

Semester Thesis
Documentation

Dashboard KPI for OST

Semester: Autumn 2024

Date: 19.12.2024

Project Team: Christoph Bodschwinna
Philipp Frank

Project Advisor: Laurent Metzger
Fabio Daniel Marti



School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Abstract

Introduction

Many companies today use Key Performance Indicators (KPIs) to monitor trends and to control their decision-making processes. The OST has also implemented KPIs to get a comprehensive overview of various aspects of their organization. Currently, these KPIs are manually tracked in a large Excel sheet. However, as the datasets grow, this approach becomes increasingly challenging to manage.

Objective

The goal of this semester thesis was to develop a prototype for a web application that could eventually replace the current solution. The application must support the creation and adjustment of KPIs and their formulas, include a user permission system to restrict access to certain data, and provide a method for entering new data. Additionally, it should allow for filtering and visualizing data sets in graphs to facilitate informed decision-making.

Conclusion

The prototype developed in this project supports all previously mentioned functionalities and introduces several new features. Users can add supplementary information to data entries, providing context for anomalies such as sudden spikes or drops in graphs. A date filter has also been added, allowing users to select specific time periods when necessary. Currently, the application supports importing large datasets directly from CSV files. In the future, the application could be further improved by retrieving data automatically from surrounding systems, reducing maintenance efforts and minimizing errors caused by manual user input.

In conclusion, this project has successfully delivered a prototype that improves confidentiality, accessibility, and user experience. Future enhancements will continue to refine the tool, ensuring it remains adaptable and meets evolving needs.

Keywords: Key Performance Indicators (KPIs), Web Application

Contents

I	Management Summary	1
1	Management Summary	2
1.1	Introduction	2
1.2	Technologies	2
1.3	Result	3
1.4	Conclusion	7
II	Product Documentation	8
2	Requirements	9
2.1	Functional Requirements	9
2.1.1	Use Case Diagram	9
2.1.2	Use Case Description	11
2.2	Non-Functional Requirements	16
2.2.1	Verification of Non-Functional Requirements	18
3	Domain Analysis	19
3.1	Domain Model	19
3.1.1	Explanations	20
4	Architecture	22
4.1	Stakeholders	22
4.2	Scope and Context	22
4.2.1	Interfaces	22
4.3	Solution Strategy	23
4.3.1	Web Application Architecture	23
4.3.2	User Roles	23
4.3.3	Calculation of KPIs	23
4.3.4	Data Format for KPIs	23
4.4	Software Structure	24
4.4.1	Backend Software Structure	24
4.4.2	Frontend Software Structure	26

4.5	Building Block View	27
4.5.1	Whitebox Overall System	27
4.5.2	Building Blocks	27
4.5.3	Level 1	28
4.5.4	Level 2	28
4.6	Deployment View	29
4.7	Architectural Decisions	30
4.7.1	Frontend	30
4.7.2	Backend	30
4.7.3	Database	31
4.7.4	Other	31
4.8	Encountered Problems	31
4.8.1	Issues with the Server	31
4.8.2	Incorrect Formulas	32
4.8.3	Change of Requirements	32
4.8.4	Limitations of UI Library	32
5	Quality Measures	33
5.1	Definition of Done	33
5.2	SonarQube	33
5.2.1	Quality Gates	34
5.3	CI/CD Pipeline	35
5.4	Test Concept	35
5.4.1	Testing Strategy	35
5.4.2	Test Environment	35
5.4.3	Test Deliverables	36
5.4.4	Test Schedule	36
5.4.5	Test Roles	37
5.4.6	Test Artefacts	37
6	Result	38
6.1	Functional Requirements	38
6.1.1	Notable Use Cases	39
6.2	Non-Functional Requirements	41
7	Conclusion	42
7.1	Result Reflection	42
7.2	Outlook	43
7.3	Closing Statement	43
	Bibliography	43
	List of Illustrations	44

List of Tables	45
Glossary	47

Part I

Management Summary

Chapter 1

Management Summary

1.1 Introduction

Key Performance Indicators (KPIs) are measurable metrics that help organizations track progress toward specific business goals. By offering clear, quantifiable data, KPIs provide valuable insights into the effectiveness of strategies, processes, and operations. They are used across various departments and industries to assess success, pinpoint areas for improvement, and guide data-driven decision-making.

KPIs can differ based on an organization's objectives, which may include financial performance, customer satisfaction, operational efficiency, or other critical areas of growth. The OST also tracks KPIs to monitor important trends within the organization.

Currently, KPIs are managed through a large Excel sheet, which is manually updated. As data sets continue to grow, this method is becoming increasingly difficult to maintain. The aim of this semester's thesis is to develop a prototype that can eventually replace the existing solution.

1.2 Technologies

The assignment was broadly defined, enabling an independent selection of suitable technologies. The web application is divided into two parts. The frontend handles most user interactions and is developed using TypeScript and React. The backend is responsible for data persistence in a PostgreSQL database and KPI calculations, implemented with Python and Django. Both parts are deployed on an Ubuntu virtual machine using Docker containers, which host the frontend, backend, and database. This architecture was chosen to ensure that each part can be replaced with new versions or alternative frameworks if required. Communication between the frontend and backend is facilitated through REST API calls.

1.3 Result

The project resulted in a user-friendly web application that incorporates a user permission system to control access to specific data based on user roles. This ensures that only authorized users can view or modify certain information. The application includes a login page for secure access and user settings, where they can update personal information such as their name, email, and password.

OST First Level Steuerung KPI

Ansichten

- Dashboard
- KPI Vergleich

Datenverwaltung

- Daten Import

Benutzereinstellungen

Benutzerprofil

Vorname
Admin

Nachname
User

Email *
admin@ost.ch

Speichern

Passwort ändern

Altes Passwort *
Altes Passwort eingeben

Neues Passwort *
Neues Passwort eingeben

Passwort bestätigen *
Neues Passwort bestätigen

Passwort ändern

AU Admin User
admin@ost.ch

Log Out

Figure 1.1: Web Application User Settings

The application includes two distinct dashboards, each designed to address specific requirements. Both dashboards offer filtering options based on date and provide additional settings to control whether invalid values are displayed and whether indicators for supplementary information should be shown on the graphs. Displayed data entries can be easily adjusted by clicking on them directly within the graph, which opens a form to modify the KPI variables for the selected data entry. Additionally, all graphs can be downloaded as PNG images for further use.

The first dashboard focuses on presenting all KPIs for a single institution within the organization, providing a detailed and comprehensive overview of its performance.

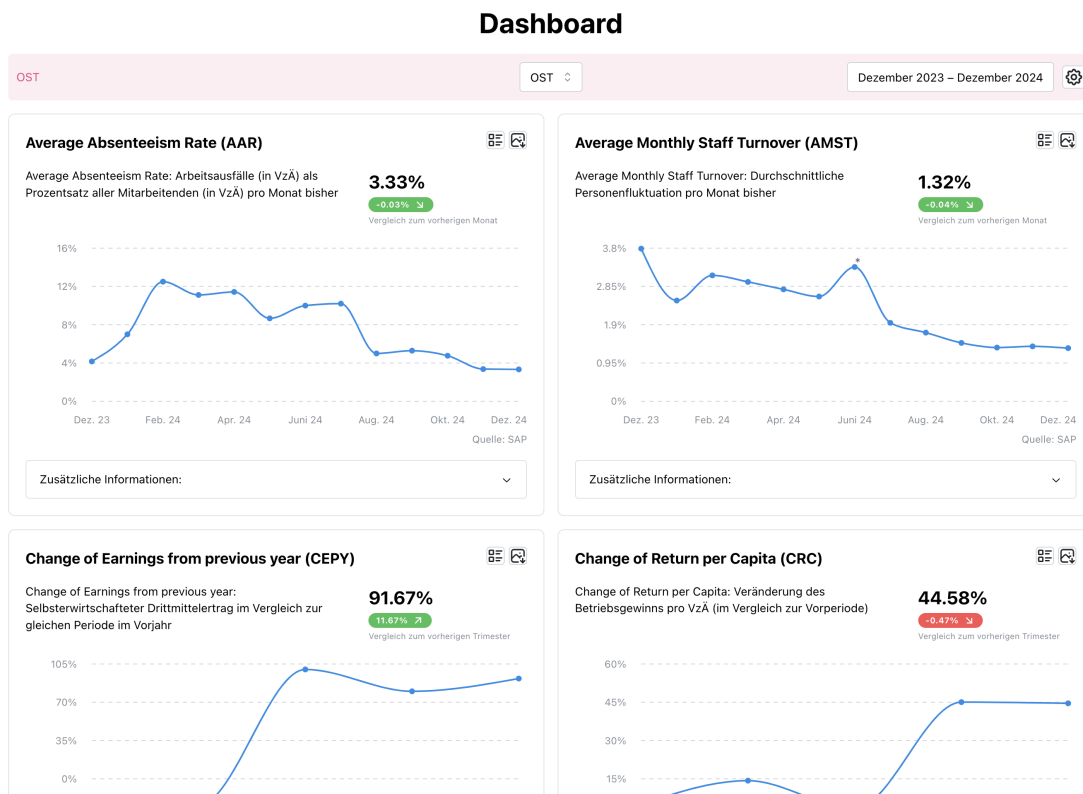


Figure 1.2: Web Application Dashboard

The second dashboard enables the comparison of KPIs across multiple institutions, offering insights into trends and variations.

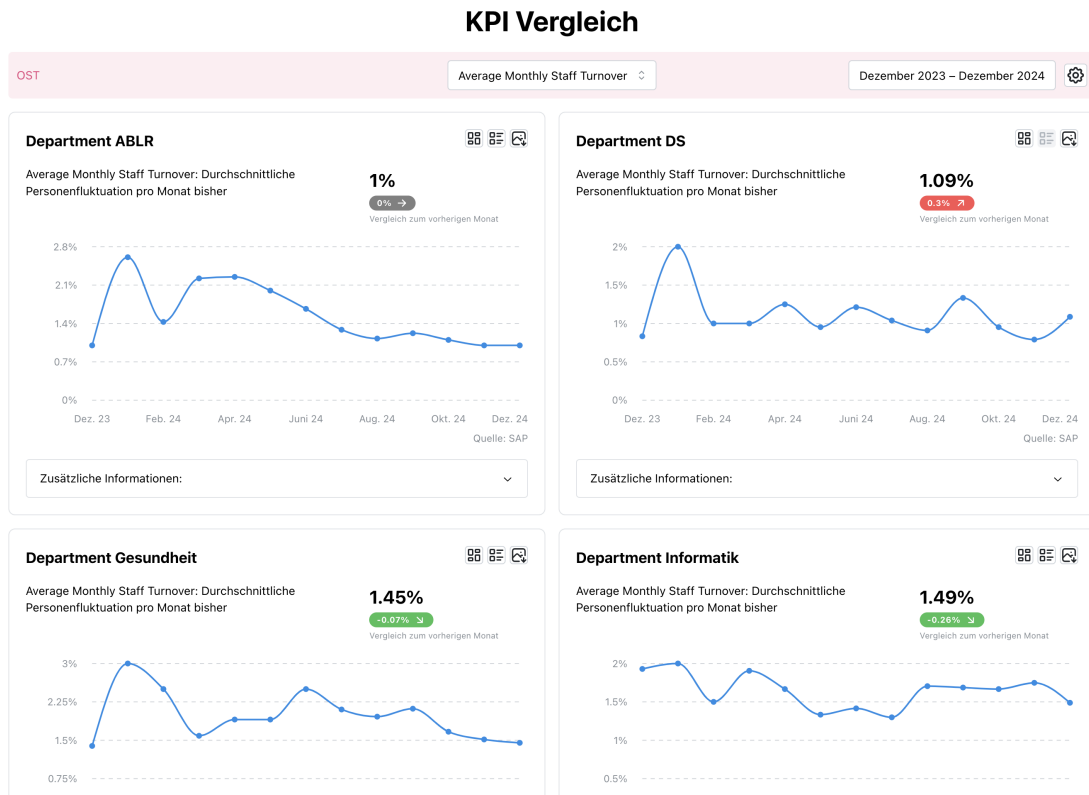


Figure 1.3: Web Application KPI Comparison

Additionally, the application streamlines the import of large datasets, making it easier to integrate new information for institutions. Users can also download a pre-configured import template tailored to the selected institution, ensuring valid data fields and consistency.

Daten Import

1 **Daten auswählen**
Formular ausfüllen

2 **Import**
Daten werden importiert

Institution *

CSV Datei *

Trennzeichen *

Figure 1.4: Web Application Import Form

At the end of the import process, a report is generated to provide feedback on its success or highlight any issues encountered. The report specifies problems such as missing user permissions for certain KPI variables, non-existent variables in the input data, or invalid values that were provided. This immediate feedback ensures, that users can quickly identify and address any errors.

Daten Import

1 **Daten auswählen**
Formular ausfüllen

2 **Import**
Daten werden importiert

✓ Datei lesen
Die Daten wurden erfolgreich für den Import vorbereitet.

📄 Daten importieren
Der Daten Import war erfolgreich.

📊 Report
Manche Datensätze konnten nicht importiert werden.

<p>Gesperrte Variablen:</p> <ul style="list-style-type: none"> 🟢 Keine 	<p>Inexistente Variablen:</p> <ul style="list-style-type: none"> 🟡 Field 1 🟡 Field 2 🟡 Field 3 	<p>Ungültige Daten:</p> <ul style="list-style-type: none"> 🟢 Keine
--	--	--

Figure 1.5: Web Application Import Report

An admin panel, provided by the backend, enables administrators to manage users, user roles, the organizational structure and configure the required KPIs, including their respective formulas. This ensures flexibility in adapting the system to evolving requirements while maintaining consistency and accuracy in KPI calculations.

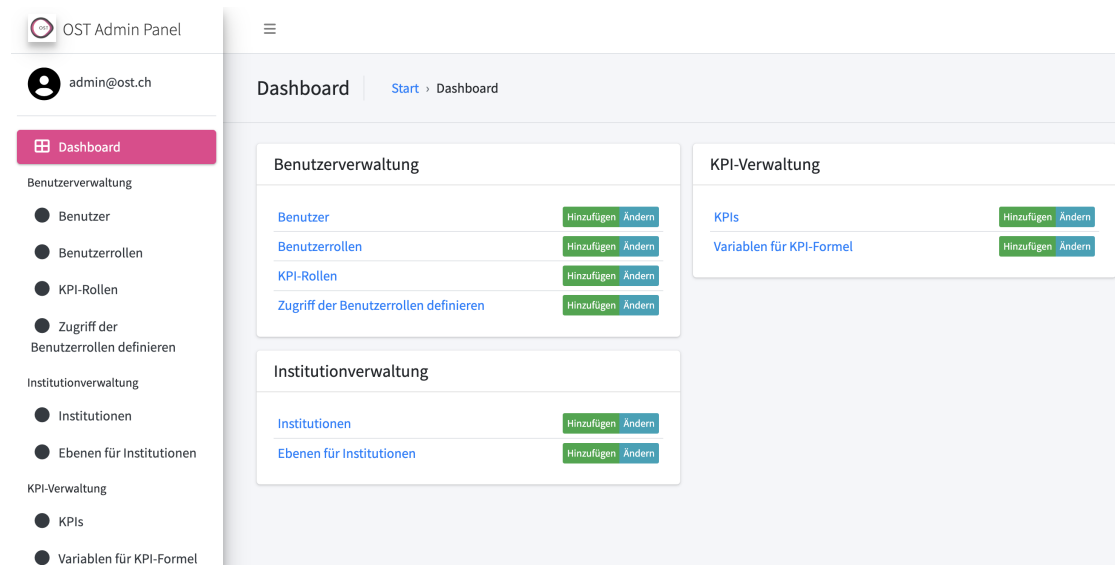


Figure 1.6: Web Application Admin Panel

1.4 Conclusion

The developed prototype introduces several new features. A permission system has been implemented to define relevant KPIs for specific institutions while restricting user access as needed. Users can now add supplementary information to data entries, providing context for anomalies such as sudden spikes or drops in graphs. Additionally, graphs can be easily downloaded for inclusion in reports or presentations. A date filtering feature has also been implemented, allowing users to select specific time periods for more targeted analysis.

Despite these advancements, the data gathering process still has room for improvement. At present, users are required to manually collect data from external systems, prepare it for import and complete the upload process. Automating data retrieval directly from these systems would significantly reduce the time currently needed to maintain the data and minimize errors associated with manual input.

In summary, the project successfully delivered a prototype that enhances data confidentiality, accessibility, and overall user experience. Future improvements will ensure that the tool remains adaptable and continues to meet evolving needs.

Part II

Product Documentation

Chapter 2

Requirements

2.1 Functional Requirements

2.1.1 Use Case Diagram

Actors

- User:
 - Goal: Log in to their account.
- Authenticated User:
 - Goal: Enter new KPI data, view the dashboard, and update their password.
- Admin:
 - Goal: Manages KPIs, users and user roles.
- Mail Server:
 - Goal: Send emails to users.

Use Case Diagram for Admin and User

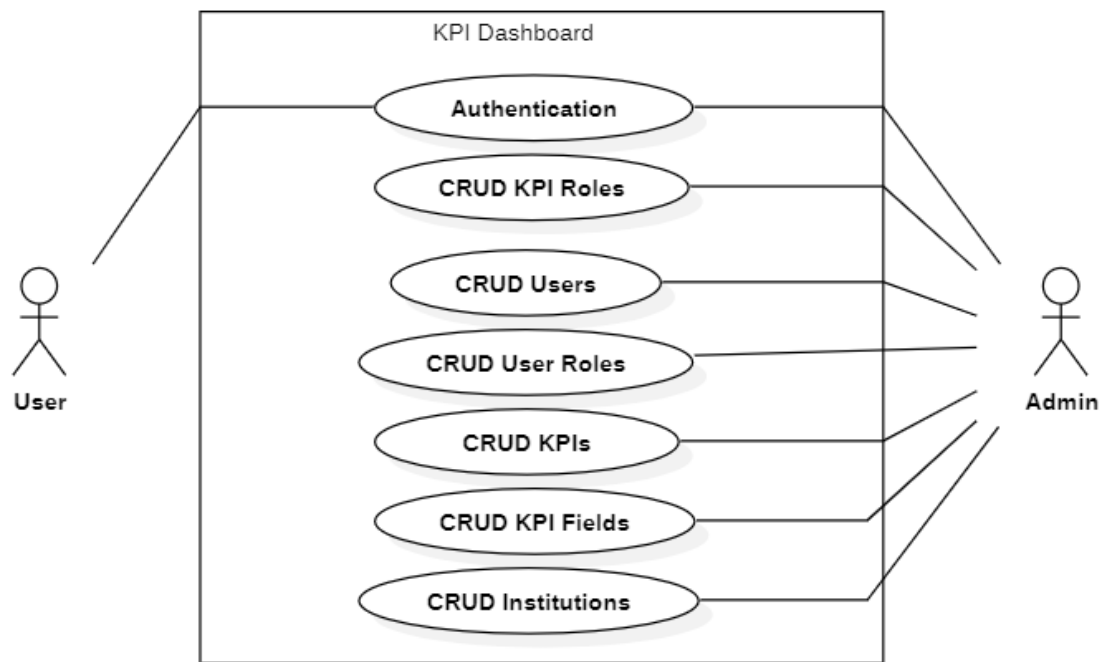


Figure 2.1: Use Case Diagram for Admin and User

Use Case Diagram for Authenticated User and Mail Server

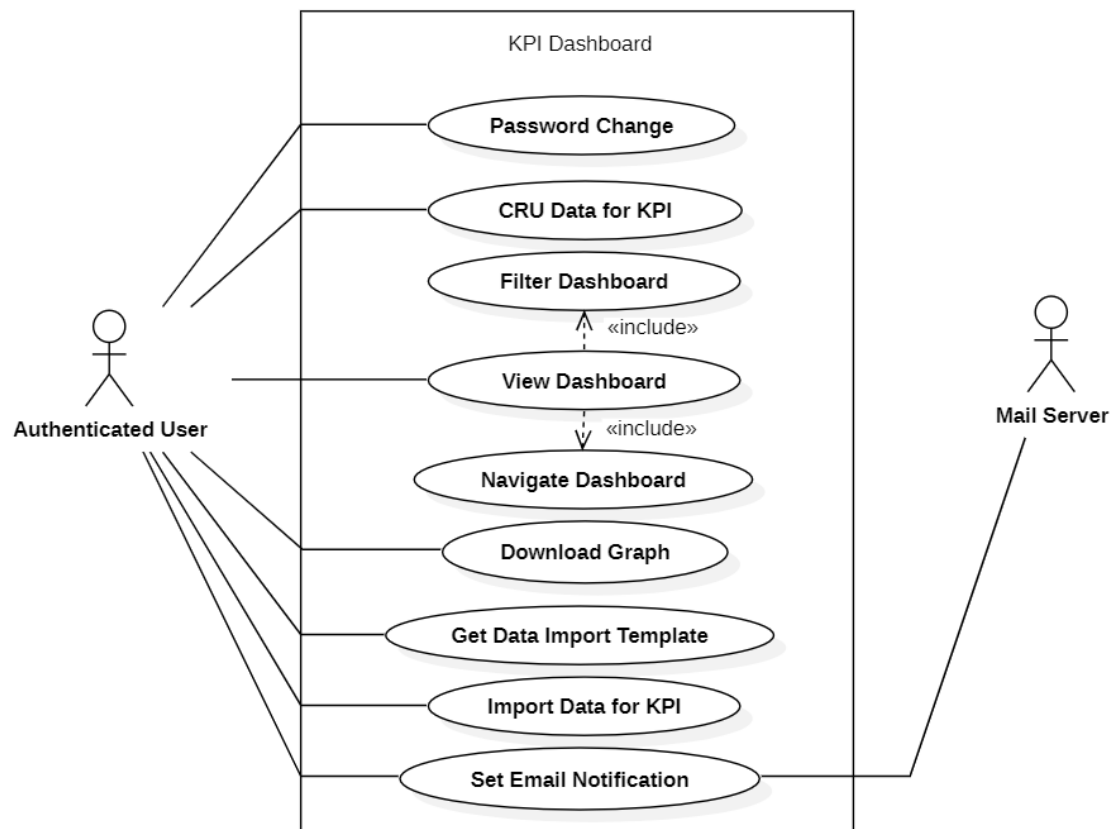


Figure 2.2: Use Case Diagram for Authenticated User and Mail Server

2.1.2 Use Case Description

The two most important cases will be written in the fully dressed and the remainder in the casual format.

Casual use cases

- **UC-1 CRUD Users:** The admin can create, read, update and delete users for the dashboard. He can also reset the password for a user if necessary.
- **UC-2 CRUD User Roles:** The admin can create, read, update and delete user roles. The user roles determine which KPI data the associated users are allowed to see and edit.
- **UC-3 CRUD KPI Roles:** The admin can create, read, update and delete KPI roles. The KPI roles determine which KPI the associated users are allowed to see.

- **UC-4 CRUD KPIs:** The admin can create, read, update and delete KPIs.
- **UC-5 CRUD Institutions:** The admin can create, read, update and delete institutions.
- **UC-6 CRUD KPI Fields:** The admin can create, read, update, and delete fields that are used in the formula of a KPI.
- **UC-7 Authentication:** The users and admin can log into their accounts with an email and password.
- **UC-8 Password Change:** The authenticated user can navigate to their profile settings where he can reset his password.
- **UC-9 Import Data for KPI:** The authenticated user can import data entries for existing KPIs.
- **UC-10 Get Data Import Template:** The authenticated user can download the template that can be used for importing data for a KPI.
- **UC-11 View Dashboard:** The authenticated user can view all KPIs they are authorized to track on the dashboard. Each KPI includes a trend indicator showing whether it has improved, remained steady, or declined, along with optional additional details.
- **UC-12 Filter Dashboard:** The authenticated user can filter the KPI data based on a specific time span.
- **UC-13 Set Email Notification:** The authenticated user can go to their profile settings to choose whether they want to receive email notifications and select a specific day of the month for the notifications to be sent.
- **UC-14 Download Graph:** The authenticated user can download the graph from a KPI.

Fully dressed use cases

ID	UC-15
Name	Navigate Dashboard
Actor	Authenticated User
Description	The authenticated user can navigate to the details of a KPI (go one tracking lower).
Pre-condition	<ul style="list-style-type: none">• The user is authenticated.• The user has permission to view KPI data for a lower tracking.• At least one KPI is defined.• Multiple KPI data entry are available.
Basic-Flow	<ul style="list-style-type: none">• The user clicks on a displayed KPI and is redirected to a lower tracking.
Alternative-Flow	The user cannot navigate to a lower tracking due to missing the necessary permissions.
Post-condition	The user still has the permissions to view KPI data on a lower tracking.
Result	The user can see the KPI data one tracking lower on the dashboard.

Table 2.1: Use Case: Navigate Dashboard

ID	UC-16
Name	CRU Data for KPI
Actor	Authenticated User
Description	The authenticated user can create, read and update data entries for existing KPIs.
Pre-condition	<ul style="list-style-type: none"> • The user is authenticated. • At least one KPI is defined.
Basic-Flow	<ul style="list-style-type: none"> • The user navigates to the 'Dashboard'. • They click on the graph at the data entry they wish to change. • The user completes the form. • They save their entry.
Alternative-Flow	Submitting new KPI data fails when not all required fields are filled out or if the user lacks the necessary permissions to add or update the data.
Post-condition	The user is navigated back to the dashboard.
Result	The user can see the newly added or updated KPI data on the dashboard.

Table 2.2: Use Case: CRU Data for KPI

Use Case Priority

The following list defines the priority of the previously defined use cases.

1. UC-11 View Dashboard
2. UC-15 Navigate Dashboard
3. UC-16 CRU Data for KPI
4. UC-12 Filter Dashboard
5. UC-7 Authentication
6. UC-1 CRUD Users
7. UC-2 CRUD User Roles
8. UC-3 CRUD KPI Roles
9. UC-4 CRUD KPI
10. UC-5 CRUD Institutions
11. UC-6 CRUD KPI Fields
12. UC-8 Password Change
13. UC-9 Import Data for KPI
14. UC-10 Get Data Import Template
15. UC-14 Download Graph
16. UC-13 Set Email Notification

2.2 Non-Functional Requirements

ID	NFR-1
Category	Performance
Requirement	The dashboard needs to load in less than 2 seconds.
Measures	The dashboard's average load time must be under 2 seconds.
Measuring technique	Chrome DevTools will be utilized to measure the website's load time.
Priority	High
Result	<p>Figure 2.3: Lighthouse Performance</p>

Table 2.3: Non-Functional Requirement: Dashboard Performance


ID	NFR-2
Category	Maintainability
Requirement	The API must be capable of being tested automatically.
Measures	Tests should cover 80% of the backend code.
Measuring technique	Tests will automatically run on every Git commit and will fail if the coverage target isn't met.
Priority	High
Result	<div style="text-align: center;"> <p>Coverage</p> <p>92.7%</p> <p>Required \geq 80.0%</p> <p>On 1.9k New Lines to cover.</p>  </div> <p style="text-align: center;">Figure 2.4: Backend Code Coverage</p>

Table 2.4: Non-Functional Requirement: API Maintainability

ID	NFR-4
Category	Maintainability
Requirement	Clean Code
Measures	Methods with identical functionality should not be defined more than once.
Measuring technique	Generating code metrics using SonarQube and verifying function reuse.
Priority	Medium
Result	<div style="text-align: center;"> <p><small>Last analysis: 3 days ago • 5.7k Lines of Code • Python, HTML, ...</small></p> <hr/> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">A 0 <small>Security</small></div> <div style="text-align: center;">A 0 <small>Reliability</small></div> <div style="text-align: center;">A 0 <small>Maintainability</small></div> <div style="text-align: center;">A 100% <small>Hotspots Reviewed</small></div> <div style="text-align: center;">92.7% <small>Coverage</small></div> <div style="text-align: center;">0.6% <small>Duplications</small></div> </div> <p>Figure 2.6: Backend Sonar</p> <hr/> <p><small>Last analysis: 3 days ago • 3.6k Lines of Code • TypeScript</small></p> <hr/> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">A 0 <small>Security</small></div> <div style="text-align: center;">A 0 <small>Reliability</small></div> <div style="text-align: center;">A 0 <small>Maintainability</small></div> <div style="text-align: center;">A — <small>Hotspots Reviewed</small></div> <div style="text-align: center;">5.6% <small>Coverage</small></div> <div style="text-align: center;">0.0% <small>Duplications</small></div> </div> <p>Figure 2.7: Frontend Sonar</p> </div>

Table 2.6: Non-Functional Requirement: Code Maintainability

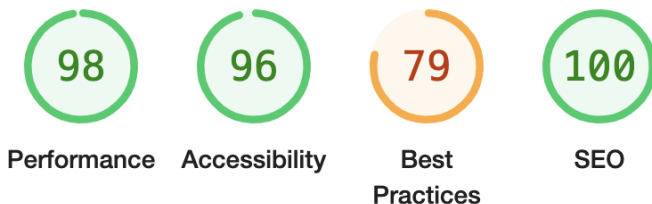
ID	NFR-3
Category	Accessibility
Requirement	The application should be accessible to a wide range of users.
Measures	The frontend must achieve a minimum audit score of 90 in Google Lighthouse.
Measuring technique	Google Lighthouse in Chrome DevTools will be utilized.
Priority	High
Result	<p>The minimum Google Lighthouse score was achieved in all sections except for Best Practices. This is due to the prototype running on HTTP instead of HTTPS.</p>  <p style="text-align: center;">Figure 2.5: Lighthouse Results</p>

Table 2.5: Non-Functional Requirement: Accessibility Compliance

2.2.1 Verification of Non-Functional Requirements

The non-functional requirements (NFRs) that can be verified automatically will be assessed throughout the entire implementation phase. The remaining NFRs will be evaluated during week 13.

Chapter 3

Domain Analysis

3.1 Domain Model

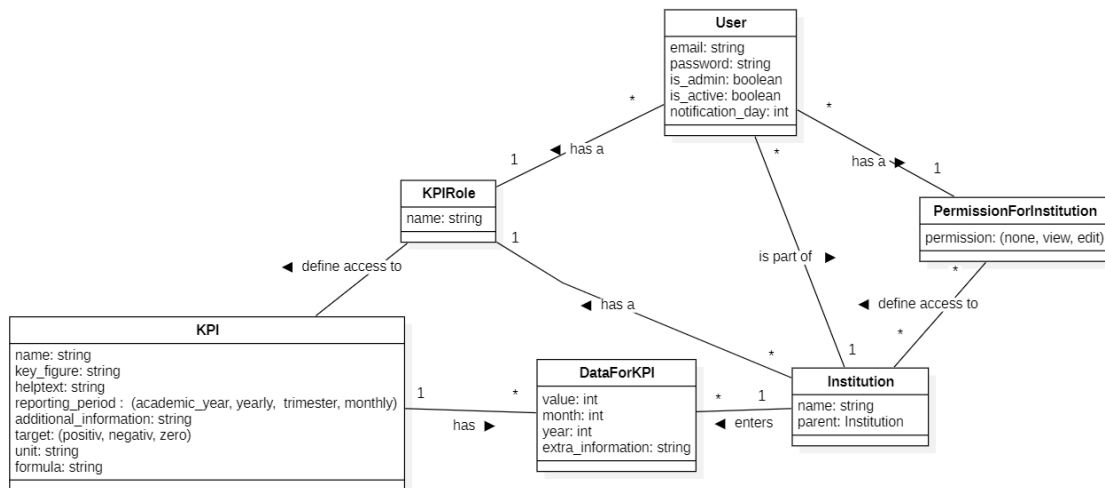


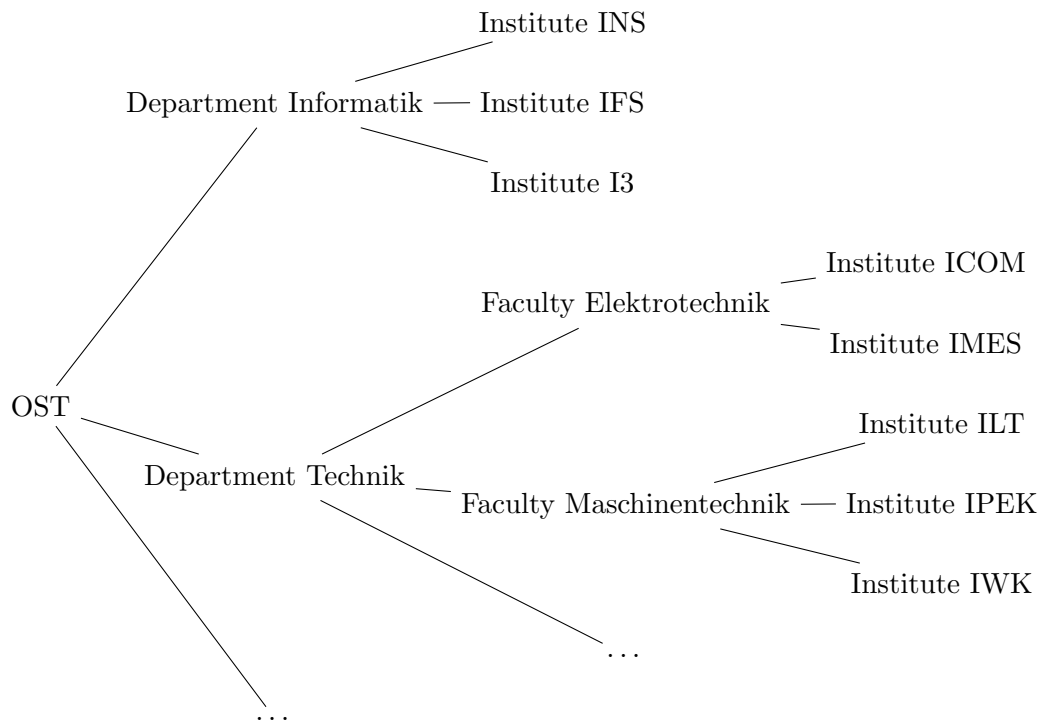
Figure 3.1: Domain Model

3.1.1 Explanations

This section provides some clarifications for the domain model.

Institution

The university's structure can be viewed as a tree, with four main hierarchical levels: OST, Department, Faculty, and Institute. To help visualize this concept, the following is a partial representation of the tree.



KPI

The available options for the field `reporting_period` are as follows:

- `academic_year`: KPIs are reported once a year in October, after the start of the academic year.
- `yearly`: KPIs are reported once a year in December.
- `trimester`: KPIs are reported every four months, in April, August, and December.
- `monthly`: KPIs are reported every month.

The options for the field target are as follows:

- positive: The KPI should aim to increase indefinitely, with higher values being ideal.
- negative: The KPI should aim to decrease indefinitely, with lower values being ideal.
- zero: The KPI should aim to approach zero.

KPIRole

KPIRoles determine access to KPIs and can be applied to both institutions and users.

Additional and Extra Information

Additional information is a static text that is used to display general information for a KPI. Extra information is a text that provides context or explanations for a given data entry.

Chapter 4

Architecture

The following chapter is based on the arc42 method.[7]

4.1 Stakeholders

These are the stakeholders for the project and their expectations:

Stakeholder	Expectations
Supervisor	The project meets academic standards.
Technical Advisor	The code meets technical industry standards.
Developer	Receive clear requirements.
Customer	The application meets all requirements.

Table 4.1: Stakeholders and Their Expectations

4.2 Scope and Context

The functionality of the application is defined in section 2.1 Functional Requirements. As this is a prototype, the automatic collection of data through APIs from surrounding systems is out of scope.

4.2.1 Interfaces

For deployment, SSH is utilized to deploy the application to the server from GitLab. The application itself provides one main interface: the backend offers a REST API for the frontend.

4.3 Solution Strategy

The following sections outline some of the core decisions. The architectural decisions regarding technology can be found in section 4.7 Architectural Decisions.

4.3.1 Web Application Architecture

The application is divided into two parts: the frontend and the backend. This separation allows each part to utilize its specific capabilities and provides the possibility to replace one part with a new version if needed. The backend is responsible for managing user accounts, handling database interactions, and providing an admin panel. The frontend delivers various dashboards with filtering options to display and compare data, while also managing all interactions for non-admin users.

4.3.2 User Roles

Since the university's structure is hierarchical, user roles determine whether a user can view data at different levels of the hierarchy. This approach reduces the configuration effort for administrators, as they do not need to create separate roles for each department. Instead, a single role, such as 'Head of Department', can be applied universally across departments. Alternatively, permissions could be structured so that each user role defines access to specific institutions. However, this would increase the configuration effort, as each department would require its own role configuration (e.g., 'Head of Department for CS' and 'Head of Department for economy').

4.3.3 Calculation of KPIs

Discussions with advisors and the customer revealed that the KPI formulas should be adaptable. It was decided that data should be entered for specific fields, such as earnings or expenses, and these fields would then be used in the KPI formulas. This approach offers the advantage that if a field is used in multiple KPIs, only one data entry is needed. Furthermore, previously entered data does not need to be re-entered for each new period.

4.3.4 Data Format for KPIs

Since the data consists of exports from existing systems and the purpose of the application is to display a dashboard in an efficient and user-friendly manner, all data is stored in a single table with fields rather than in separate tables for each data type (e.g., personal data, financial data).

4.4 Software Structure

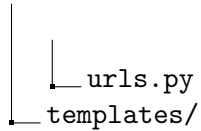
The following directory trees define the structure used in the application development process.

4.4.1 Backend Software Structure

The backend is organized into six main components:

- **models/**: Connects the database with Python, providing an ORM for efficient data access.
- **serializers/**: Prepares data for views and validates incoming data.
- **views/**: Manages incoming HTTP requests and permissions, delivering responses in a JSON format.
- **admin/**: Provides functionality for managing application data through the admin panel.
- **permissions/**: Defines custom permissions for controlling access to views.
- **static/**: Contains static files, including the user guide and custom CSS for styling the admin panel.

```
dashboard-kpi-backend/  
├── src/  
│   ├── manage.py  
│   ├── dashboard_kpi/  
│   │   ├── signals.py  
│   │   ├── backend.py  
│   │   ├── forms.py  
│   │   ├── helpers/  
│   │   ├── migrations/  
│   │   ├── models/  
│   │   ├── permissions/  
│   │   ├── serializers/  
│   │   ├── static/  
│   │   ├── tests/  
│   │   │   ├── helpers/  
│   │   │   ├── models/  
│   │   │   └── views/  
│   │   ├── views/  
│   │   └── admin/  
│   └── dashboard_kpi_backend/  
│       └── settings.py
```



- **manage.py**: The primary file for managing the Django server, including starting, creating and applying migrations.
- **settings.py**: Contains configuration settings such as database configurations, installed apps, and middleware.
- **urls.py**: Maps URL patterns to their corresponding views, defining the application's routing structure.
- **backend.py**: Defines a custom authentication backend that allows users to authenticate using their email addresses instead of usernames.
- **forms.py**: Defines custom forms for creating and updating user instances, including password validation and management.
- **signals.py**: Implements signal handlers to invalidate cached data.

4.4.2 Frontend Software Structure

The frontend is organized into five main parts:

- **public/**: Contains static files.
- **apis/**: Manages REST API calls for backend communication.
- **components/**: Includes reusable components.
- **pages/**: Contains the application's main pages.
- **views/**: Defines various dashboard views.

```
dashboard-kpi-frontend/  
├── patches/  
├── public/  
├── src/  
│   ├── apis/  
│   │   └── fetch/  
│   ├── components/  
│   │   ├── data-import-steps/  
│   │   │   └── timeline-items/  
│   │   ├── kpi-card/  
│   │   │   └── data-chart/  
│   │   ├── sidebar/  
│   │   └── user-forms/  
│   ├── helpers/  
│   │   └── DataMapper.ts  
│   ├── hooks/  
│   ├── pages/  
│   ├── types/  
│   ├── views/  
│   └── App.tsx  
├── tests/  
│   └── resources/  
└── index.html
```

- **DataMapper.ts**: Transforms backend DTOs into the required frontend objects.
- **App.tsx**: The main component that handles theme management, date display localization, and routing for pages and views.
- **index.html**: The root HTML page of the application.

4.5 Building Block View

4.5.1 Whitebox Overall System

The backend communicates with the database to verify credentials and return content to the user. Within the OST network, the frontend is accessible via HTTP on port 3000, and the backend on port 4434.

4.5.2 Building Blocks

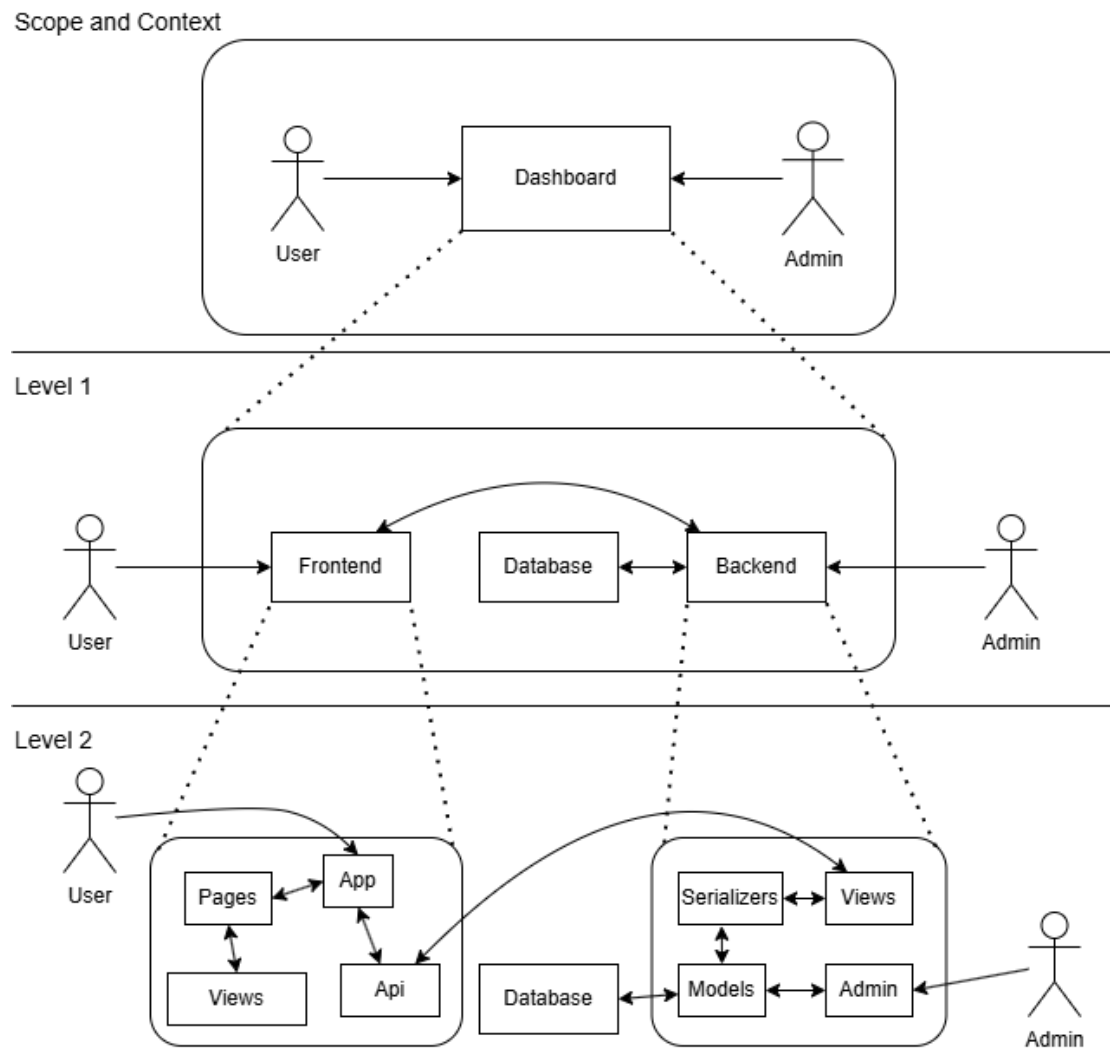


Figure 4.1: Building Blocks

4.5.3 Level 1

The user interacts with the frontend, which requests all necessary information from the backend server via an API. The backend server, in turn, has access to the database. The admin connects directly to the backend's admin panel.

Name	Responsibility
Frontend	Manages user interactions, authentication and user profiles.
Backend	Handles all application logic, responds to API calls and provides access to the admin panel.
Database	Stores all data required by the backend.

Table 4.2: Level 1 Components and Their Responsibilities

4.5.4 Level 2

Name	Responsibility
App	Manages routing and theming in the frontend.
Pages	The application's main pages.
Views	Individual views used within pages.
Api	Handles API requests to the backend.
URL	Manages routing in the backend.
Views	Handles requests and responses.
Serializers	Validates data.
Models	Manages data interactions with the database.
Admin	Manages the admin panel.

Table 4.3: Level 2 Components and Their Responsibilities

4.6 Deployment View

The production environment runs on an Ubuntu VM hosted within the OST infrastructure. On this VM, Docker containers are used to run the frontend, backend and database. The database is not accessible from outside the VM and is only reachable from the backend container. PostgreSQL is used as the database system. The production environment is structured as follows:

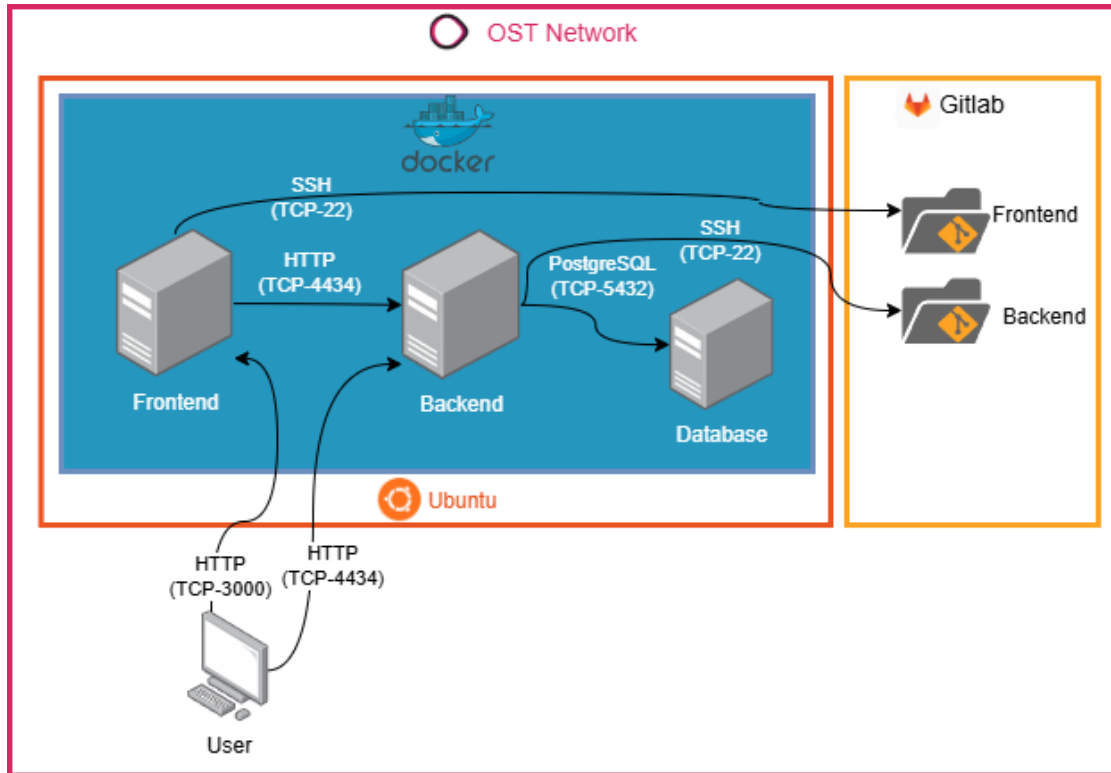


Figure 4.2: Deployment View

4.7 Architectural Decisions

4.7.1 Frontend

Development Stack

Vite, TypeScript and React were chosen for their speed, reliability and scalability. Vite enables fast development with native ES modules, TypeScript improves code quality through static typing and React's component-based architecture ensures reusability and efficient rendering.

UI Library

Mantine was selected for its comprehensive component library, which provides a wide range of customizable components that accelerate the development process. This library promotes consistency in design and simplifies the implementation of user interfaces. Additionally, Mantine offers built-in support for themes and responsive design. The extensive documentation allows for an efficient implementation.[5]

State Persistence

User and authentication data are stored in the browser's local storage, allowing users to stay logged in across multiple tabs without needing to log in again. Selected filters and other temporary states are stored in session storage, ensuring they persist even after a page reload. Since session storage is unique to each tab, it prevents any confusion or unintended filter changes when reloading different tabs.

4.7.2 Backend

Development Stack

Python was selected for its simplicity, versatility and strong community support in back-end development. Django complements this with its built-in ORM, REST framework for fast API development and an admin panel that reduces implementation time.[3]

Caching

To improve the performance and consistency of frequently used endpoints, caching was introduced. Primarily, these endpoints pertain to the structure of institutions and their associated permissions. To maintain full control over caching and its invalidation, a low-level cache API with model signals was chosen. This approach allows for selectively deleting the cache for a specific user when that user's data is updated or when there are changes in an institution or its permissions. The cache key was designed to include the name of the view, the user, and the user ID, as responses may vary for each user. The 'clear()' function was deliberately avoided for deleting cache keys, as its use can lead to unintended side effects.[4]

Evaluation of Formula

The ‘evaluate()’ function from the ‘numexpr’ library was chosen for evaluating the formula because it offers much of the functionality of ‘eval()’ while being safer. This safety stems from its design, as it only allows a restricted set of operators and functions specifically for numerical computations.[2]

Theme for Admin Panel

To enhance the user experience in the backend admin panel, a theme was used. Jazzmin was chosen due to its modern design, as well as its quick and easy setup process.

4.7.3 Database

PostgreSQL

PostgreSQL was chosen for its robustness, ACID compliance and support for complex data types, ensuring high performance, reliability and data integrity.

4.7.4 Other

Server

The OST server was selected because it resides within the OST VPN, which enhances security. Additionally, the production version is expected to be deployed on the same infrastructure.

Code Analysis Tool

SonarQube was selected over SonarCloud because the free version of SonarCloud does not support private projects. Since this project involves OST KPIs, making it public was not an option, leading to the choice of SonarQube.

4.8 Encountered Problems

Only issues with an impact of at least one day are mentioned here.

4.8.1 Issues with the Server

During development, two unexpected problems occurred with the server:

- A server restart initiated by the INS broke the SonarQube instance, causing the loss of custom configurations. The configurations were manually re-applied to restore functionality.
- The server storage was exhausted due to old Docker images not being deleted. After identifying the issue, the pipeline was updated to remove outdated images after deploying the latest one.

4.8.2 Incorrect Formulas

Some of the provided KPI formulas differed from those used in the current solution. Out of the six that could be verified, three were found to be different:

- **CEPY**: The provided formula divides by the current period, whereas the current solution divides by the absolute value of the previous period.
- **CRC**: The provided formula divides by the current period, whereas the current solution divides by the absolute value of the previous period.
- **AAR**: In the provided formula, the calculation uses the number of employees and the number of reported cases of illness. In the current solution, both values are represented in hours, and instead of the number of employees, the target time of employees is used.

CSPY could not be verified, but based on the discrepancies in the other formulas, it is also likely to involve division by the absolute value of the previous period.

Consequently, the formulas that could be verified were updated to match those in the current solution.

4.8.3 Change of Requirements

After the MVP demonstration, the end-user decided to move away from entering data monthly and instead opted to import the data in one go. As a result, the email reminder feature was removed and the remaining features were reevaluated and reprioritized.

4.8.4 Limitations of UI Library

The line chart component provided by the Mantine library was initially considered but had too many limitations for the required use cases. To address this, a custom line chart component was developed using Mantine's underlying chart library Recharts. This ensured the implementation of all necessary features while maintaining the same look and feel as the Mantine component.

Chapter 5

Quality Measures

5.1 Definition of Done

The following criteria establish benchmarks for determining when our product increments are deemed complete, ensuring a common understanding among the team of what ‘done’ entails[1]:

- The intended functionality is operational.
- All automated tests have successfully passed.
- The code has met all quality gates.
- The code adheres to all standard criteria.
- Documentation is current.
- The project plan is updated.
- Time tracking is accurate.

5.2 SonarQube

SonarQube is an open-source platform created by SonarSource for continuous code quality inspection. It offers a suite of static code analysis tools that identify bugs, vulnerabilities and code smells across various programming languages. SonarQube integrates effortlessly with popular version control systems such as GitHub and GitLab, providing automatic analysis and real-time feedback on code quality within the development workflow. This functionality enables teams to uphold high standards, recognize technical debt and improve software maintainability and reliability. Additionally, SonarQube produces detailed reports and metrics to track code quality trends and prioritize areas for improvement.[6]

We incorporate SonarQube into our development process for several key reasons. First, it aids in detecting and addressing code issues early in the development lifecycle, which minimizes the risk of introducing bugs and vulnerabilities into our software. SonarQube also provides actionable insights and recommendations, empowering developers to write cleaner and more maintainable code. Its seamless integration with our CI/CD pipeline allows us to automate code analysis and uphold quality standards throughout the workflow. Additionally, SonarQube offers comprehensive metrics, including lines of code, duplication, complexity, and code coverage, which help us assess the quality and maintainability of our codebase. By analyzing these metrics, we can identify areas for improvement, track code quality over time and optimize our development process.

5.2.1 Quality Gates

Different quality gates have been configured and applied for both the frontend and backend.

Frontend

- Duplicated Lines are less than 5%
- Maintainability Rating is A
- Reliability Rating is A
- Security Hotspots Reviewed is 100%
- Security Rating is A

Backend

- Coverage is more than 80%
- Duplicated Lines are less than 5%
- Maintainability Rating is A
- Reliability Rating is A
- Security Hotspots Reviewed is 100%
- Security Rating is A

5.3 CI/CD Pipeline

The GitLab CI/CD pipelines consist of the following stages:

- **lint**: Running the linter, performing a type check, and checking code formatting (frontend only).
- **build**: Building the application (frontend only).
- **test**: Running the linter (backend only) and executing tests.
- **sonar**: Performing a code quality analysis using SonarQube.
- **deploy**: Building a docker image and deploying it to the OST server.

The pipeline is triggered with each new commit to the main branch or for every merge request, excluding the deploy stage in the latter case. It is defined in the `.gitlab-ci.yml` file located in the project's root directory.

5.4 Test Concept

The test concept defines the scope, methodology, resources and timeline for testing activities. The test plan specifies the subjects, functionalities to be tested and assigned responsibilities. Together, these provide the foundation for organizing the testing process, allocating resources and executing the tests efficiently.

5.4.1 Testing Strategy

The testing strategy will include the following steps:

- Identification of test cases
- Execution of tests (both manual and automated)
- Reporting of bugs and issues
- Retesting of bugs / issues after they have been resolved

5.4.2 Test Environment

Server

- OS: Ubuntu 22.04.5 LTS
- Docker: 27.3.1
- Docker Compose: 2.29.7
- Postgres: Version 16

Client

- OS: macOS Sequoia Version 15.1.1, Windows 11 Version 10.0.22.631
- Browser: Chrome Version 131.0.6778.140

5.4.3 Test Deliverables

The test deliverables are:

- Test Artifacts

5.4.4 Test Schedule

Testing will be conducted before each milestone (MVP, Beta, Final) using the relevant test cases.

Rough Planing

- Week 9: Test MVP (DTC-1 – DTC-12, ATC-1 – ATC-9)
- Week 12: Test Beta (DTC-1 – DTC-20, ATC-1 – ATC-20)
- Week 14: Test Final Submission (all)

Detailed Planing

Frontend

The frontend application will mostly undergo manual testing. We have decided against extensive HTML/component tests due to the time it would require to implement them. The testing approach includes:

- UI/UX: Testing with Google Lighthouse
- Code style: ES-Lint checks within the pipeline
- Usability: Conducting usability tests with non-technical users

The different test cases can be found in the appendix.

Backend

Our backend will be tested through automated unit tests. These unit tests will be run alongside a code coverage checker to ensure we cover as much of our codebase as possible, with a target of achieving 80% code coverage.

The testing strategy includes:

- Functionality: Unit tests within the pipeline
- Code coverage: Sonar check within the pipeline
- Code style: Ruff checks within the pipeline

5.4.5 Test Roles

- Lead: Responsible for designing test cases, executing them and reporting results.
- Tester: Responsible for executing tests, reporting bugs and retesting issues.

5.4.6 Test Artefacts

The different test artefacts can be found in the appendix.

Chapter 6

Result

6.1 Functional Requirements

The following use cases have been successfully implemented. Detailed descriptions of each use case are provided in section 2.1.2 Use Case Description.

Available in the admin panel:

- UC-1 CRUD Users
- UC-2 CRUD User Roles
- UC-3 CRUD KPI Roles
- UC-4 CRUD KPIs
- UC-5 CRUD Institutions
- UC-6 CRUD KPI Fields

Available in the frontend:

- UC-7 Authentication
- UC-8 Password Change
- UC-9 Import Data for KPI
- UC-10 Get Data Import Template
- UC-11 View Dashboard
- UC-12 Filter Dashboard
- UC-14 Download Graph

- UC-15 Navigate Dashboard
- UC-16 CRU Data for KPI

The use case ‘UC-13 Set Email Notification‘ was not implemented due to time constraints and changes in prioritization. Further details are provided in the section 4.8.3 Change of Requirements.

6.1.1 Notable Use Cases

The results for the following use cases are described in more detail.

UC-2 CRUD User Roles

The matrix below defines the access permissions for user roles. This approach simplifies configuration for administrators and enhances clarity, as it eliminates the need to create separate roles for each individual user.

Definiere den Zugriff für Benutzerrollen

Benutzerrollen	OST	Department	Fachabteilung	Institut
Hochschulleitung	Edit ▾	Edit ▾	Edit ▾	Edit ▾
Departement	None ▾	View ▾	View ▾	View ▾
Fachabteilung	None ▾	None ▾	Edit ▾	Edit ▾
Institut	None ▾	None ▾	None ▾	Edit ▾

Legende:

- **Zeilen:** Benutzerrollen
- **Spalten:** Ebenen für Institutionen
- **Dropdown:** Zugriffstyp auswählen: None, View, oder Edit

[Speichern](#)

Figure 6.1: Matrix for user role access.

UC-4 CRUD KPIs

KPIs can be configured in the admin panel by completing the appropriate form. Below is an example of a filled-out form for the KPI 'Average Monthly Staff Turnover (AMST)':

Name *	<input type="text" value="Average Monthly Staff Turnover"/>
Kennzahl *	<input type="text" value="AMST"/>
Erläuterung *	<input type="text" value="Average Monthly Staff Turnover: Durchschnittliche Personenfluktuation pro Monat bisher"/>
Zusätzliche Informationen	<input type="text" value="Nicht berücksichtigt werden Personen mit einem Pensum von weniger als 20% sowie Lernende und Praktikanten."/>
Thytmus *	<input type="text" value="Monthly"/>
Ziel *	<input type="text" value="Zero"/>
Einheit *	<input type="text" value=""/> <small>Wird neben dem Ergebnis angezeigt. z.b. CHF oder %</small>
Formel *	<input type="text" value="(((Abgaenge) /(Mitarbeitende_Monatlich)) /months_in_year_so_far) *100"/> <small>Bitte beachten Sie das Benutzeranleitung für Erklärungen.</small>

Figure 6.2: An example of a KPI in the admin panel.

UC-11 View Dashboard

The configured KPIs will be automatically accessible in the frontend. Below is the same example, now displayed with data already populated.

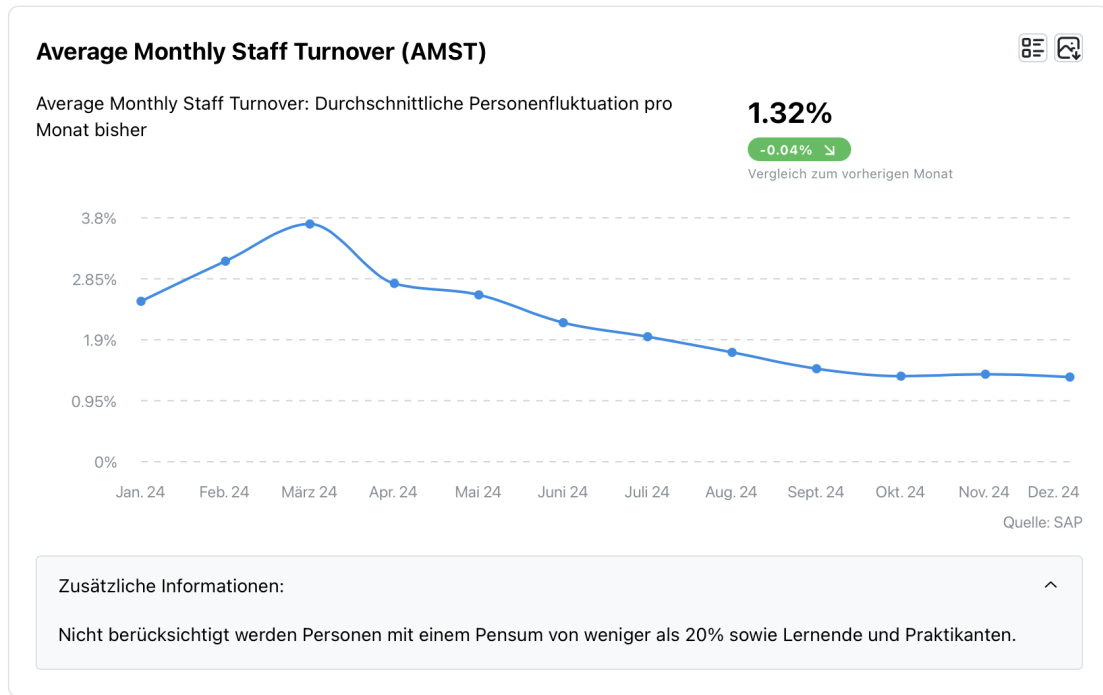


Figure 6.3: An example of a KPI in the frontend.

6.2 Non-Functional Requirements

The following NFRs were successfully implemented and met:

- **NFR-1:** The dashboard's average load time must be under 2 seconds.
- **NFR-2:** Tests should cover 80% of the backend code.
- **NFR-3:** The frontend must achieve a minimum audit score of 90 in Google Lighthouse.
- **NFR-4:** Methods with identical functionality should not be defined more than once.

Details of the results can be found in section 2.2 Non-Functional Requirements.

Chapter 7

Conclusion

7.1 Result Reflection

The goal of this project was to develop a prototype to replace the current Excel sheet used for calculating and managing key performance indicators (KPIs).

The primary challenges with the existing solution are its limited scalability and lack of user permissions. As the dataset grows, managing and maintaining the Excel sheet becomes increasingly difficult. Additionally, the absence of a permission system means users can either view all data or none, raising concerns about the confidentiality of sensitive information. Moreover, KPI graphs need to be manually adjusted, which becomes increasingly time-consuming as the number of institutions grows.

The prototype developed in this project addresses these issues. As a web service, it can be accessed via a browser, making data easier to share and update. A permission system has been implemented, enabling the definition of relevant KPIs for specific institutions and restricting user access where necessary. The prototype dynamically calculates all KPI formulas and visualizes them in interactive graphs.

Additionally, the prototype introduces several useful features. Users can add supplementary information to data entries, helping to explain anomalies such as sudden increases or drops in a graph. Graphs can be easily downloaded for use in reports or presentations. A date filtering feature has also been implemented, enabling users to select on specific time periods. This enables more precise analysis of trends and patterns, empowering users to make better-informed decisions based on recent or relevant historical data.

To facilitate a smooth transition, the prototype includes all existing institutions, along with their associated permissions and KPIs, as part of the initial setup. It also supports importing large datasets directly through CSV files.

7.2 Outlook

The data gathering process for the tool has room for improvement. Currently, users must manually collect data from surrounding systems, prepare it for import, and then complete the import process. Automating data retrieval directly from these systems would significantly reduce maintenance efforts and minimize errors caused by manual user input.

Another potential enhancement is enabling the direct referencing of other KPIs within formulas. At present, users must re-enter the entire formula of another KPI when referencing it, which adds unnecessary complexity. Allowing direct KPI references would simplify formula definitions and further improve the user experience.

As this is a prototype, the focus was primarily on achieving feature parity, resolving functional issues in the existing solution and enhancing user experience. Moving the application from HTTP to HTTPS would notably improve its overall security.

Although efforts were made to keep the frontend and backend clean and maintainable, further improvements are possible. For instance, implementing an interface that allows the frontend to retrieve types directly for the API would reduce the need for double definitions and prevent type inconsistencies when changes occur. This would streamline development and enhance overall system reliability.

Additional features or enhancements:

- Enable the login system to utilize the Switch edu-ID, as all end-users will be OST employees.
- Include additional details in graph downloads for better usability.
- Implement caching for KPI results, as older results are less likely to change.
- Add support for importing data from multiple institutions simultaneously.
- Add support for categorizing KPIs (e.g., by staff category or degree programs).
- Add functionality to generate reports directly from the dashboard.

7.3 Closing Statement

In conclusion, this project has successfully delivered a prototype that improves confidentiality, accessibility, and user experience. Future enhancements will continue to refine the tool, ensuring it remains adaptable and meets evolving needs.

Bibliography

- [1] Atlassian. What is the Definition of Done? <https://www.atlassian.com/agile/project-management/definition-of-done>, 2024. Accessed: 2024-10-04.
- [2] David M. Cooke, Francesc Alted, et al. NumExpr 2.0 User Guide. https://numexpr.readthedocs.io/en/latest/user_guide.html, 2024. Accessed: 2024-12-12.
- [3] Django Software Foundation. Django Documentation. <https://www.djangoproject.com>, 2024. Accessed: 2024-10-27.
- [4] The Python Software Foundation. The low-level cache API. <https://docs.djangoproject.com/en/5.1/topics/cache/#the-low-level-cache-api>, 2024. Accessed: 2024-12-12.
- [5] Vitaly Rtishchev and Contributors. Mantine Documentation. <https://mantine.dev>, 2024. Accessed: 2024-12-06.
- [6] SonarSource. SonarQube 10.6 Documentation. <https://docs.sonarsource.com/sonarqube/10.6/>, 2024. Accessed: 2024-10-01.
- [7] Dr. Gernot Starke. arc42 Documentation. <https://arc42.org/overview>, 2024. Accessed: 2024-12-14.

List of Figures

1.1	Web Application User Settings	3
1.2	Web Application Dashboard	4
1.3	Web Application KPI Comparison	5
1.4	Web Application Import Form	6
1.5	Web Application Import Report	6
1.6	Web Application Admin Panel	7
2.1	Use Case Diagram for Admin and User	10
2.2	Use Case Diagram for Authenticated User and Mail Server	11
2.3	Lighthouse Performance	16
2.4	Backend Code Coverage	17
2.6	Backend Sonar	17
2.7	Frontend Sonar	17
2.5	Lighthouse Results	18
3.1	Domain Model	19
4.1	Building Blocks	27
4.2	Deployment View	29
6.1	Matrix for user role access.	39
6.2	An example of a KPI in the admin panel.	40
6.3	An example of a KPI in the frontend.	41

List of Tables

2.1	Use Case: Navigate Dashboard	13
2.2	Use Case: CRU Data for KPI	14
2.3	Non-Functional Requirement: Dashboard Performance	16
2.4	Non-Functional Requirement: API Maintainability	17
2.6	Non-Functional Requirement: Code Maintainability	17
2.5	Non-Functional Requirement: Accessibility Compliance	18
4.1	Stakeholders and Their Expectations	22
4.2	Level 1 Components and Their Responsibilities	28
4.3	Level 2 Components and Their Responsibilities	28

Glossary

AAR Average Absenteeism Rate, an OST KPI.

CEPY Change of Earnings from previous year, an OST KPI.

CRC Change of Return per Capita, an OST KPI.

CSPY Change of Subsidies from previous Year, an OST KPI.

DTO Data Transfer Object, an object used to transfer data between processes, layers, or systems, typically without containing any business logic.

INS Institute for Network and Security, an OST institute.

KPI Key Performance Indicator, a measurable value that demonstrates how effectively an individual, team, or organization is achieving a specific objective.

ORM Object-Relational Mapping, a technique that maps database tables to objects, simplifying database interactions.

Tracking This term refers to the various institutional levels for which KPIs are tracked. The terminology is taken directly from the current solution.