event type by making a list of desired event IDs to a name inside the application, its also possible to obtain the username and the timestamp, but those have to be extracted from the XML that is part of the event log.

3.3.2 Communication with Assembly Odin

The collected data is formatted into JSON objects, which are then stored in a MongoDB database. This data collection and storage process is intended to occur periodically, that

time being defined in the Heimdall implementation. The Heimdall component doesn't need to know what the main application is doing or that even exists. Its intended for Heimdall to be able to be deployed as easily and linearly as possible, and it just has to have a place to store the data in its implementation, which in the context of this architecture is intended for a non relational database, but can even be a local file. The second rule is that it must abide by the minimum amount of information to export, as determined by its architecture.

3.4 Odin Backend Application

3.4.1 Backend Architecture

The overall architecture consists of several key components, each serving a specific role in the system. These components are designed to work together seamlessly, providing a solid foundation for the application.

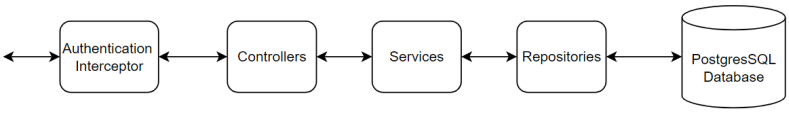

Figure 3.3: Architecture Design for Odin's backend

Repository

The repository layer is responsible for data access. It uses Spring Data JPA to interact with the relational database. This layer abstracts the database operations, allowing for cleaner and more maintainable code. By using Spring Data JPA, repetitive code is reduced through the use of predefined interfaces and query methods.

Model

The model layer defines the core entities of the application. These entities represent the various aspects of the system, such as users, classes, and attendance records. The model layer is designed to align with the relational database schema, ensuring data consistency and integrity.

Services

The service layer contains the business logic of the application. It processes data received from the repository layer and performs the necessary operations to fulfill the application's requirements. This layer ensures that the business rules are applied consistently across the application.

Controller

The controller layer handles incoming HTTP requests and maps them to the appropriate service methods. It acts as an interface between the client and the backend, ensuring that requests are processed correctly and responses are returned in the appropriate format.

3.4.2 API Design

The backend API is developed using Kotlin and Spring Boot. This framework provides a robust and efficient server-side application. The API follows RESTful design principles, ensuring a clean and scalable architecture. It includes various endpoints for managing users, classes, and points, each secured with appropriate authentication and authorization mechanisms.

For the security of our application, we decided to use Spring Security. This framework protects against attacks such as session fixation, CSRF, and click jacking, and helps prevent brute force attacks. To complement Spring Security, we implemented OAuth2 for Single Sign-On (SSO), which enhances security and convenience by allowing users to have fewer credentials across different applications. This reduces the implementation effort and increases overall security.

We also use Spring Data JPA, which helps reduce repetitive code by leveraging the interfaces and methods provided by the framework. This results in faster development and less code to maintain.

3.4.3 Communication with Heimdall

The communication with Heimdall involves Odin running a Spring scheduled task that reads the data that Heimdall stored in MongoDB, calculates the time deltas between login and logout events for each user, and updates the points for users accordingly, by creating credit logs.

By structuring the backend in this manner, the Odin application ensures a robust, scalable, and maintainable system capable of handling complex data requirements and providing a secure and efficient user experience.

3.5 Odin Frontend Application

For our frontend application, we chose to use a Single Page Application (SPA) running on a web client. We selected SPA due to its speed and responsiveness compared to other methods like multi-page applications, which require a reload each time an update is executed. Additionally, the numerous available libraries help accelerate the development process.

3.5.1 Technology Analysis

In this project, Typescript was chosen for its strong typing compared to JavaScript. Typescript supports JSX out of the box, providing full support for React web development, simplifying the need for other configurations.

React is one of the main frameworks used in the market today. It allows you to combine elements into reusable, nestable components. This makes the development process more efficient and the codebase easier to maintain.

We also used Material UI, which includes a comprehensive collection of prebuilt components ready for production use right out of the box. This accelerates the development process.

3.5.2 State Management

State management is handled using React hooks and context API. This approach ensures that the application state is maintained efficiently across different components. It also allows for easy management of global state, such as user authentication and session data.

3.5.3 API Integration

The frontend communicates with the backend through a well-defined set of APIs. This ensures that the application can fetch and display the latest data dynamically. The use of React allows for efficient data fetching and state management, providing a seamless user experience.

The combination of these technologies ensures a robust, scalable, and user-friendly application, capable of handling the diverse needs of students, teachers, and administrators at Assembly School.

CHAPTER 4

Heimdall Implementation

Contents

4.1 Heimdall Monitoring System

The Heimdall Monitoring System is designed to provide a solution for monitoring, logging, and exporting user usage of Windows machines. It leverages a combination of a Windows service executable and the Windows Event Log system to gather and process information. Here follows an image describing Heimdall's Application Logic

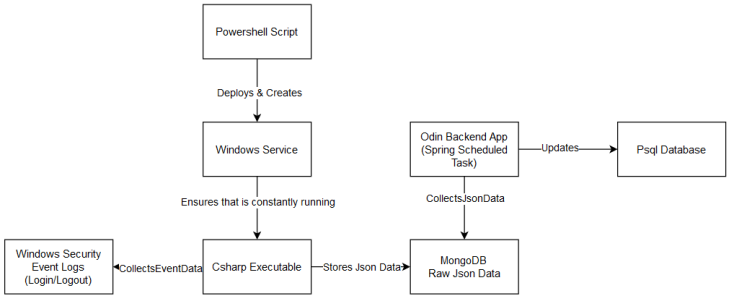

Figure 4.1: Heimdall Application Logic

4.1.1 App Logic

The Heimdall system processes security event logs to extract user information regarding activities such as logins, logouts, and screen locks. The executable queries Windows event logs to gather these events using specific event IDs. The system extracts necessary details such as the event type, username, and timestamp from the XML content of the event logs. Key steps in event log processing:

- Event ID Mapping: Identifies and maps relevant event IDs to event types (Lock, login, logout).
- XML Parsing: Extracts the username and timestamp from the event log's XML data.
- Data Formatting: Formats the extracted data into JSON objects.
- Validate: It validates the gathered data according to certain parameters
- Storage- Stores the relevant data (in this case, in a MongoDB)

It preforms all of this actions in a set amount of time, according to the last log relevant to this machine stored in the MongoDB as a way to avoid connection failures. We will know talk about the component structure, and how each of the steps work whilst discussing the logic behind the decisions made when implementing them, keeping in mind that the fact

that the computer logons in assembly are made using Azure AD accounts, which change the data that is stored in the event logs themselves.

4.1.2 Windows Event Logs

Windows Event Logs are a vital feature of the Windows operating system, providing detailed records of system, security, and application events. They help administrators and developers monitor system health, diagnose issues, and audit user activities.

Types of Windows Event Logs

- **System Logs**: Events logged by the operating system and its components, such as driver failures and hardware issues.
- **Security Logs**: Security-related events like login attempts and changes to security settings, crucial for auditing and system security.
- **Application Logs**: Events logged by applications and services, including errors, warnings, and informational messages.

Key Features

- **Event IDs**: Unique identifiers for each event type (e.g., login, logoff).
- **Timestamps**: Precise time information for when events occurred.
- **Event Sources**: Identifies the origin of events (operating system, applications, etc.).
- **Event Data**: Detailed information about each event, often in XML format.

In the Heimdall Monitoring System, Windows Event Logs are used to capture user activities such as logins, logouts, and screen locks. Heimdall processes these logs to monitor and log user activities. We elected to use event logs as a convenient and already existing way of tracking said information, without the need for more software and solutions. For that we will be using the security logs specifically, making use of specific event logs. we elected to use event logs with the ids: 4647, 4648 and 4800. These are the events for User initiated logoff, A logon was attempted using explicit credentials and The workstation was locked, respectively. 4647 and 4648 were picked due them happening only once, when the user logs out or logins into an account. There is a chance that the event 4648 happens twice, due to permission scaling (when a certain user is an admin). This is taken into account on the processing side of these outputs, and we will talk about it later, when we discuss the Heimdall processing component of the Odin app. 4800 was needed due

to sometimes students locking the screen instead of simply logging out, but when a user reenter the computer a new 4648 is issued, removing the need for a 4801 event, which is commonly used for reentering a locked workstation.

Other relevant information

Its also very relevant to talk about how the usage of AzureAD accounts influences the the event logs. Besides changing what events can be used, there is a specific feature, that very much alters how the logs outputted by Heimdall need to be processed. When a user logins into a AzureAD account they use their email. But when they log out the account that is logged in has a name. This is defined in the SubjectUserName section of the log data, in the XML. So there is a need to take that into account when processing the logs.

4.1.3 Component Structure: LogService

The LogService component is a critical part of the Heimdall Monitoring System, focusing on the gathering of significant logs from windows event logs, using event log queries.

This component has one main method to be called from the outside of it, and several supporting methods

`List<Dictionary<string, string>> GetEventsAsAzure(DateTime fromTime):`

This method is designed to retrieve specific events related to Azure AD account logins and logoffs. It uses a set of predefined event IDs to filter relevant events from the Windows Security log after fromTime. The method performs several key operations in the following order using support methods:

- **Event Query**: Constructs a query to filter events based on specific event IDs.such as driver failures and hardware issues.
- **XML Parsing**: Converts event records into XML format and extracts necessary information using XPath queries.
- **Data Mapping**: Ensures the data conforms to the required format and verifies the user account as an Azure AD account.
- **Data Validation**: Ensures the data conforms to the required format and verifies the user account as an Azure AD account.
- **JSON Serialization**: Converting the structured data into JSON format for storage and further processing.

During this process all of the relevant data from the outputted to a List of Dictionaries, each dictionary being the relevant data from a log.

4.1.4 Component Structure: HeimdallWindowsService

The HeimdallWindowsService is the primary component responsible for managing the Windows service life cycle, logging relevant tasks, and calling the necessary methods. This component is designed to ensure continuous monitoring and logging of event logs in a Windows environment.

Detailed Description

The HeimdallWindowsService component is implemented as a background service that periodically (every ten minutes) calls for logging of events, giving it the time since the last successful logon from this device, by calling the get last event stored from its repo component. The events are then formatted into JSON objects. This data is then stored in the MongoDB by calling the store method of its repo component.

This ensures that if a network failure, or some other issue arises with accessing the repo component, there is no problem with ensuring no logs are lost, since it gets all logs since the last successful storage.

4.1.5 Component Structure: MongoRepo

The MongoRepo component is responsible for managing interactions with the MongoDB database. This includes establishing connections, performing database operations, and ensuring that the application can store and retrieve data.

The MongoRepo component is configured to connect to a MongoDB database using a specific connection URI, given when this app is compiled.

4.1.6 Component Structure: Program

The Program component sets up and runs the Heimdall Windows service. It configures service dependencies and initializes the logging aspects of it. The structuring of this component is rather standard for CSharp applications, and eases the adding of dependencies and differences features, by allowing them to be injected via this file.

4.2 Communication with Assembly Odin

The collected event data is formatted into JSON objects and stored in a MongoDB database. This process is periodic and independent of the main application (Odin). Heimdall ensures minimal data requirements are met and allows for flexible data storage solutions. In this specific implementation we elected to include the machine id from where the

log originates to ease the feature of being able to harness the last log from said machine from mongo, to simplify retry logic, avoiding it, instead we only need to fetch all logs since the last successful one every time. Also this causes the need to process more logs at once, its a much better trade of than using other retry systems, as saving app states, last logins times locally or such, as these can be tampered with, but that is not doable with this solution. Its also important to mention the possibility of errors with the logs. Although it was possible to have a more thorough approach to log correction on the Heimdall side of it, one of the main points for Heimdall was making it as least intrusive and light as possible, reducing the need for software patches, as these will require a lot of overhead (As Heimdall is meant to be installed in a lot of machine at a time). Instead the responsibility to ensure the validate the logs is given to Odin, which is easier to patch and update.

4.3 Deployment and Scalability

Heimdall is meant to be deployed as a Windows service using a Power Shell script running as admin. This deployment strategy ensures that this program can be deployed via Microsoft Intune, which was one of the features Assembly asked for. Other relevant topic to discuss in this section is the scalability of this app. As the app itself is rather simple its only long term points of failure are the possibility of a MongoDB location change, which will force a update of all Heimdall instances, which is a issue that needs to be taken into account. We studies several possible solutions, but we didn't arrive at one that would satisfy our current predicament. One of said solutions was the possibility of creating a config file or a ENV variable, where Heimdall would get the desired link for our MongoDB. This would be a good idea, but the possible of tempering is too big of a risk with this app, as a user tampering with that feature would cause issues with detection. At a later date, with more time, we plan to circumvent this, to implement log health checks on the Odin side, or even on a possible middle ware to handle the Logs on the server side, which was not initially done, due to time constraints, but would be a good aproach at a later date to avoid long term issues with this feature.

CHAPTER 5

Backend Implementation

Contents

5.1 Backend Development

5.1.1 Setting Up the Database

In this section, we will discuss the proposed solution for the database utilized in our project. This database will store all essential information, including user data, theoretical and practical class details, and the points management system, which is the primary objective of our work.

We have chosen PostgreSQL as our database technology due to its reliability, data integrity, and extensive feature set. PostgreSQL offers advanced capabilities that are crucial for handling the diverse data requirements of our system. We designed the Entity-Relationship (ER) model for our database, which will be presented next. This model outlines the structure and relationships of the data, ensuring efficient storage, retrieval, and management of information.

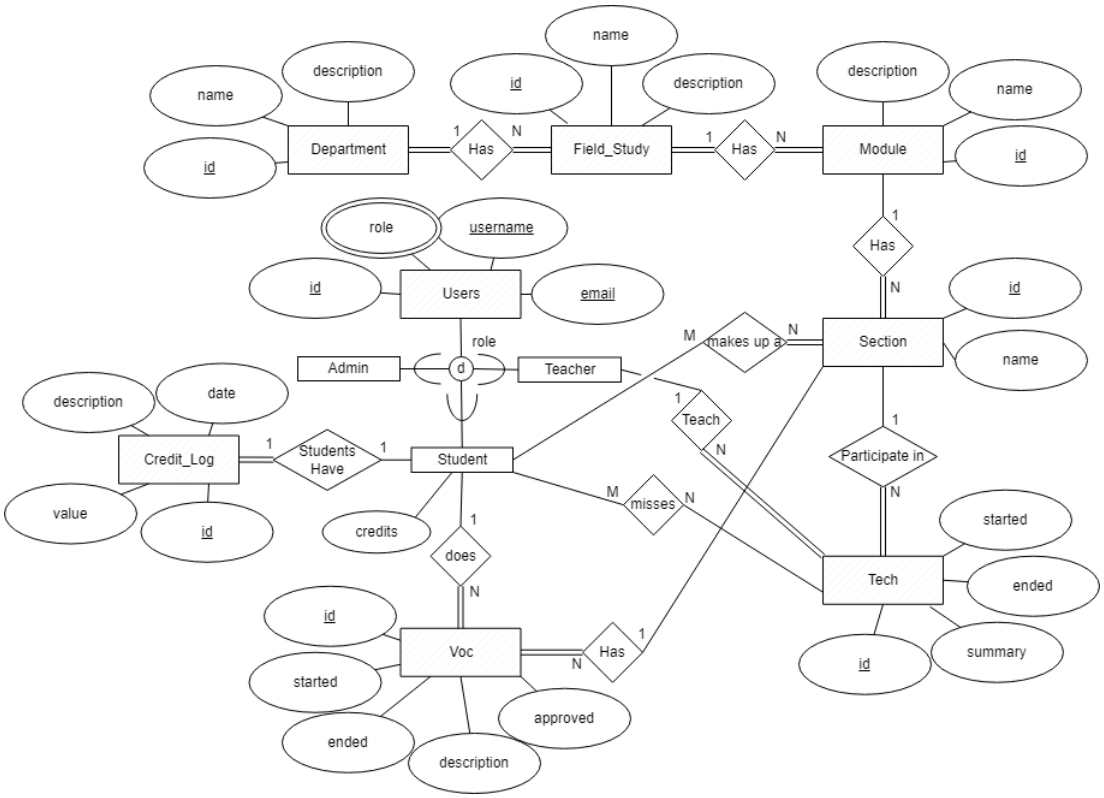

Figure 5.1: EA Data Base Model

Our database model consists of eight entities representing the entire database structure. The foundation of our application is the **User** entity 5.2. This entity is designed to integrate with the chosen authentication method, Office 365 Single Sign-On (SSO), and provide users with relevant information while using the web application.

The **User** entity includes the following attributes:

- **Username**: The unique identifier for each user.
- **Email**: The user's email address will also be unique.
- **Role**: Defines the user's access level within the application.

The **Role** attribute differentiates access levels within our web application:

- **Student**: Can manage their own resources.
- **Teacher**: Can manage their resources and those of their students.
- **Admin**: Has comprehensive management access to all application resources.

Since the objective of our application is to manage the points of a **Student**, we will create a **Student** generalization, which be a child of the **User**.

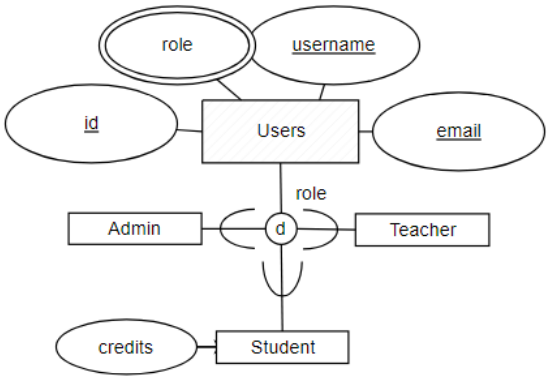

Figure 5.2: User and Relations

Besides defining the user entities, we also need to define entities for **Theoretical Classes** and **Practical Classes**. These entities will have a module representing the courses that students and teachers will be engaged in. They are crucial for maintaining the organization of scheduled, conducted, and attended classes.

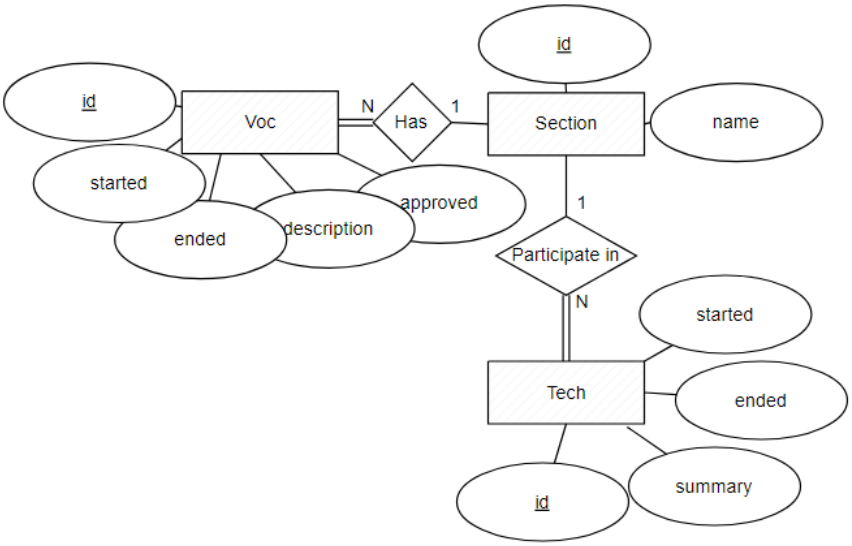

Figure 5.3: Tech and Voc relation to Section

Starting with the theoretical classes, these are designed for teachers to deliver theoretical content for a specific module within the Assembly School. The **Theoretical Class** entity includes the following attributes:

- **Started**: The date the class is planned to start.
- **Ended**: The date for the class to end.
- **Summary**: A brief overview of the class content.

The **Theoretical Class** entity is related to Students with a miss tech relation, which records student absences for that class. This relationship also allows students to see which classes they are enrolled in and their attendance status.

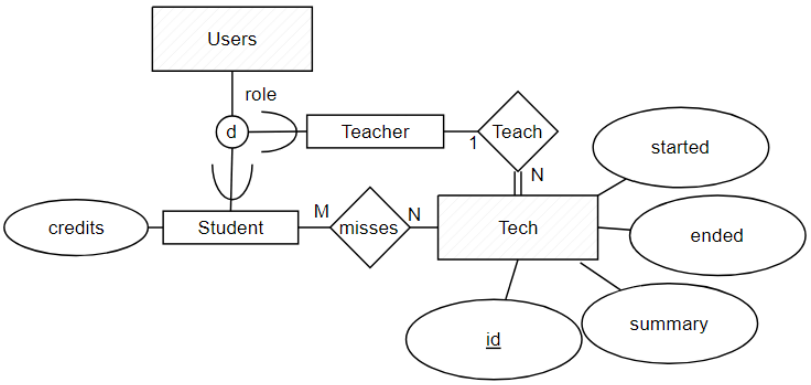

Figure 5.4: Tech and User Relation

For practical classes, the process is similar, but these classes can be scheduled by either students or teachers. As such, there are a few differences in their representation. The **Practical Class** entity includes the following attributes:

- **Description**: A brief overview of what the student needs or will do in the class.
- **Aprproved**: Confirmation given by the teacher.
- **Started**: The start date of the class.
- **Ended**: The end date of the class.

The duration of these classes is particularly important because it will be converted into points for the students. This conversion ensures that students are rewarded for the time spent in practical learning sessions.

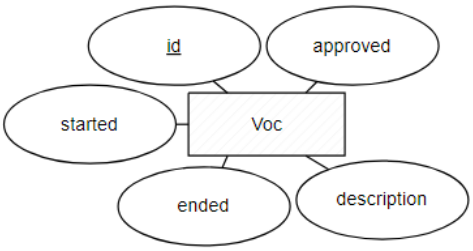

Figure 5.5: Voc

Modules are specific units of study or learning components within a Field of Study. They represent individual topics or courses that can be studied. Fields of Study are more specific divisions within a category, refining and segmenting the Department into more detailed areas. Departments are broad and general groupings that organize large areas of knowledge or topics.

This hierarchical organization allows us to structure content or information in a clear and organized manner.

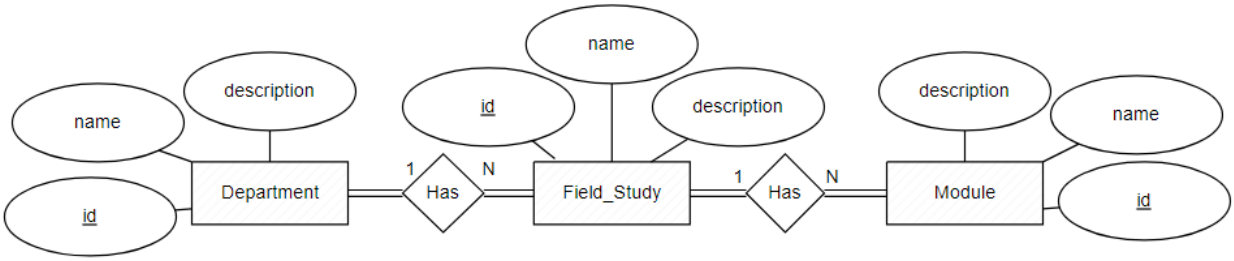

Figure 5.6: Departments, Fields of Study and Modules

The entity Sections will be used to represent a class associated with each module taught in each practical lesson. This entity is fundamental for the efficient organization of both practical and theoretical lessons, as well as facilitating attendance tracking for each student in the classes. This entity will only include the class name, its module, and the students present in that class.

Figure 5.7: Sections

5.1.2 API Development

In this section, we present the proposed solution for the construction of the API. We will begin by discussing the use of the Spring Web MVC framework for API development and the rationale behind choosing this specific programming language. Following this, we will examine the advantages and disadvantages of various database access technologies, providing a justification for the chosen approach. Lastly, we will dive into the authentication and authorization module.

The API can be divided into five parts:

- **Filters** – Responsible for intercepting requests, used to verify their content or to add information before reaching the handler responsible for processing them. Examples of these filters are those provided by Spring Security, which handle authentication and authorization checks when necessary.

- **Controllers** – Responsible for handling requests, their main objective is to obtain the information sent, transform it into domain objects, and route it to a specific function in the next layer, the service. They are also responsible for indicating necessary permissions to access different resources, to be used by the previous layer.

- **Services** – They know the business logic, understand how to obtain information or execute a given action, usually by using operations provided by various repositories. They are also responsible for performing additional authorization checks for resources that require more complex authorization.

- **Repositories** – Persistence layers, their goal is to provide a data access interface.

The architecture of the Web API is shown in Figure 3.11.

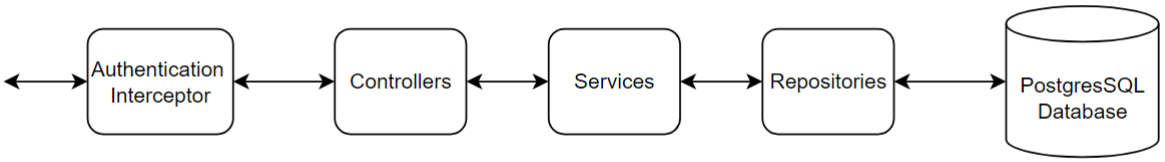

Figure 5.8: API Architecture

Framework Spring Web MVC

To implement the Web API, the Spring Web MVC framework was utilized along with the Kotlin programming language. Kotlin code is considerably more concise compared to Java, which accelerates the development pace. Kotlin allows us to leverage both object-oriented and functional programming principles, which are native to the language. Moreover, the Spring framework provides full support for the Kotlin language. Spring is the most popular infrastructure within the Java ecosystem, with Spring Web MVC being a variant of the Spring framework.

The main advantages of using the Spring Web MVC framework in the project are:

- Simplified configuration through annotations on the application's objects.
- Provides an HTTP server within the applications.
- Automates configurations related to web application deployment, such as assigning a method to a specific route and allowing these definitions to be changed if necessary.
- Based on dependency injection, which facilitates the use of other libraries and infrastructures within the application.
- Integrated support for creating a REST API.

Database Access

In our system, we considered three options for accessing the PostgreSQL relational database: Java Database Connectivity (JDBC), Java Database Interface (JDBI), and Java Persistence API (JPA). Each option was evaluated based on code organization, control over queries, and ease of use. The conclusions are as follows:

- Using JDBC implies less organized code and, consequently, a worse user experience. However, it allows full control over executed queries.

- JPA offers better code organization, leading to a good user experience, but it removes all control over queries.

- JDBI makes the code relatively compact, allowing for good code organization. Similar to JDBC, it provides full control over the queries.

Given our requirements, we opted to use JPA with Spring Data and Hibernate for the following reasons:

Java Persistence API (JPA)

JPA allows us to manage relational data in Java applications using a domain model. The primary features include:

- **Annotations**: JPA uses annotations to map Java objects to database tables, making the code cleaner and more readable.

- **Entity Management**: JPA handles the lifecycle of entities, including CRUD operations, which simplifies data access logic.

- **Query Language**: JPA provides JPQL (Java Persistence Query Language), a platform-independent query language that simplifies complex query operations.

5.1.3 Spring Data JPA

Spring Data JPA builds on top of JPA, providing additional features to reduce boilerplate code and enhance functionality:

- **Repositories**: Spring Data JPA introduces repositories, which are interfaces for data access that eliminate the need for boilerplate code.

- **Query Methods**: It allows the creation of query methods in repositories by simply defining method signatures, reducing the need for manual query definitions.

- **Pagination and Sorting**: It supports pagination and sorting out-of-the-box, making it easier to handle large datasets.

5.1.4 Spring Security Configuration

The Web API's security configuration leverages Spring Security to handle authentication and authorization using OAuth2 with Office365. Here's a detailed overview of the configuration:

- **CORS and CSRF**: CORS is enabled to allow requests from different origins, and CSRF protection is disabled to simplify the authentication process.
- **Authorization**: The configuration allows unrestricted access to the root (/) and requires authentication for all other requests.
- **OAuth2 Login**: OAuth2 login is set up to use custom user services and success handlers, ensuring user data is processed correctly during the authentication process.

Filter Implementation by Spring Security

Instead of using a custom filter for JWT, Spring Security automatically handles the filtering process for OAuth2 authentication. This is accomplished through the following configuration:

OAuth2 User Service

A custom service loads user details and ensures the user's information and roles are correctly managed in the PostgreSQL database.

Session-Based Authentication

In the current Spring Security configuration, we use OAuth2 for authentication, relying on session-based authentication rather than JSON Web Tokens (JWT). Here is a explanation:

Session-Based Authentication:

- **Session Management**: When a user logs in using OAuth2, Spring Security creates a session for the authenticated user. This session is identified by a JSESSIONID cookie, which is sent to the client and included in subsequent requests.
- **Server-Side Storage**: The server maintains the session state, and the session ID (JSESSIONID) is used to look up the user's session information stored on the server.

Upon successful OAuth2 authentication, Spring Security creates an HTTP session and sends a JSESSIONID cookie to the client. The client includes the JSESSIONID cookie in all subsequent requests. The server uses the session ID to retrieve the user's session and determine if the request is authenticated.

Configuring Azure AD App Registration

To integrate OAuth2 authentication using Office365 with our Spring Security setup, we configured an Azure AD App Registration. This setup involves several key steps, briefly outlined below:

- **App Registration Creation**: Was created a new app registration in the Azure portal, providing it a unique name and setting it up to support the necessary account types.
- **Authentication Configuration**: Under the app's authentication settings, we added the necessary redirect URIs and configured the platform settings to ensure both ID tokens and access tokens are issued during the authentication process.
- **API Permissions**: Granted the app permissions to access the Microsoft Graph API, including permissions such as openid, profile, email, and User.Read. Admin consent was granted to allow the application to access these resources on behalf of the users.
- **Client Secret Generation**: A client secret was generated to secure the application. This secret, along with the client ID, is essential for the OAuth2 authentication flow. These credentials are stored securely and used in the application's configuration.

CHAPTER 6

Frontend Implementation

Contents

6.1 Odin Web Application

6.1.1 Architecture and Design

Architecture and Design

The frontend application is designed as a Single Page Application (SPA). It uses hooks to maintain the state and perform queries. This architecture ensures that the application can provide a seamless user experience by dynamically updating content without needing to reload the entire page. The SPA architecture allows for a responsive and interactive interface, enhancing the overall user experience.

Application Structure

The structure of our application is as follows:

- **public**: The public folder is a common directory in front-end projects that contains static files such as index.html.
- **src**: The src folder is where the application's source code is located and will be discussed in more detail later.
- **package.json**: The package.json file is a fundamental file in projects as it describes the project and its dependencies.
- **package.json**: The package.json file is a fundamental file in projects as it describes the project and its dependencies.
- **tsconfig.json**: The tsconfig.json file is a TypeScript configuration that specifies the compiler options for the project.
- **webpack.config.js**: The webpack.config.js file is the configuration for webpack, a modern build tool for JavaScript.

For the division of our source folder, we use the following architecture:

- **assets**: Used to store all the static resources of the application, such as images.
- **components**: Contains reusable user interface components.
- **hooks**: Stores custom React hooks.
- **pages**: Contains components that represent entire pages of the application.

- **services**: Used to store business logic and interactions with external APIs.
- **session**: Manages logic related to user sessions.
- **utils**: Contains utility functions and helpers that are used throughout the project.

6.1.2 Componemt Structure

App

This component is the main container of our React application, housing all other components. It includes the routes for our application and a layout component called Dashboard, which defines the overall layout. The content for the routes will be rendered within this layout. Note: The routes are placed inside the Dashboard, but it may not be the most correct way to describe it; please verify or consult if needed.

Dashboard

As mentioned earlier, the Dashboard serves as the layout component of our application, containing various sections. The Appbar is located at the top and primarily enhances the application with the Assembly logo. The Drawer is our sidebar, containing all functionalities such as class creation, profile viewing, logout, and more. The Main Content area dynamically renders content based on user interactions, ensuring it doesn't overlap with the Appbar. Depending on the user and their permissions, different information and functionalities will be accessible.

Session Provider

Our Session Provider plays a crucial role in user authentication and information storage, especially for components like the Dashboard that rely on user role data to display appropriate information. It uses React's context API to make user-related data available to all other components.

We will have our Session interface, representing the user's session and encapsulating relevant authentication details. The Session Manager interface will provide methods for managing the user session, such as ClearSession when the user logs out. There is no need for a setLogin method because, with Spring Security, all routes are protected. Upon launching our app, a request is always made to our server, and if the user is not authenticated, they are redirected to the login page.

```
interface Session {
    username: string;
    email: string;
    role: string;
}
```

```
export interface SessionManager {
    sessionData: Session | null;
    set: (credentials: Session) => void;
    clear: () => void;
}
```

The SessionContext.Provider will wrap all child components, thereby providing an instance for session management and associated methods within this context.

6.1.3 Pages Structure

The pages in our frontend are organized consistently throughout the application, starting with the use of a corresponding hook for each page. This hook is responsible for managing the page's state, performing operations to fetch or store information from the backend, and providing additional functionalities, such as search mechanisms used throughout the web application. Utilizing the methods provided by this hook, the page structure is built with the help of components. These components help reduce repetitive code, making the code more readable and maintaining a clearer presentation. The same components can be reused in other parts of the application, promoting code reuse.

6.1.4 User Interface

The choice of this interface was based on the need to organize and limit access to system functionalities according to each user's role. This ensures that:

- **Security and Privacy**: Each user group (student, teacher, and admin) has access only to the information and functionalities necessary for their specific roles, thus protecting sensitive data.

- **Ease of Use**: The interface is clear and intuitive, making navigation easier for users and ensuring they can quickly access the functionalities they need.

- **Efficiency in Management**: Allowing admins full access facilitates overall system management, while teachers can focus on managing their classes and sections, and students can access information relevant to their progress.

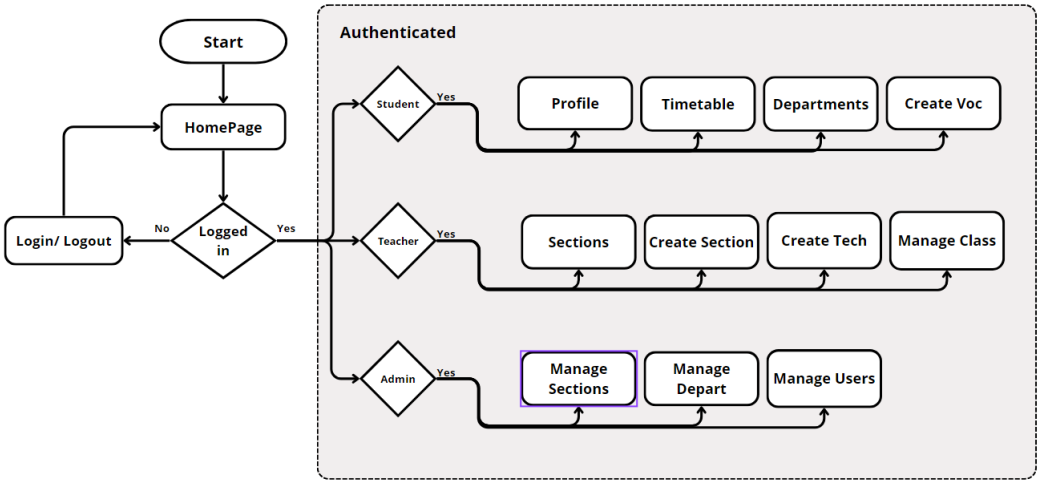

Figure 6.1: User Interface

6.1.5 Integration with Backend

Interaction with Odin Backend Services

The frontend application interacts with the Odin backend services by calling its API to populate the fields within the SPA. This interaction is crucial for dynamically displaying the latest data and ensuring that user actions are reflected in real-time.

CHAPTER 7

Development and Testing

Contents

7.1 Development

The development process was rather extensive, encompassing various stages of planning, implementation, testing, and refinement. The following sections outline the key aspects of our development methodology and the tools we utilized.

7.1.1 Agile Development

We adopted an agile development approach to manage and execute our project. This methodology allowed us to be flexible and adaptive to changes and was chosen with the main intent of having a functional project at every step, so that we could keep testing it, and gain valuable insight. Although this organization looks to be made for waterfall, its just meant to be used to understand what components of the application are developed at each stage.

- **Iterative Development**: We broke down the project into smaller, manageable tasks and developed them in iterations. This approach helped us focus on delivering functional parts of the project regularly, and can be checked in the following picture, where the steps are outlined.

- **Sprints**: We organized our project development into sprints with two weeks of length each, having bi-weekly meetings at the end of each of them.

- **Bi-weekly Discussions**: Every two weeks, we held discussions to review the progress of our assigned work and tasks. These meetings were crucial for identifying any blockers, re-evaluating priorities, and planning the next steps. We also included one colleague at Assembly that helped us with the paper of Product Owner. He helped us pick the most important features to invest in at certain stages, to ensure our vision was meet.

7.1.2 Usage of GitHub

GitHub was an integral part of our development process. It served as the central repository for our codebase and provided various tools to facilitate project management and collaboration.

We organized our repo in a tree as follows. The main branch is meant for code deliveries, the dev branch is for code that is under development, but has no major issues, and the feature branches serve as a way to fix assigned issues, without disturbing the code base.

- **Code Hosting**: Our entire codebase was hosted on GitHub, ensuring easy access and version control. This setup allowed team members to contribute to the project from different locations, maintaining a consistent and up-to-date code repository.
- **GitHub Issues**: We used GitHub Issues to track bugs, feature requests, and other tasks. Each issue was tagged and categorized to provide a clear overview of the project's status. This system helped us prioritize and address tasks systematically.
- **GitHub Projects**: To further streamline our agile process, we utilized GitHub Projects with a Kanban board. This tool allowed us to create and manage project sprints, where we could visualize and organize our tasks for each sprint in columns representing different stages of development (e.g., To Do, In Progress, Done). This helped stream line communication and have staff at assembly give opinions on the project development.
- **Feature Branches**: Our feature branches were made with GitHub Issues as a baseline. Each feature branch was created to fix a specific issue or target a particular feature. This approach ensured that development was organized and that each branch addressed a well-defined goal, making it easier to track progress and manage code merges. This also was a massive help later on, as it facilitated backtracking on certain issues, and ensured the code present in the dev branch was always up to standard.

7.2 Beta Test

7.2.1 Methodology

The overall approach for beta testing was hands-on and interactive. We engaged a group of students to use the software in a controlled environment to assess its functionality and usability. The testing phases included:

- **Usage Session**: Students were asked to use the software for a few minutes. During this session, they were specifically instructed to create a VOC (Voice of Customer) to test different features of the software.
- **Gaming Test**: Students also used gaming PCs with Heimdall installed to identify any abnormal behaviors and to ensure the software accurately tracked their points.

The primary goal was to identify bugs, assess usability, and gather user opinions. Feedback was collected through a Google form filled out by the students after their session.

Beta testers were selected based on age groups (9-12, 13-14, 15-17) to ensure diverse feedback. Nine students participated, and their personal information remained anonymous throughout the testing process. The results for the google forms done during this beta for student feedback can be seen in Appendix A.

7.2.2 Challenges Faced

During the beta testing phase, several challenges were encountered:

- **Processing Heimdall Logs**: Initially, there was difficulty in processing the Heimdall logs due to a delay in the implementation. This issue was resolved a few days later, allowing us to use the logs stored in MongoDB to test the software's efficacy.
- **Office 365 Redirection**: Another significant challenge was ensuring proper redirection from Office 365. The resolution required deploying the application with SSL certificates to enable proper redirection via Office 365 Azure. However, this issue remains unresolved due to two main factors:
  - Lack of ownership of a valid domain, which we have asked for Assembly to buy, but is still pending for the second issue.
  - Issues with the router at Assembly's production server location, which is scheduled for repair.

7.2.3 Analysis of Results

The beta testing yielded valuable insights:

- The software successfully managed to export logs for gaming sessions with success, and those logs could be read by Odin eventually, which was a huge success. It also told of us of the small issues with tuning on Heimdalls side, as the presence of a lot of "useless" logs (not student relevant logs).
- Feedback from the students provided crucial information on the usability of the software. Based on their input, we identified areas for improvement, particularly in the user interface and we also got confirmation of the need for the software from them.
- The testing confirmed the effectiveness of the software's core functionalities, although some technical issues needed to be addressed.

CHAPTER 8

Deployment

Contents

8.1 Deployment

Below, we outline our deployment strategy, environment setup, deployment process, challenges faced, monitoring and maintenance.

8.1.1 Deployment Strategy

Our deployment strategy primarily relied on manual deployment using Docker. This approach was chosen to meet our specific needs for consistency rather than continuous updates. The goal was to have a reliable and stable deployment process that could be executed and verified manually.

- **Manual Deployment with Docker**:
  - **Reason for Choice**: We prioritized a deployment strategy that provided consistency and reliability. Continuous deployment (CD) was not necessary for our use case, as our primary requirement was to ensure that the application worked consistently and reliably whenever deployed, not that it was ideal for continuous development.

8.1.2 Environment Setup

Development Environment

- **Local Development**: During development, the software was run locally. We used a proxy configuration on a frontend Webpack to communicate with the backend services. The backend services accessed a PostgreSQL database running in a Docker container.
- **Heimdall Setup**: For Heimdall, we set up a virtual machine (VM) assigned to be a part of Assembly's network. This VM was treated as a separate machine and connected to a MongoDB instance hosted in the cloud. This setup allowed us to simulate real-world scenarios and test Heimdall effectively.

Production Environment

- **Docker Containers**: For the production environment, all components of the software were deployed inside Docker containers running on an on-premises server at Assembly. This included the backend services, frontend application, and databases.
- **Nginx Proxy**: We utilized an Nginx proxy to handle incoming requests to the Odin frontend and API.

- **Heimdall Deployment**: Heimdall was designed to be deployable to relevant machines within Assembly. This involved creating a Windows service using a PowerShell script, which ensured that Heimdall ran continuously on the target machines.

8.1.3 Deployment Process

The deployment process involved several steps to ensure that the application was correctly set up and configured in the target environment.

- **Package Transfer and Deployment**:
  - **Using Bitvise**: We used Bitvise, an SSH client, to transfer the deployment package containing the application code to the production server.
  - **Running Docker Compose**: Once the package was on the server, we ran Docker Compose to build and deploy the Docker containers. This ensured that all services were correctly instantiated and linked together.
- **Heimdall Deployment**:
  - **Code Transfer**: The compiled code for Heimdall was transferred using a USB drive.
  - **PowerShell Script**: We executed a PowerShell script on the target machines to create a Windows service for Heimdall. This script ensured that Heimdall started automatically and ran in the background, providing continuous monitoring and logging capabilities.

8.1.4 Challenges Faced

During the deployment process, we encountered several challenges that required careful attention and problem-solving.

- **Office 365 Redirection**:
  - **Issue**: Ensuring proper redirection from Office 365 required deploying the application with SSL certificates. This was a complex challenge due to the restraints imposed by Office 365 SSO system itself which didn't allow us to set a redirection IP that wasn't local host or a https domain.
  - **Resolution**: The issue remains partially unresolved due to lack of access to Assembly's domain and issues with the router at Assembly's production server location. These are scheduled to be addressed to enable proper redirection.

CHAPTER 9

Evaluation and Future Work

Contents

9.1 System State

The Assembly Odin project has made significant strides in automating and improving the management of student activities at the Assembly School of Technologies. We have successfully developed a functional website and backend application that has been well-received by both students and staff, despite some initial issues with redirects. The Heimdall component is fully operational, sending logs to MongoDB, which Odin can efficiently read and process.

9.2 Challenges Faced

During the project, we encountered several challenges that needed to be addressed:

9.2.1 Redirect Issues

There were initial problems with redirects within the application, causing inconvenience to users. These issues already have a solution planned, and hope to be fixed in a near future.

9.2.2 Data Synchronization

Ensuring data synchronization between the Heimdall component and the Odin application posed significant challenges. The team had to invest a lot of time into a architecture to fix and mitigate this issues

9.2.3 Scalability Concerns

As the application has taken into account the growth of the user base, scalability became a critical concern. The system needed to handle an increasing amount of data and user interactions without compromising performance. This required careful planning on the architecture front to ensure this issues could be addressed in the future when they arise, without being hampered by the baseline architecture

9.2.4 User Interface and Experience

Designing an intuitive and user-friendly interface was challenging, especially given the diverse user base, including students, teachers, and administrators. Continuous feedback and iterative design were necessary to improve the user experience.

9.3 Future Work

The current implementation of Odin's component that reads incoming data from Heimdall via MongoDB is in a simplified state. We plan to expand this component into a comprehensive middleware service to enhance maintainability and logic in the main application. Our future work includes:

9.3.1 Middleware Development

Developing a full-fledged middleware service to handle data processing more efficiently. This will simplify the main application's logic and improve maintainability.

9.3.2 Extended Testing Period

We aim to conduct a longer-term testing period (approximately one month of real usage) to gather more data on system performance and user interactions. This will help us identify areas for improvement and make necessary adaptations.

9.3.3 Integration with Additional Systems

Exploring the possibility of integrating Assembly Odin with other educational tools and platforms to provide a more holistic solution for managing student activities and learning processes, like moodle and microsoft teams.

9.4 Conclusion

In conclusion, the Assembly Odin project has successfully addressed many of the inefficiencies in the current manual system at the Assembly School of Technologies. The project has demonstrated the technical proficiency and innovative thinking of its developers. Future work will focus on expanding and refining the system to ensure it continues to meet the evolving needs of the institution. This also served as our first experience creating a real production software and the lessons that derived from it are invaluable to our future both as developers and engineers and people

Appendices

APPENDIX A

Appendix A: Beta test results

Timestamp	O quão satisfeito ficaste com o produto	O que achaste do aspeto visual do site	O que achaste da usabilidade	Do ponto de vista da existencia de features, o que achaste?	achas que este sistema tem a possibilidade de melhorar o teu dia à dia na Assembly?	Que features mais gostaste ou gostavas de ver investir mais em	Que features menos gostaste/achaste mais inuteis	Que features gostavas de ver adicionados?	Mais alguma coisa que gostavas de sugerir?
6/14/2024 15:51:52	5	1	3	5	5	poder adicionar vocs autoritarian	a cara, departments	free points	nao
6/14/2024 15:51:57	5	3	5	4	4	poder adicionar pontos automati	a profille picture	free points	nao
6/14/2024 16:55:15	4	3	4	3	5	gostaria que o site fosse mais ir	nenhuma	idk	nope
7/13/2024 22:55:12	3	3	3	4	5	No meu horario	esta muti branco	sair da app	nao
7/13/2024 22:56:05	2	3	2	4	4	nenhuma	das imagens	nada	tempo no site
7/13/2024 22:58:11	4	3	4	4	4	aspeto para ficar 5	imagem do meu prefil	nao consigo pensar er	nao
7/13/2024 22:58:23	3	3	3	3	3				
7/13/2024 23:00:11	3	4	4	3	3	esta quase bom	tema	dark mode	suporte
7/13/2024 23:00:55	2	3	3	2	3	em geral	animacoes	melhorar as pesquiisa	seguranca