Dash+

Development of supply chain management dashboard application (Occasse)

CSE2000 Software Project

Group 17D

Junwon Yoon
Kwangjin Lee
Kyongsu Kim
Michel Chen
Justin Jo

TA: Ana Marcu
TU Coach: Inald Lagendijk
Client: Occator

Preface

This report was written for the CSE2000, Software project course in Computer Science and Engineering at Delft University of Technology. The authors are a group of five computer science students at the same institute. The project was developing a dashboard application for *Occator*, a supply chain consultant company, and it was commenced on April 22nd 2024.

The purpose of this report is to document the development process of the dashboard application, outline the methodologies used, present the results of our testing, and evaluate the project's outcomes. It aims to provide a detailed account of our work and insights gained during the project.

The summary, introduction, and conclusion chapters are written with the purpose of ensuring that readers without any background in this project can understand it. Conversely, the middle chapters require knowledge of programming.

Readers particularly interested in the design and architecture of the application can find detailed information in Chapter 5. Those seeking information on the key features of the application can refer to Chapter 6.

We would like to express our appreciation to our client, *Occator*, for their guidance and support throughout this project. We also appreciate the assistance provided by the TU Delft coordinators, Teaching Assistants, and our peers who contributed valuable feedback. We hope this report provides a clear understanding of our project and the efforts invested in developing the dashboard application for *Occator*.

Delft, 21 June 2024

Junwon Yoon
Kwangjin Lee
Kyongsu Kim
Michel Chen
Justin Jo

[1]

[1]This text was written within the course Software Project CSE2000 of the faculty EEMCS of Delft University of Technology. Total word count: 10068 In writing this text, We: - Used generative AI (for inspiration) to structure the text. - Used generative AI to improve the grammar, style, and/or spelling of the text.

Summary

DASH+ is an extension of the OccaSee supply chain management application, designed to address the need for a comprehensive dashboard feature. The goal of this extension is to provide OccaSee users with a single view displaying key information critical for supply chain management, facilitating efficient analysis and decision-making.

The main issue with OccaSee is its lack of a feature that consolidates and displays essential data in a single view. To address this, we've integrated DASH+, a data visualization tool that allows users to customize and organize data according to their needs. DASH+ offers a variety of components, including bar charts and line graphs, which can be tailored to display data for specific domains and ranges selected by the user.

This report outlines the development process of DASH+, focusing on key decisions made to ensure the application meets the client's requirements effectively. The use of Scrum methodology guided the project, enabling clear goal identification, progress tracking, and iterative development.

The development process was structured into several phases, including requirement elicitation, design, implementation, and evaluation. Functional requirements were categorized based on the MoSCoW model, ensuring prioritization of critical features. Additionally, non-functional requirements were defined to ensure the application's reliability, compatibility, and usability.

Our final product should be an interactive dashboard application that meets the specified requirements. The main dashboard will contain different components, where components show different types of important data about the production plans. There should be 'Resource Occupancy', 'Effectiveness', 'Adherence' component, along with other KPI (Key point information) components, as those were the main requirements from the client. The components could be added, removed, or edited by the user. The components could be resized and moved around freely around the dashboard. Users can switch between different dashboard that the user have created. The dashboard is refreshed automatically every 10 minutes, or could be done manually by the user. The choice of 10 minutes for auto-refresh was due to the fact that refreshing whenever there's a change in the database will cause large overheads.

Throughout the development process, effective communication and collaboration among team members were prioritized, ensuring alignment with client expectations and project goals. By following established development practices, including thorough testing and adherence to coding standards, DASH+ was successfully implemented, meeting the client's needs and requirements.

Looking ahead, potential areas for future development include enhancing data visualization capabilities, integrating with additional data sources, and further emphasis on user testing to identify and address any usability issues early in the development process. Overall, DASH+ represents a valuable extension to the OccaSee application, providing users with a powerful tool for supply chain management analysis and decision-making.

Contents

1 Introduction

The world of commerce operates within the intricate and complex network of supply chain management. Supply chains act as the essential lifeline of goods that supports businesses worldwide, and gathering multifaceted information from them can be very valuable yet exhausting [1]. Being able to view all the key information of a program at a glance is very useful when analyzing. This is also true for the company Occator that we are working with: Occator is a supply chain consultant that provides overviews of schedules, inventory availability, orders, performances, and remarks to its clients using their own application “OccaSee”. The current problem with OccaSee is that it does not have a feature that displays the most important information in one view. In this project, the goal is to extend OccaSee to have such a feature by developing an interactive dashboard application tailored to the user.

This report aims to describe and elaborate on the decisions that were made during the development of the dashboard application. In addressing the problems, Scrum methodology [2] was used to manage projects, create sprints, and work towards a common goal throughout the development process. Furthermore, we also inspected on the feasibility and risks of these goals and made changes accordingly if necessary. The feasibility study and risk analysis could be found on appendix A.

This report is structured as follows. The problem that we aimed to solve will be introduced in detail in chapter 2. The outlined solution will be discussed in chapter 3, followed by the analyzed requirements in chapter 3. Chapter 4 will introduce the methodology to address the problem, and chapter 5 and 6 will describe the detailed design and development processes. The implementation of the tests will be considered in chapter 7, and the evaluation of this application will be reported in chapter 8. Finally, the conclusion of this whole process will be discussed in chapter 10.

2 Current Issues of Supply Chain Management Application

This chapter aims to explore the context and challenges specific to the DASH+ project. In Section 2.1, we provide a detailed overview of the DASH+ domain and the key stakeholders, highlighting their requirements and expectations. This sets the stage for understanding the project's goals and the chosen solution approach. Section 2.2 presents the specific problem statements, including an introduction to the application we are extending, the stakeholders involved, and real-life examples that inspired our approach. This comprehensive analysis establishes the groundwork for subsequent chapters to delve into specific requirements and implementation strategies.

2.1. Problem Statement and identification

In this section, a more detailed information about the problem domain and the context will be provided. Subsection 2.1.1 will give an introduction of the application that we are extending and developing. Next, subsection 2.1.2 will introduce stakeholders in the problem. Finally, subsection 2.1.3 will show the real-life examples that the team got inspirations from.

2.1.1. About OccaSee and DASH+

OccaSee is an application developed and provided by Occator, which helps businesses publish and gain information about their production plans [3]. This information includes production orders, inventory levels, product details, and other live data from the production. However, the current problem with OccaSee is that it does not have a feature that shows only the data that the user needs in one view. This lack of consolidated data display makes it difficult for users to quickly access and analyze the specific information they need, leading to inefficiencies and potential delays in decision-making. This introduced the need for DASH+, which is the application that we have integrated inside OccaSee. DASH+ is a data visualization tool that enables the users to customize the data and arrange them in such a way that fits their needs. It is a kind of a dashboard application which offers various components, where each component displays data in different formats such as but not limited to bar charts and line graphs. Components can be tailored to display data about a specific domain and range the user has chosen.

2.1.2. Stakeholders

Two key stakeholders are identified in this domain. The first is **Occator**, which will be responsible for deploying and maintaining the application service. The second is **Occator's clients**, the businesses that will actively use the application to manage their production plans and data. For Occator, it is important that they are aware of the application's architecture and the details of the implemented features. This is because they will be managing the application and also supplying to their clients, which means they need to have a thorough knowledge of the application. Moreover, this way, they can extend the application to meet further demands of their clients and maximize functionality and usability.

For the clients of Occator, the application must provide useful insights into their production while being easily usable and understandable. CEOs of the client companies want the dashboard to be used for business strategy and to provide a return on investment. Managers oversee the overall status of the production process and plan more efficient supply chain schedules. Workers, who use the application daily, need intuitive interfaces that facilitate quick and efficient data retrieval and usage. To identify what information would be more beneficial for the users, we have consulted with Occator to determine which data to use and how to handle them.

2.1.3. Use Cases of SCM

Supply Chain Management (SCM) tools are critical for modern enterprises to optimize and streamline their supply chain processes. These tools provide comprehensive information across various stages of the supply chain, including demand planning, supply network planning, production planning, detailed scheduling, and transportation management. They offer real-time visibility into inventory levels, order statuses, and shipment tracking, which is crucial for maintaining efficient operations and meeting customer demands. The tool that the we researched into specifically was SAP SCM. SAP SCM integrates data from different parts of the supply chain, facilitating collaboration between suppliers, manufacturers, and distributors. [4] This integration helps in reducing lead times, managing risks, and improving the overall agility of the supply chain. The tool's advanced analytics and reporting capabilities enable companies to forecast demand accurately, optimize resource allocation, and reduce costs by identifying inefficiencies and opportunities for improvement. The development of the dashboard for SAP SCM emphasizes integrating real-time data, fostering enhanced collaboration, and facilitating improved decision-making across the supply chain. The dashboard provides comprehensive visibility into supply chain metrics, predictive analytics, and optimization tools, enabling consultants to deliver actionable insights and strategic recommendations. By mirroring SAP SCM's robust functionalities, the dashboard helps drive efficiency, reduce costs, and enhance overall supply chain performance for clients.

2.2. Solution Outline of DASH+

This section outlines the comprehensive solution approach for developing DASH+ within OccaSee. It will delve into the specific goals of the project, detailing what we aim to achieve with the integration of DASH+. Additionally, the section will cover the chosen solution architecture, explaining how it will effectively meet the project's objectives, and will also discuss the technologies and frameworks to be utilized.

We aim to develop DASH+, an integrated data visualization tool within OccaSee, to address the current limitations of data presentation and customization. Drawing inspiration from SAP SCM's capabilities, we seek to provide OccaSee users with enhanced visibility and control over their production data. By integrating DASH+ into OccaSee, users will be able to customize their data views and gain actionable insights to optimize their production processes.

The frontend of DASH+ will be developed using React, a popular JavaScript library known for its component-based architecture and efficient rendering. This choice allows for dynamic and interactive user interfaces, essential for providing customizable data visualization components [5]. For the backend, we will employ Django, a high-level Python web framework renowned for its simplicity and scalability. This framework will facilitate rapid development and robust backend functionalities, ensuring seamless integration with OccaSee's existing infrastructure [6]. Since OccaSee was originally implemented using both React and Django, aligning DASH+ with these technologies ensures consistency and compatibility with the existing application.

In addition to frontend and backend technologies, we will set up a database structure similar to OccaSee's to ensure smooth integration and data consistency. To optimize frontend state management and ensure a smooth user experience, we will utilize Redux, a predictable state container for JavaScript apps. By centralizing application state and actions, Redux will enhance performance and maintainability, crucial for handling large datasets and complex interactions [5]. The communication of these resources will be handled with Docker. Subsequent chapters will delve into the specific requirements and detailed steps taken to implement DASH+ within OccaSee, including data integration, user interface design, and testing procedures.

3 Requirements for DASH+

Requirement analysis is crucial for gathering stakeholders' expectations for the final product and ensuring a unified understanding between the development team and the client[7]. This process ensures that our dashboard application aligns with Occator's requirements and business goals. This chapter starts with the section "Requirement Analysis and Prioritization," which focuses on an overview of our requirements. Section 3.1 explains the requirement development process and how we utilized the MoSCoW method[8]. Following that, in Section 3.2 and 3.3 sections, we will explain the functional requirements and non-functional requirements respectively. This chapter mainly focuses on the requirement list decided at the early stages of the project.

3.1. Requirement Analysis and Prioritization

To effectively manage the development process, we utilized the MoSCoW method, which categorizes requirements into four priority levels: Must have, Should have, Could have, and Won't have. This method aids in distributing tasks and focusing on the most critical functionalities[9].

Initially, the client provided a functional design. From this, we identified and rewrote the requirements in the MoSCoW style, ensuring a shared understanding among the team and aligning priorities based on the client's needs. Below is a comprehensive overview of the must and should have functional requirements. The could, won't have and non-functional requirements will not be presented below, as they won't be shown directly in the application. Full requirement list including these will be included in appendix B. The list of the requirements is as following:

Must have

1. The application has a dashboard.
2. Dashboards are user-specific (saved per user).
3. The application has components that show different data about the plan.
4. A component has a title and a description.
5. Each component must be able to show data in different time buckets.
6. The size of time buckets can be selected by the user: 1h, 8h, 12h, 1d, 3d, 7d.
7. The component must have a filter to filter plans per product, resource, or custom-made selection.
8. Users can see a dropdown list of available components.
9. Users can add new components to the dashboard from a dropdown list of available components.
10. Users can delete a component from the dashboard.
11. The dashboard will automatically save all the user's dashboard changes
12. The dashboard fetches new data and renders it every 10 minutes.

13. Users must be able to edit the layout of the dashboard.
14. There must be a component showing resource occupancy in a selected time bucket with percentage values for: productive, non-productive (maintenance and downtime), setup, changeover, idle
15. There must be a component showing information about the effectiveness of the plan.
16. The components are movable and resizable

Should have

1. Users can create a new board and switch between boards using a dropdown list.
2. Users can name the dashboard
3. There should be a component calculating KPI information following a predefined formula.
4. The application must make the results exportable to PDF.
5. Users can switch between plans.
6. A group of users can share boards, utilizing the group functionality already supported in the OccaSee application.
7. Admins can see the list of components that are available to the user.
8. Admins can define new components and make them available to the user, including title, description, axis, color, size, etc.
9. Admins can delete an existing component.
10. Admins can edit components (title, description, axis, color, size, etc.). If the component is being used by a user, a warning should be given (do you want to make a copy instead?).
11. The application should support the admin to modify and configure the formula that calculates the KPI information.
12. There should be a component showing information for adherence

3.2. Functional Requirements

The functional requirements section provides detailed information about the important functionalities of the dashboard application. These requirements ensure that the application meets user needs and performs essential tasks effectively.

Dashboard

As a must have feature, the application will feature a primary dashboard accessible by the user (Must 2), where they can view performance metrics such as graphs, numeric values, and diagrams related to plan information (Must 3). Users can add new components from a dropdown list of available components (Must 8). The components are resizable and movable (Must 16). Additionally, users can delete, edit components from the dashboard (Must 10, 13), providing flexibility in managing the dashboard's content.

The ability to create new dashboards, switch between them, and name each dashboard are classified as should have functionalities (Should 1, 2). These features allow users to organize different sets of data for varied insights and easily identify and manage multiple boards. Sharing dashboards among group members is also a should-have feature (Should 6), which will utilize the group functionality already supported in the OccaSee application.

Data Management

To maintain a seamless user experience, the layout of the dashboard will be saved automatically whenever changes are made, a critical must have requirement (Must 11). The dashboard must fetch and render new data every 10 minutes (Must 12) or when the user refreshes the page, ensuring that users always have access to the most current information.

This data management part changed several times based on our application's ability. Initially, an automatic refresh option that responded to database changes was considered. However, due to the large size of our data and long loading times, as well as the fact that other OccaSee functionalities do not use automatic refresh, we decided to follow the existing OccaSee approach of refreshing every 10 minutes. Additionally, we initially considered having a save button, but to prevent users from losing their dashboard settings if they forgot to save, we decided to implement an automatic saving feature. All changes were made after thorough discussions and meetings with the client.

Component

Components are a very important feature in our application. The explanation of each component will be introduced in the section 6. They refer to one box in the dashboard and contain a title, description, and size that will be shown on the board (Must 4, 16). The user will not input the size in numbers; instead, when the user resizes the box, it will automatically update the size.

Users, as a must have, are able to enter more information about the components depending on what component it is, such as time bucket, end time, and resource list (Must 5, 6, 7).

- **Time bucket** is a predefined time interval for which the data is aggregated and displayed. It helps in breaking down the data into manageable chunks, making it easier to analyze performance over specific periods.
- **Start time** refers to the specific point in time when the data collection or the analysis starts. This is essential for setting boundaries for the data visualization, ensuring that users are looking at the correct timeframe.
- **Resource list** is a selection of user-defined resources that the component will track or display information about. Users can select resources relevant to their analysis, ensuring that the data shown is pertinent to their needs.

Each component's flexibility and the ability to customize these details enhance the application's usefulness, allowing users to tailor the dashboard to their specific requirements and providing a more detailed and accurate representation of the data. Detailed explanation of each component will be done at Types of components.

3.3. Non-functional Requirements

Non-functional requirements specify the system's operational capabilities and constraints, ensuring that our dashboard application is robust, user-friendly, and compatible with various environments and platforms. Below are the key non-functional requirements for the application, explained briefly.

- **Development Tools** The development tools were chosen based on the original OccaSee application's languages to ensure consistency and maintainability. React and Redux will be used for frontend development, providing a dynamic and efficient user interface. Django will be employed for backend development, offering rapid development and robust performance.
- **Project Organization** The project files will be organized similarly to OccaSee, enhancing productivity and reducing the learning curve by providing a familiar structure.
- **Database Management** A mocked database will be used for development and testing to maintain data integrity and allow for robust testing scenarios without affecting live data.
- **Compatibility and Adaptability** The application will be compatible with the latest versions of Windows and Mac OS and adaptable to various screen sizes, ensuring a versatile user experience across different devices.

By addressing these non-functional requirements, we ensure that the application is built on a solid foundation, offering reliability, compatibility, and ease of use. This approach helps in delivering a high-quality product that meets user expectations and industry standards.

4 Development Methodology

In this chapter, the general methodology used in this project to obtain the final product will be discussed. In section 4.1, the definition of 'done' within the team will be described. The following section, 4.2, will explain the SCRUM methodology that has been used throughout the project. Finally, in section 4.3 the communication methods among the team will be illustrated.

4.1. Definition of Done

To ensure consistency and quality throughout the development process, we have established criteria for the completion of tasks and merge requests. Each task will be considered done when it meets the predefined acceptance criteria, passes code review, and has been successfully tested. The following criteria have been set:

- It has been correctly implemented based on the requirements
- It was reviewed by at least 1 team member of each side(backend & frontend)
- Before marking as done, all tasks should be tested with at least 90% of branch coverage.
- The task has a sufficient amount of comments

4.2. SCRUM Method

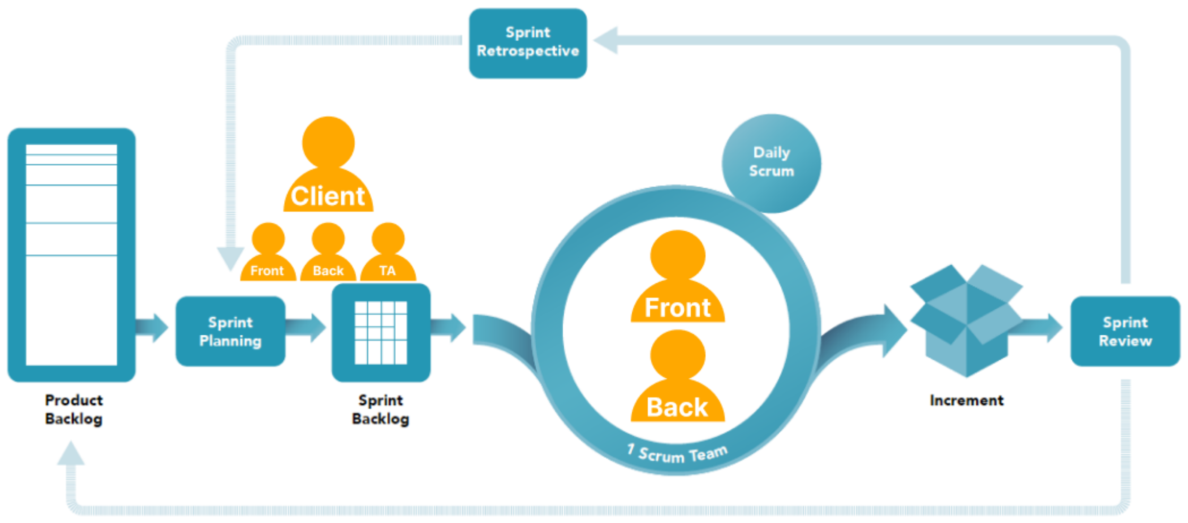


Figure 4.1: Scrum cycle

Figure 4.1 illustrates the fundamental cycle we adhered to throughout the project.

- Define the weekly sprint in collaboration with clients, TA, and team members.
- Establish the daily schedule.
- Conduct a review at the end of each day.
- Update the backlog accordingly.

The method that was used throughout the development is the Scrum method, which is a kind of Agile development methodology. Scrum, an agile software development framework, emphasizes adaptability to changing requirements, known as “requirements volatility”. Unlike traditional forecasting methods, Scrum allows for possible changes in customer requirements during development. This approach recognizes the uncertainty of pre-defining the problem thus it prioritizes responses to evolving needs, technological developments, and market developments [2].

4.3. Team Communication

Effective communication is crucial for the success of the project. We have established a structured approach to ensure clear and efficient communication among members, the client, and the teaching assistant (TA). The subsection will contain 4.3.1 the description of GitLab employment will be delivered. The following section 4.3.2 will illustrate the code of conduct.

4.3.1. Git

During the development process, GitLab was selected as the main management tool for the sprints and the communication tool for code review. It is the version control tool that enables effective collaboration and tracking of progress. By assigning issues to each member, setting clear milestones, weight, and time tracking, the issue will instruct weekly goal. Every git issue has been closed with the following rules:

- Each team member initially works on their own branch and merges later.
- At least two team members must approve the merge request.
- All merge requests should be done in the ‘dev’ branch first.
- If all members approve, merge the ‘dev’ branch into the ‘main’ branch.
- The frontend and backend parts should use the ‘frontend’ and ‘backend’ branches, respectively.
- The ‘frontend’ branch should be merged with at least one approval.
- The ‘backend’ branch should be merged with at least one approval.

To elaborate further, in Scrum methodology, each issue can employ child tasks to decompose a larger goal into more manageable sub-tasks. As each sub-task is completed, it provides an overview of the current progress. Moreover, labels can be applied to categorize issues, while weights can be assigned to prioritize them. These functionalities support backlog management and sprint planning by aiding in task organization and prioritization, thus facilitating the effective execution of the Scrum framework. Finally, creating boards for each functional requirement helps to enhance our top priorities.

4.3.2. Code of Conduct

To ensure an effective and efficient team working process, we have established a code of conduct that prioritizes shared values and norms. These include:

- **Fairness, Respect, Transparency, and Clear Communication**: Guiding principles for all interactions.
- **Commitment to Quality**: We aim for a minimum grade of 8.0 and commits to delivering a high-quality dashboard application for Occator.
- **Rotation of Roles**: Roles such as chairperson and note-taker rotate weekly to ensure equal participation.
- **Conflict Resolution**: Emphasis on open communication, mutual respect, and conflict prevention and support.

Success factors include balanced team composition, strong communication, active participation, shared vision, and diversity of backgrounds and experiences. Evaluation norms cover task completion, respectful behavior, adherence to requirements, clear documentation, teamwork, time management, and communication of significant changes. The detailed code of conduct is shared among group members and is accessible to all.

5 Process

This chapter will describe how we went about designing DASH+. The design of the application is very important for the group to develop in a proper way. It helps the group by aligning everyone's ideas of the project, and it also provides us with a structure to systematically solve problems and work towards the end goal. The design not only ensures that they meet the requirements, but also ensures that it resonates with the end user on a meaningful level. Section 5.1 will describe the system architecture design and the thought process behind it. The following section, 5.2, will contain the chosen frameworks. Lastly, section 5.3 explains the user interdace.

5.1. System Architecture Design

When designing the system architecture, it is important to keep in mind the structure that OccaSee already has and extend from it whenever possible. Figure 5.1 shows the system architecture we designed based on OccaSee's structure.

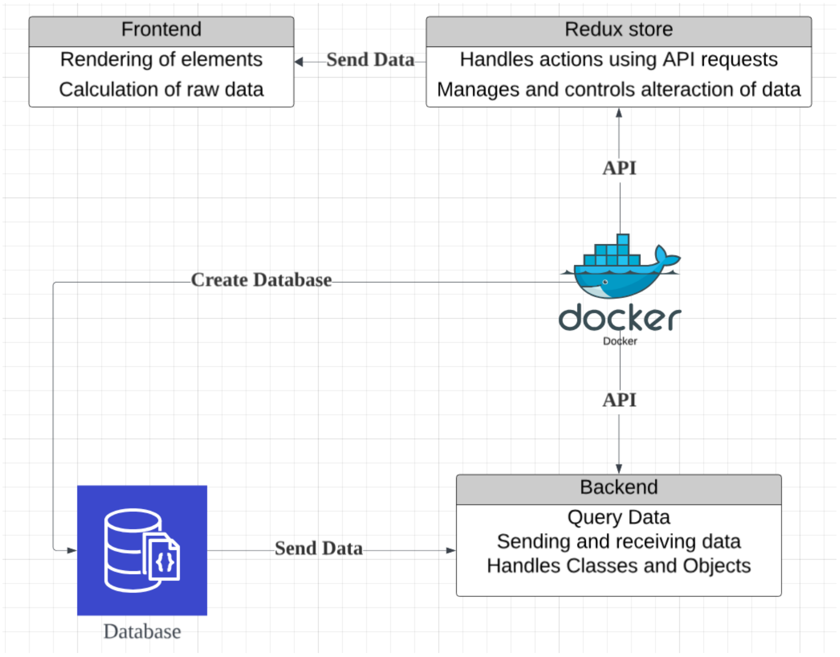


Figure 5.1: Diagram of the system architecture

The frontend part of DASH+ is what the user interacts with. Its role is to manipulate the data fetched from the database to calculate and display meaningful statistics. Also, it listens for user inputs, which triggers event dispatches to the Redux store. Redux store is reponsible for updating the application state based on the actions given in the frontend and the data received from the backend. The frontend is refreshed every 10 minutes automatically. We have decided to implement in this manner, as refreshing every single time there's a change in the database will cause a lot of overhead, and we wanted the clients to view the most recent data, no older than 10 minutes.

The backend part of DASH+ is where the actual data is stored and managed. Our database started with the database structure Occasee has provided us. This included important data that we needed for the components. For example, the resource occupancy component needs the changeover time. Our goal is to extend this database if needed for our application. The first table that we added is a dashboard. It has its id, title and the user it is connected to. In this way, one user could have multiple dashboards.

Then we added tables for each component type. The most important fields for the components are id and titles. Ids are the primary key of components, which means they are unique among all other components. Title is the text that is displayed on the component, and they could be the same. Other variables that we need for the visualization of a component are its x and y values for their positions and sizes, because each component should have the same location and size whenever the user refreshes and fetches. Time buckets were also important as the data must be shown on a chosen timeline. There are other fields that are specific to some components, for instance 'Component Resource Occupancy' contains a list or resources. The list of resources are only references, so that the frontend can fetch the newest data from Occasee. This method was chosen over saving the actual data, because we decided that it would not be optimal to do so as if the frontend gets the component from the backend, it might be outdated and it will not be possible to query to the actual data. This also helps mitigating redundancy in the database. This method was applied similarly for all the list types of data.

The connection between the frontend and backend is handled by Docker. The Docker setup contains multiple containers: one each for React frontend, Django backend, and the database. A more detailed explanation on this architecture is mentioned in section 5.2.

5.2. Chosen Frameworks

For each part identified in system architecture, different frameworks were used to implement their required features. In this section, each framework will be explained in more details to elaborate how these features were implemented. In paragraph Django, the architecture of the backend part of DASH+ will be explained. The following paragraph Redux will explain how the Redux store works and the benefits of having it. Lastly, in paragraph Docker, we will explain how the data communication is achieved between frontend and backend side of DASH+.

Django

In the backend, leveraging Django, a Python-based framework, we implemented various functionalities related to components within the dashboard class. Given that OccaSee, the existing platform we were building upon, was already implemented using Django, it made sense to leverage the same technology stack to ensure compatibility and seamless integration. One advantage of using Django is its robust built-in ORM (Object-Relational Mapping) system. This allows us to define complex data models in Python classes, abstracting away the underlying SQL queries and database operations. The provided code includes models for different types of components, each representing a specific aspect of data visualization on the dashboard, such as resource occupancy component and effectiveness component. Each component model contains fields for attributes such as title, location coordinates, size, time frame, and relevant data sources. By utilizing Django's ORM, we streamline the development process and ensure efficient management of these diverse data elements within the dashboard, while maintaining clear, structured models that integrate seamlessly into our existing Django-based platform.

Redux

Initially, the foundation of the frontend application was implemented using basic functionalities from React. Although DASH+ could be built by only using React, we decided to integrate Redux to our frontend application, in order to gain advantages in maintaining application state in a more concise manner.

Redux provided a *single source of truth* for the application's state. Many data about the dashboard and its components are saved as states, and Redux store manages these states and provides functions that allows manipulation of these data. This facilitates simplified data flow, easier tracking of state changes in debugging since the state is consistent and changes are predictable. This fixed numerous errors derived from passing too many props to various React components in the main file. By eliminating the need to pass props down multiple component levels using Redux, frontend side of DASH+ had improved state access and modification from any part of the application.

Moreover, before using Redux, most of the functionalities were implemented in the main file of the frontend side, resulting in an overly lengthy and complex code. Centralizing the state with Redux allowed us to write more readable and maintainable code. For instance, previously, fetching data in multiple components led to repeated code. Redux streamlined this by managing state through actions, reducers, and a store, and by calling actions within React components. This clear separation of concerns enhanced the maintainability of our codebase, making it easier to understand, scale, and modify.

Docker

Docker is an open-source platform designed to automate the deployment, scaling, and management of applications using containers. Containers are self-sufficient environments that include everything needed to run a piece of software, such as the code, runtime, system tools, libraries, and settings. By using Docker, it enables us to package applications and their dependencies into containers, ensuring consistency across multiple development, testing, and production environments. For example, as shown in figure 5.1, deploying the backend, frontend, and database as separate containers in each environment allows them to communicate seamlessly through API Endpoints.

5.3. User Interface

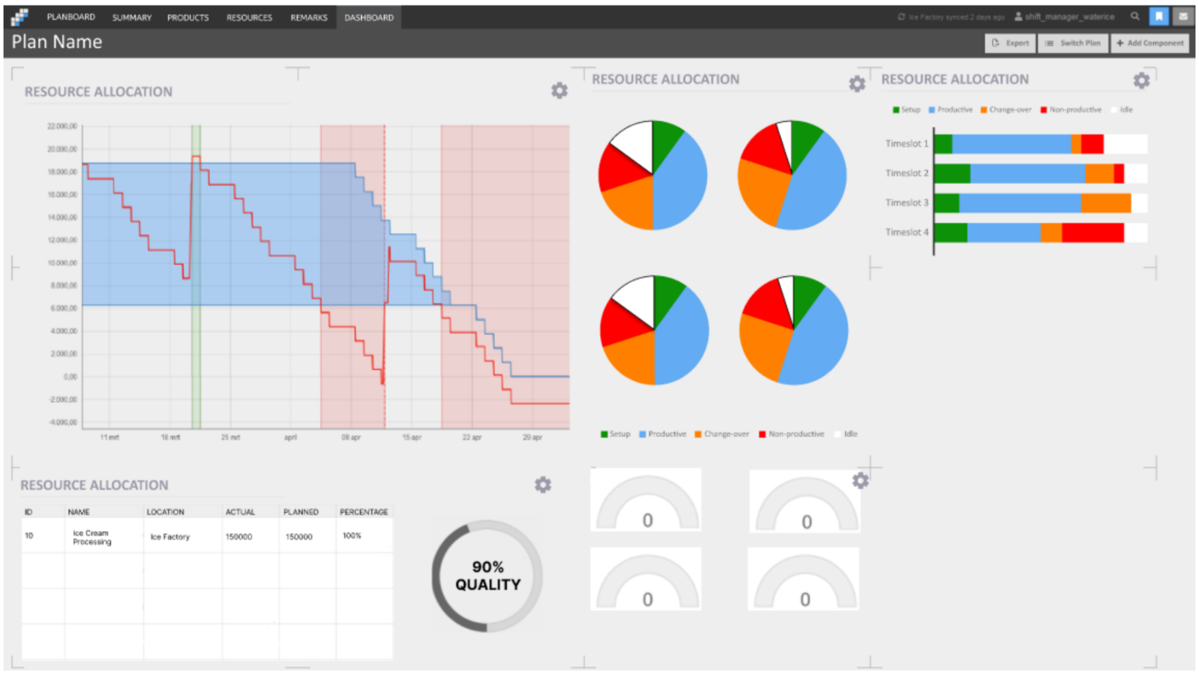
The user interface (UI) is a crucial component of any application, ensuring clear and error-free interaction with users. To create an interface that meets the client's needs, we developed several prototypes, refining and perfecting our design through an iterative process.

Early Design Phase

In the early phase of the design, we created a low-fidelity prototype by sketching ideas and discussing them with the client. This included scenarios and drawings of the application's functionalities, illustrating how our team envisioned implementing each required feature. These initial sketches provided a general idea of the application's overall look and feel, helping us gain a deeper insight into the client's design preferences and requirements.

Medium-Fidelity Prototype

Based on the initial understanding, the team progressed to creating a medium-fidelity prototype. This version more accurately depicted how the application would look at the end of the project. It was developed using the client's feedback and a prioritized list of requirements. The medium-fidelity prototype included some interactive functionalities, allowing the client to get a tangible sense of how the application would function and appear once completed. This iterative process ensured that the final design would align closely with the client's expectations and needs.



ID	NAME	LOCATION	ACTUAL	PLANNED	PERCENTAGE
10	Ice Cream Processing	Ice Factory	150000	150000	100%

Figure 5.2: Medium-fidelity prototype of DASH+

The medium-fidelity prototype (Appendix C) included some interactive functionalities, allowing the client to get a tangible sense of how the application would function and appear once completed. This iterative process ensured that the final design would align closely with the client's expectations and needs. In designing the DASH+ interface, we adhered to several key design principles (Appendix D) to ensure usability and effectiveness.

User Evaluation and Iteration

The user interface of DASH+ was refined through extensive user testing. These tests allowed us to evaluate the effectiveness and usability of our interface, receiving valuable feedback to drive improvements. A more detailed overview of this process is illustrated in chapter 8.

6
Key Features of DASH+

This chapter provides a comprehensive overview of the key features of the application, detailing their implementation and highlighting their significance for DASH+. Each feature is crucial to the application's functionality and user experience, and this chapter aims to introduce why these features are essential.

We will explore the dashboard, which serves as the central hub of the application, allowing users to interact with and manage various aspects of their data. The dashboard's components, such as data visualization, and user interface elements, will be introduced and examined in detail.

API Endpoints

To effectively display the information users need, the server must accurately transmit the required data. In the server running in backend Django project, we create models, apply views to these models for external presentation, and connect them to URLs to complete the connections. These URLs can then be accessed by the front end through HTTP requests, facilitating the transmission of the requested data.

Since displaying all information from the requested data is both inefficient and conflicting with the program's purpose, implementing a filtering function is crucial. This is done in the frontend part of the application, where only the data chosen by the user is processed, calculated, and displayed. This feature allows users to view only the necessary information on their dashboard. This method prevents unnecessary traffic by avoiding the display of the entire dataset, thereby enhancing the application's performance.

Dashboard

The dashboard serves as the central hub of the application, where all components are displayed and managed. Each dashboard is saved in the database along with its list of components. Any changes made to the dashboard are automatically saved to ensure that user configurations are preserved. To prevent users from viewing outdated data, the dashboard refreshes every 10 minutes, fetching the latest information.

To avoid overwhelming users with excessive data and to address the varying data needs of different users and situations, the application allows users to create multiple dashboards and switch between them as needed. This feature provides flexibility and customization, enabling users to view only the most relevant information for their specific purposes. Additionally, users can rename these dashboards to better organize and manage their data views.

Components

Adding components

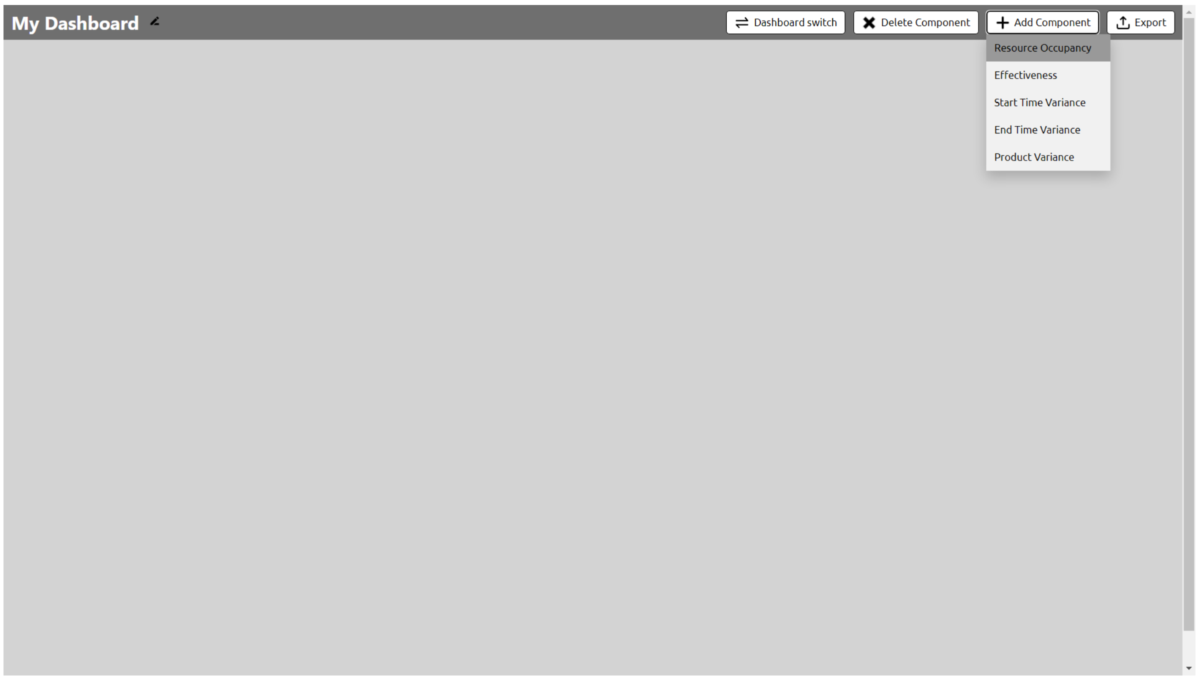


Figure 6.1: Adding component from the dropdown menu

As the figure 6.1 shows, components can be added to the dashboard by clicking on the dropdown menu located in the top right corner. This menu contains types of the components that can be added to the dashboard.

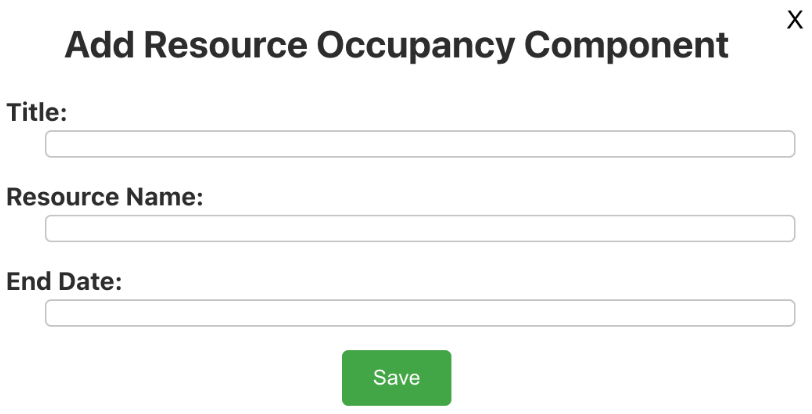


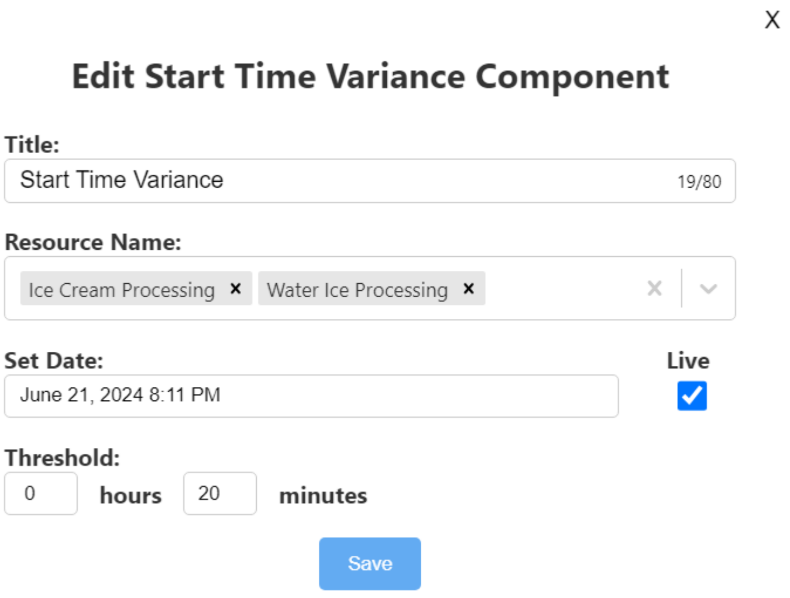
Figure 6.2: Example form for adding components

Upon selecting an option, a form will appear, allowing users to specify details such as the title of the component, the resources from which to display data, and the time frame for the data. Each component will have different fields in this form to cater to its unique requirements. For example, some components will include a threshold field to set specific data limits. This process ensures that users can customize each component to scope specific data, enhancing the relevance and utility of the dashboard.

Moving and resizing components

Components on the dashboard can be moved and resized freely, offering a customizable layout. The sizes and locations of components are saved in the database, ensuring that users do not have to reset the size and location of components each time, preserving their preferred layout as desired. Additionally, these values are saved as ratios relative to the window size, which ensures that the dashboard remains responsive and appropriately scaled.

Editing components



X

Edit Start Time Variance Component

Title: Start Time Variance 19/80

Resource Name: Ice Cream Processing × Water Ice Processing ×

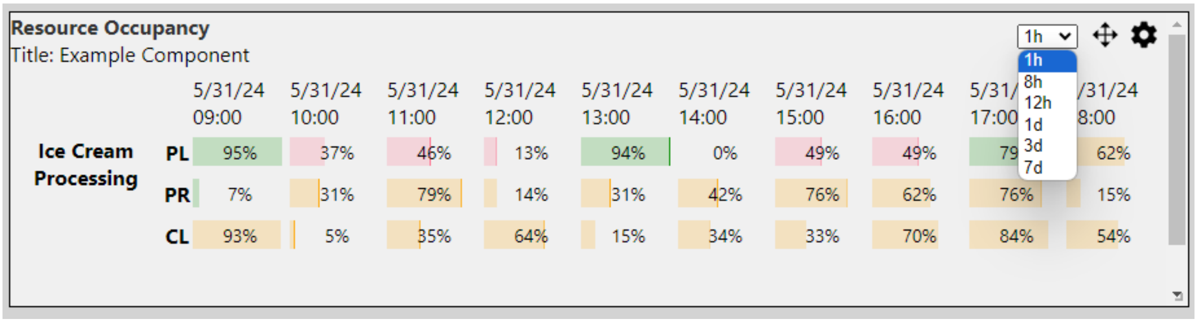
Set Date: June 21, 2024 8:11 PM

Live ✓

Threshold: 0 hours 20 minutes

Save

Figure 6.3: Edit component detail



Resource Occupancy
Title: Example Component

1h: 1h, 8h, 12h, 1d, 3d, 7d

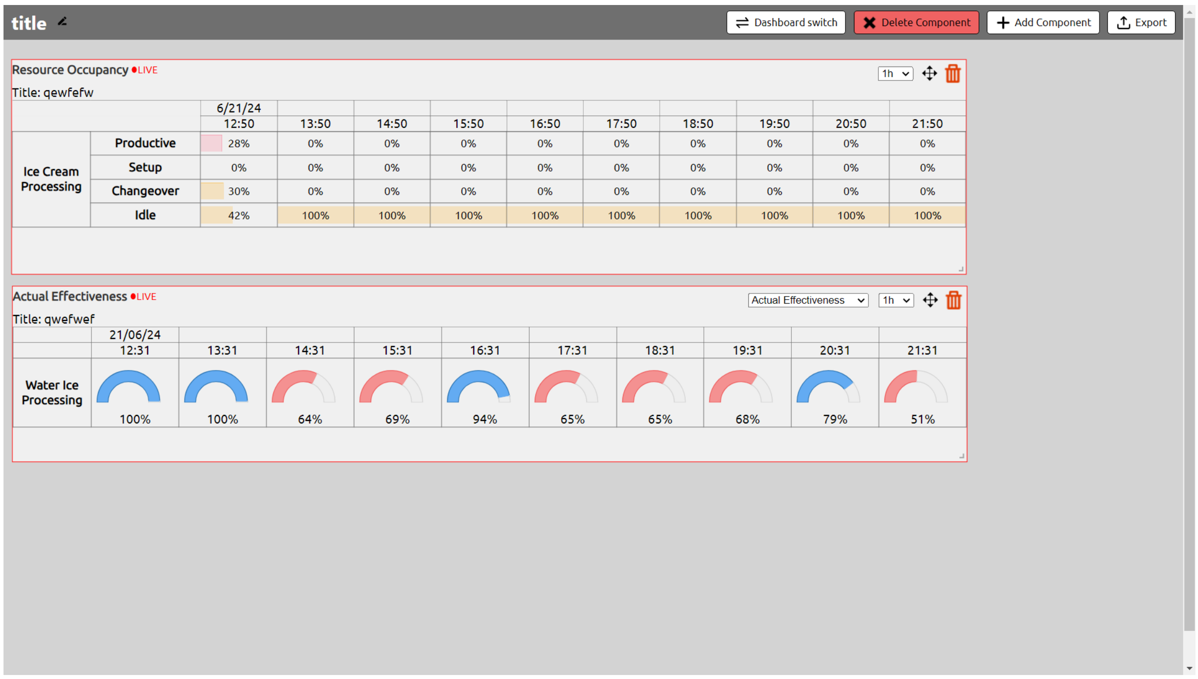
		5/31/24 09:00	5/31/24 10:00	5/31/24 11:00	5/31/24 12:00	5/31/24 13:00	5/31/24 14:00	5/31/24 15:00	5/31/24 16:00	5/31/ 17:00	/31/24 8:00
Ice Cream Processing	PL	95%	37%	46%	13%	94%	0%	49%	49%	79	62%
	PR	7%	31%	79%	14%	31%	42%	76%	62%	76%	15%
	CL	93%	5%	35%	64%	15%	34%	33%	70%	84%	54%

Figure 6.4: Editing time bucket of a component

Editing a component(figure 6.3) is straightforward and can be done by clicking the settings icon on the top right corner of a component as well as the time bucket can be easily changed as the figure 6.4 shows. This opens up the same form that was used to add the component, ensuring that users can easily adjust configurations as their needs change.

The time bucket for a component, which determines the time frame of the data displayed, can be edited using a dropdown menu. This allows users to view aggregations of data over different periods, providing flexibility in data analysis.

Removing components



Resource Occupancy •LIVE

Title: qewfefw

		6/21/24 12:50	13:50	14:50	15:50	16:50	17:50	18:50	19:50	20:50	21:50
Ice Cream Processing	Productive	28%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	Setup	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	Changeover	30%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	Idle	42%	100%	100%	100%	100%	100%	100%	100%	100%	100%

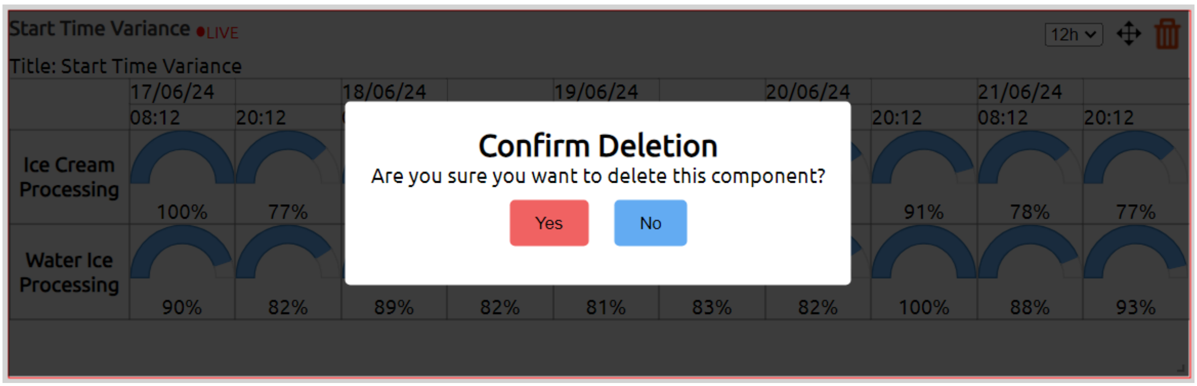
Actual Effectiveness •LIVE

Title: qwefwef

	21/06/24 12:31	13:31	14:31	15:31	16:31	17:31	18:31	19:31	20:31	21:31
Water Ice Processing	100%	100%	64%	69%	94%	65%	65%	68%	79%	51%

Figure 6.5: Delete mode enabled dashboard

To remove components from the dashboard, users can click the "Delete components" button. This action enables delete mode as the figure 6.5 displays, which is highlighted by the red borders around the components. When this is enabled, each component will display a trashcan button. Clicking the "Remove components" button again will disable delete mode, preventing accidental deletions and providing a user-friendly experience.



Start Time Variance •LIVE

Title: Start Time Variance

	17/06/24 08:12	20:12	18/06/24		19/06/24		20/06/24	20:12	21/06/24 08:12	20:12
Ice Cream Processing	100%	77%						91%	78%	77%
Water Ice Processing	90%	82%	89%	82%	81%	83%	82%	100%	88%	93%

Confirm Deletion

Are you sure you want to delete this component?

Figure 6.6: Confirmation of removing a component

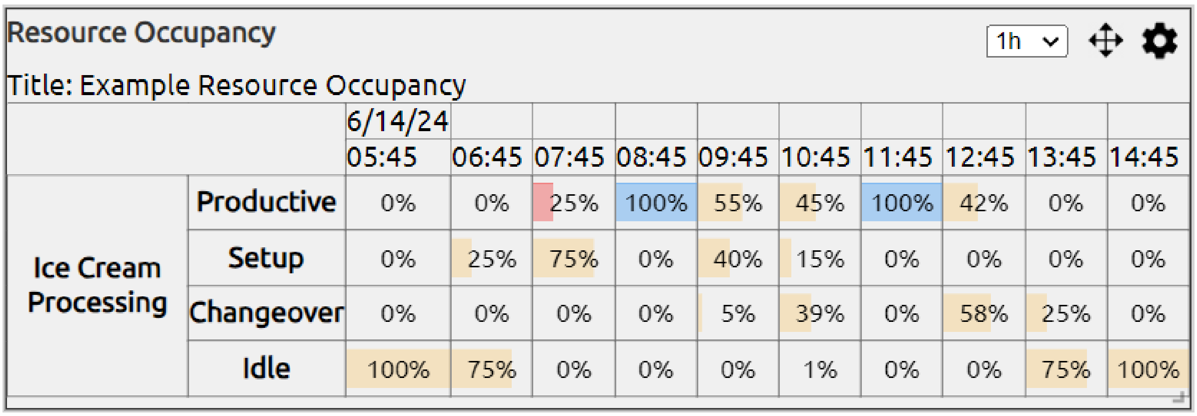
Clicking the trashcan button on a component will show another popup asking for confirmation: this feature was added to prevent accidental deletion and ensure failure safety of the application. Clicking "Yes" will remove the component from the dashboard.

Types of components

DASH+ consists of 5 different types of components that provides different insights to the production plan. Each of these components have titles, dates, and time buckets. Date defines the time where the component will show data from, and time bucket determines the range of time where the component will aggregate data. There are six configured time buckets - 1 hour, 8 hours, 12 hours, 1 day, 3 days, and 7 days - and user can select which one to use when displaying data. The user can also create a component as "Live mode", where the component shows real-time data fetched from database.

The fields that each component uses from the database and the calculation logic can be found in Appendix E

Resource Occupancy


Resource Occupancy

Title: Example Resource Occupancy

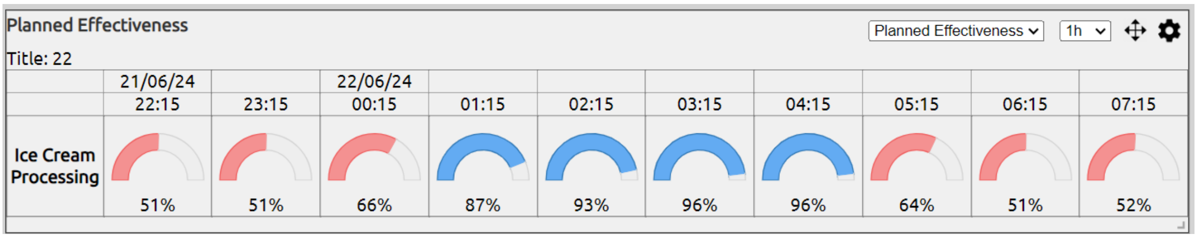
1h

		6/14/24 05:45	06:45	07:45	08:45	09:45	10:45	11:45	12:45	13:45	14:45
Ice Cream Processing	Productive	0%	0%	25%	100%	55%	45%	100%	42%	0%	0%
	Setup	0%	25%	75%	0%	40%	15%	0%	0%	0%	0%
	Changeover	0%	0%	0%	0%	5%	39%	0%	58%	25%	0%
	Idle	100%	75%	0%	0%	0%	1%	0%	0%	75%	100%

Figure 6.7: Example resource occupancy component

Resource Occupancy shows how a resource is being used over time. It shows the ratio of productive time, setup time, changeover time, and idle time of a resource in a given time period, which is calculated using the production batch schedules. This component helps the user evaluate the efficiency of the plan in using the selected resources. In order to help the user interpret the data quicker and easier, the percentages are also displayed in bars which are colour-coded:

Effectiveness


Planned Effectiveness

Title: 22

Planned Effectiveness

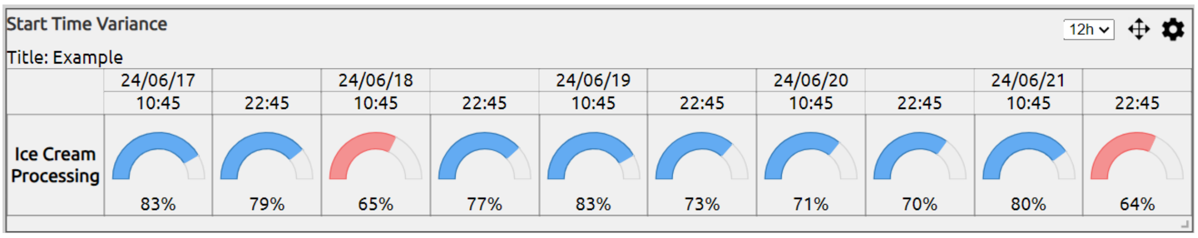
1h

	21/06/24 22:15	23:15	22/06/24 00:15	01:15	02:15	03:15	04:15	05:15	06:15	07:15
Ice Cream Processing	51%	51%	66%	87%	93%	96%	96%	64%	51%	52%

Figure 6.8: Example effectiveness component

Effectiveness shows how effective a production is for a resource over time. Using the products that are produced by the resource in the time period, this component compares the production time and quantity and compares with how much is expected to produce. This component can show this data for both the planned production and actual production, so that the user can see how effective their use of the resource is in both their plan and practice. Effectiveness also makes use of colours to communicate, where percentage higher than 70% uses blue, between 50%and 70% uses mild red, and below 50% uses bright red.

Start Time Variance


Start Time Variance

12h

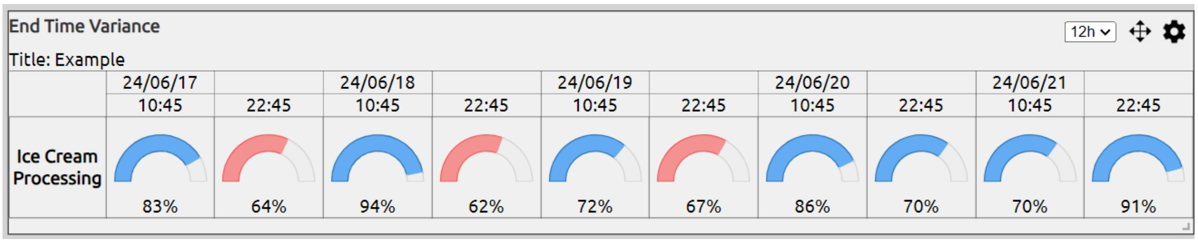
Title: Example

	24/06/17 10:45	22:45	24/06/18 10:45	22:45	24/06/19 10:45	22:45	24/06/20 10:45	22:45	24/06/21 10:45	22:45
Ice Cream Processing	83%	79%	65%	77%	83%	73%	71%	70%	80%	64%

Figure 6.9: Example start time variance component

Start time variance component takes an extra input of threshold time from the user. This component displays the ratio of production batches that have start time delays larger than the threshold. For consistency, this component uses the same layout as effectiveness component.

End Time Variance


End Time Variance

12h

Title: Example

	24/06/17 10:45	22:45	24/06/18 10:45	22:45	24/06/19 10:45	22:45	24/06/20 10:45	22:45	24/06/21 10:45	22:45
Ice Cream Processing	83%	64%	94%	62%	72%	67%	86%	70%	70%	91%

Figure 6.10: Example end time variance component

Similar to start time variance component, end time variance component shows the ratio of production batches that have end time delays larger than the threshold.

Product Variance

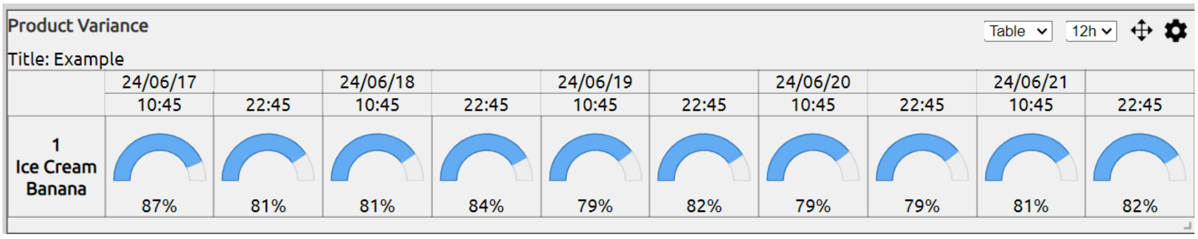

Product Variance

Table 12h

Title: Example

	24/06/17 10:45	22:45	24/06/18 10:45	22:45	24/06/19 10:45	22:45	24/06/20 10:45	22:45	24/06/21 10:45	22:45
1 Ice Cream Banana	87%	81%	81%	84%	79%	82%	79%	79%	81%	82%

Figure 6.11: Example product variance component

Product variance shows the ratio of produced quantity to planned quantity of a product over given time period. This component is similar to effectiveness, but product variance shows the data per product, which is helpful to understand whether a product is being produced efficiently.

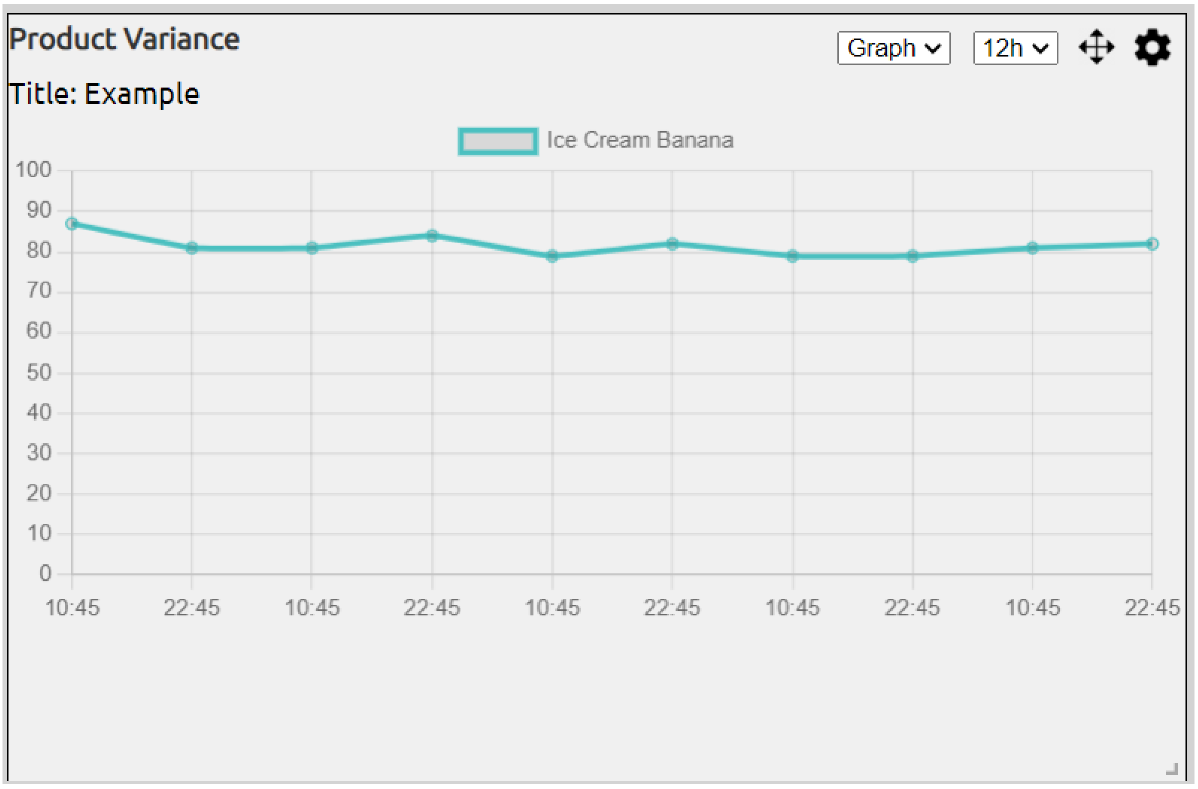


Figure 6.12: Line graph visualization

Product variance component also has another version of data display, which is a line graph. This helps to show the trend in the variance in selected products over time, which can help the user to evaluate their production for different goods and diagnose potential problems.

7 Testing

Testing is the most essential part of the development process, as it helps for developers to find bugs easily, and ensure the software follows the given requirements. This chapter will illustrate what kinds of testing methods are used during the project. In section 7.1 unit testing will be explained with various relevant methods. After that, there will be a description of integration testing in section 7.2, and then the system testing in section 7.3. Finally, the last section 7.4 will have results of the tests and coverage.

7.1. Unit Testing

Unit testing, the smallest testable part of software development, is performed by developers focusing on individual units rather than the entire system. It is a cost-effective and time-efficient method that allows multiple engineers to conduct tests simultaneously and even while fixing issues[10].

White-box testing

White-box testing is one of the methods in the unit testing, which includes structural, and API testing, involving a comprehensive evaluation of the internal structure of the system[11]. This technique focuses on the internal structure and the flow of inputs and outputs within the application, making it highly effective in detecting and resolving issues before they occur. It is used to test the software, code, and internal structure.

Mock Testing & API Testing

API testing is crucial for ensuring the reliability and stability of services, as it helps detect and resolve issues before they cause major failures, which could result in significant financial and trust losses for organizations[12]. Since our application relies on API Endpoints, conducting API testing and mock testing has been essential.

On the backend side, the key features include storing, loading, and sending data. These are using RESTful APIs, commonly known as web APIs, consist of endpoints, each representing a concrete implementation of a business process. These APIs are generally accessible over HTTP (Hypertext Transfer Protocol) using standard verbs like GET, POST, PUT, and DELETE, and are invoked via specific addresses[12].

For testing these APIs, the existence of a mock database was necessary. Since many data entities in the database are related in a many-to-many relationship, objects of various classes were required. These objects were also created as mock data. The generated mock data were then used to test each functionality through HTTP requests.

On the frontend side, testing API logic involved using mock data to simulate the responses from backend API endpoints. By mocking the return values, we could simulate various scenarios and ensure that the frontend components correctly handled different states and data structures returned by the API. This approach is important for verifying the behavior of components such as data fetching, state management, and UI rendering under different conditions without relying on the actual backend services. By employing mock data, we could thoroughly test error handling, edge cases, and different data payloads, thereby enhancing the reliability of our frontend application.

Coverage-based Testing

In software product line testing (SPLT), the test coverage criterion is a crucial concept as it measures the extent of domain testing performed and helps avoid redundant application testing based on the achieved test coverage level. However, no previous literature reviews on SPLT have specifically addressed the test coverage criterion[13].

Coverage.py was used to report coverage of the testing on the backend side, and for the frontend side, *junit* was used for generating the test reports regarding the code coverage. Our internal goal for the coverage was 90% of codes are covered by the unit tests as the Karl Meinke[14] mentioned testing can stop when 100% coverage is achieved under an agreed coverage model, which should be precise and measurable. In practice, testing may terminate at 90% coverage or another acceptable figure for pragmatic reasons. The role of coverage is to define for the testing team how much testing is sufficient.

7.2. Integration Testing

Integration testing ensures that multiple components work together as expected within the application. This involves testing interactions between components to ensure they integrate seamlessly and maintain consistent behavior across different screens or parts of the application. By using tools like Jest and React Testing Library, we can render multiple components together to verify their behaviour and interactions. Integration testing often involves simulating user interactions such as clicking buttons, entering text, or navigating through the application.

DASH+ uses Redux for state management, therefore integration testing often involves not only testing components together but also how they interact with the Redux store and its integration with React components. Testing Redux involves verifying the following aspects: firstly, ensuring reducers are properly handling actions and return the expected state changes, maintaining the application's state integrity. Secondly, testing the action creators to guarantee that they produce the intended action objects. Lastly, testing the Redux store by validating that the global state changes correctly based on components dispatching actions, ensuring UI updates reflect state changes. These tests ensure consistent and reliable application behavior throughout development and updates.

7.3. System Testing

To verify that the software requirements, architecture, design, and code meet the business requirements and standards, ensuring these artifacts are error-free and built according to specifications.

System testing covers the end-to-end functions of a system, providing reliability by identifying and solving bugs post-production. It tests the entire system architecture against business requirements and integrates new and existing functionalities, helping users understand the benefits of newly added features[15].

Our team undertook several steps for system testing, ensuring the entire program was tested in a real-world scenario to verify it meets the requirements. The testing was conducted in two ways: one by our team and the other by the client, acting as the actual user.

We created a checklist of the requirements and conducted comprehensive testing of the application. By checking off each completed requirement, we could easily identify which areas were lacking and needed further attention. This method helped us discover unexpected errors. However, since the testing was done by project members familiar with how to compile and run the program, it wasn't perfect.

To gather functional feedback from users without background knowledge of the project's development and code, we also conducted testing with the client. We created a README file to explain the installation and usage procedures, allowing the client to test the application freely. This process was particularly useful for gathering feedback on critical features such as component creation, deletion, and editing, as well as design aspects. It provided valuable insights into how well our program met the client's requirements. Additionally, observing how non-developer users interacted with the application revealed different perspectives and helped us identify unexpected bugs.

7.4. Test result

This section will cover the result of the testing, in terms of the code coverage of the application. As mentioned in Coverage-based Testing, the aim to reach 90% coverage was achieved, as shown in Table 7.1&7.2. Since we assumed some of the code given by Occator had been tested, this figure shows the only coverage for the codes we created, full coverage can be shown in Appendix F. The coverage reports illustrate the number of lines of code that should be covered in the first column and the number of missed lines in the second column. Excluding tests that were deemed unnecessary, it is evident that most of the code was tested smoothly.

File	% Stmts	% Branch	% Funcs	% Lines
All files	95.23	87.55	96.8	95.62
components/Button	100	100	100	100
DeleteButton.jsx	100	100	100	100
SaveButton.jsx	100	100	100	100
components/DeleteConfirmation	100	100	100	100
DeletePopup.jsx	100	100	100	100
components/Utils	100	80	100	100
percentageTranslator.js	100	80	100	100
store/actions	93.51	82.5	95.65	93.79
componentActions.js	86.2	77.63	90.32	85.48
dashboardActions.js	100	90	100	100
pbatchEFActions.js	100	100	100	100
pbatchETActions.js	100	100	100	100
pbatchPVActions.js	100	100	100	100
pbatchSTActions.js	100	100	100	100
productsActions.js	100	100	100	100
resourcesActions.js	100	100	100	100
speedActions.js	93.75	80	100	100
store/reducers	100	96.47	100	100
componentReducer.js	100	90	100	100
dashboardReducer.js	100	90	100	100
pbatchEFReducer.js	100	100	100	100
pbatchETReducer.js	100	100	100	100
pbatchPVReducer.js	100	100	100	100
pbatchROReducer.js	100	100	100	100
pbatchSTReducer.js	100	100	100	100
pbatchTimebucketReducer.js	100	100	100	100
productsReducer.js	100	100	100	100
resourcesReducer.js	100	80	100	100
speedReducer.js	100	100	100	100

Table 7.1: Part of the coverage Report for frontend

Name	**Stmts**	**Miss**	**Cover (%)**
core/__init__.py	0	0	100
coreapp/__init__.py	0	0	100
coreapp/admin.py	1	0	100
coreapp/api_tests/__init__.py	0	0	100
coreapp/api_tests/test_api_component_effectiveness.py	91	5	95
coreapp/api_tests/test_api_component_occupancy.py	86	5	94
coreapp/api_tests/test_api_component_quantity_variance.py	85	2	98
coreapp/api_tests/test_api_dashboard.py	74	5	93
coreapp/api_tests/test_api_nominal_speed.py	46	3	93
coreapp/api_tests/test_api_order_view_set.py	48	3	94
coreapp/api_tests/test_api_pbatch_view_set.py	82	7	91
coreapp/api_tests/test_api_start_end.py	126	9	93
coreapp/apps.py	4	0	100
coreapp/migrations/0001_initial.py	12	0	100
coreapp/migrations/0002_pbatch_shiftpattern_trafficlight_and_more.py	6	0	100
coreapp/migrations/0003_remove_componentperfectorder_product_and_more.py	4	0	100
coreapp/migrations/0004_componentdeliverytime_is_live_and_more.py	4	0	100
coreapp/migrations/0005_componentdeliverytime_time_bucket_and_more.py	4	0	100
coreapp/migrations/0006_componentdeliverytime_estimated_delivery_time.py	4	0	100
coreapp/migrations/0007_componentend_componentproductionquantity_and_more.py	5	0	100
coreapp/migrations/0008_componentproductionquantityvariance_and_more.py	5	0	100
coreapp/migrations/0009_rename_actual_end_pbatch_actual_end_inflow_and_more.py	4	0	100
coreapp/migrations/__init__.py	0	0	100
coreapp/model.py	6	6	0
coreapp/serializers.py	1	1	0
coreapp/serializers/__init__.py	0	0	100
coreapp/settings.py	19	0	100
coreapp/settings_test.py	2	0	100
coreapp/tests/__init__.py	0	0	100
models/core/components.py	99	5	95
models/core/dashboard.py	12	1	92
models/serializer.py	69	0	100
models/settings.py	19	19	0
models/urls.py	19	0	100
models/views.py	168	13	92

Table 7.2: Part of the coverage Report for backend

8 Evaluation&Limitations

This chapter covers the evaluations conducted throughout the project. Usability assessments, based on the 12 fundamental design principles[16], were performed alongside evaluations guided by Nielsen's heuristic evaluation methodology[17]. Section 8.1 details the processes, methods, and results of our evaluations. In section 8.2, we introduce the metrics used during the evaluation process. In addition to that, section 8.3. Finally, sections 8.4 and 8.5 discusses the limitations encountered during the project and development process, and outlines the remaining tasks for future work.

8.1. Evaluation Process

In order to evaluate our application and improve our design, we conducted a series of user experiments and gathered feedback from Occator's clients. A usability test was conducted, where the clients were given the following materials:

- The demo version of our application
- The evaluation metrics (section 8.2)
- A consent form
- An instruction sheet that asks the client to perform certain actions

Clients were asked to interact with the application, focusing on tasks such as adding and removing components, moving and resizing components, and switching to different dashboards. The twelve evaluation metrics assessed various aspects of usability, including visibility, consistency, navigation, etc. After completing the tasks, clients provided feedback through a structured evaluation sheet and questionnaire.

The collected data was analyzed to identify common issues and areas for improvement. Key findings indicated that while most users found the application intuitive and effective for their needs, some suggested minor upgrades. This feedback was instrumental in refining our design and ensuring the application met the high standards expected by Occator's clients.

8.2. Evaluation Metrics

Heuristic evaluation is a usability engineering method aimed at identifying usability problems in user interface designs to address them in an iterative design process. Each evaluator inspects the interface independently to avoid bias. They then aggregate their findings to form a comprehensive list of usability issues. In our case, we applied the design principles to the testing outlined in the chapter introduction, along with our team's interpretation of each criterion. These principles, detailed in [16], are listed in Appendix D.

To quantify the usability of the interface, we used several metrics with a list of tasks (FigureD.3, Appendix D) during the evaluation:

- Severity Ratings: Evaluators rated the severity of each usability issue on a scale from 1 to 10
- Task Completion Rate: The percentage of tasks successfully completed by users during testing.
- Error Rate: The frequency of errors made by users during task completion.

8.3. Evaluation result

This section provides a detailed discussion of the results obtained from the comprehensive evaluation conducted as part of our project. For a more extensive review of the evaluation outcomes, please refer to Appendix D.

The two evaluation metric we have evaluated were task completion rate and error rate. There were four testing users, and each user received the same 6 tasks to perform on the application. Among the four users, two have successfully completed all 6 tasks, and the remaining two have finished 5 out of 6 tasks. Therefore the task completion rate was 22/24 = 91.6%. Therefore the error rate is 2/24 = 8.4%. The completion rate is reasonably high, but there is more room for improvement.

According to the severity ratings, another evaluation metric, it was observed that the scores for 'control' and 'constraints' were relatively low. This means that there isn't enough emphasis when a user is operating an action, and the application doesn't prohibit a user from doing a dangerous operation. While supervising the experimenters, it became evident that these areas required attention and improvement to enhance user experience and efficiency. In order to do so, we have made several changes in our applications. The 'delete' mode is now exited when the user switches dashboard, or adds a new component. The button color of confirming the deletion of components have been modified to appropriate colors.

The additional comments that the users have provided were highly valuable as well. Some common opinions were that 'The multiple drop downs can be opened at the same time', 'There are no word limits for the titles', 'Drop downs are kept open until the user clicks the drop down button again'. Based on the evaluation results, we implemented several updates, including 'multiple drop downs are not opened at the same time', 'the word limit of title for dashboard and components is 80 characters', 'drop downs are automatically closed when the user clicks outside the drop down'. These improvements were validated in subsequent meetings, confirming that the application effectively supports decision-making in supply chain management and aligns with user expectations.

8.4. Limitations

While the project achieved significant progress, several limitations were encountered that impacted the overall development and evaluation process. These limitations have hindered the team from fulfilling a few requirements from the user, such as implementing the admin feature and enabling users to access dashboards as a group. Additionally, these constraints affected the comprehensive testing of the application, limiting the ability to fully evaluate its performance and usability under real-world conditions. Following is the list of limitations we have faced during the project:

- The full requirement was given by the client in week 5, which was halfway through the project. This late clarification disrupted our development timeline and required significant adjustments.
- There was a lack of initiative of the group members and miscommunication with the client, causing us to misinterpret the requirements and the client's needs. This led to some features being developed that were not aligned with the client's expectations.
- The absence of real data during the development phase meant we could only use mock data. This limited our ability to test the application under realistic conditions and to fully validate its functionality and performance.

In conclusion, while significant progress was made, these limitations impeded the project's overall success. Addressing these issues in future iterations will be crucial for fully meeting user requirements and ensuring the application performs effectively in real-world scenarios. By improving communication, obtaining timely requirements, and using real data, future projects can avoid these pitfalls.

8.5. Future Work

The application was implemented in a scalable way, ensuring that future enhancements and additions can be integrated smoothly. The codebase is well-structured and thoroughly commented, allowing future programmers to easily understand the functionality and logic of the code without needing to perform an in-depth analysis. This structured approach not only facilitates easier maintenance and debugging but also accelerates the onboarding process for new developers who might join the project in the future. Below is a list of possible improvements and enhancements that could be added to the application:

- **Admin Feature Implementation:** Developing and integrating a comprehensive admin feature is crucial for enhancing the application's flexibility and management capabilities. This feature will enable administrators to define and add new components that would be shown to users, effectively allowing for dynamic updates and customizations without needing to modify the underlying codebase.
  - *Component Management:* Administrators will have the capability to create, update, or delete components within the dashboard. This includes widgets, data visualizations, and other interactive elements. By managing these components through an admin interface, administrators can tailor the dashboard to meet the evolving needs of users, ensuring that relevant and useful information is always displayed.
  - *User and Role Management:* The admin feature will also include functionalities for managing user accounts and roles. Administrators can assign roles to users, granting them specific permissions and access levels. This role-based access control will ensure that sensitive data and critical functionalities are accessible only to authorized personnel.
- **Dashboard by Group:** Currently, the dashboard is bounded to each user, which limits its collaborative potential. Implementing a feature that allows dashboards to be accessible for groups of users will significantly enhance the application's utility, especially in team-based and organizational environments.
  - *Shared Dashboards:* With this feature, groups of users can access shared dashboards, facilitating better collaboration and information sharing. Team members working on the same project or department can view and interact with the same set of data and visualizations, ensuring everyone is on the same page.
  - *Group Permissions:* Administrators can define groups and assign specific permissions to each group. This means certain dashboards or components can be made available only to particular groups, maintaining data security while enabling collaboration. For example, financial data dashboards can be restricted to the finance team, while project status dashboards can be shared with the entire project team.
  - *Customized Views:* Different groups might require different views of the same data. The application can allow for customized dashboard views based on the group's needs and preferences, enhancing the relevance and usability of the data presented.

Incorporating these features will make the application more usable and flexible, catering to diverse organizational needs and improving overall user experience. These enhancements will also future-proof the application by ensuring it can adapt to evolving requirements without major code overhauls. This approach aligns with industry best practices for scalable software development, emphasizing modularity, user-centric design, and efficient maintenance strategies.

9 Ethical Implication

This chapter will elaborate about the ethical implications and concerns that the project contains. When developing our dashboard application, it was imperative to address various ethical considerations, particularly since it handles real data from supply chain management customers. During the development phase, we exclusively utilized dummy data provided by the client, which limited exposure to potential ethical dilemmas. However, the prospect of merging the dashboard application with the OccaSee platform and launching it into the market with real customer data necessitates careful consideration of ethical concerns.

9.1. Confidential business information

Confidential business information is a primary ethical consideration for the dashboard application, given its role in handling OccaSee's sensitive data. As OccaSee is based in Europe and collects information on EU citizens, adherence to the EU General Data Protection Regulation (GDPR) is mandatory. While the current dashboard application operated solely with dummy data, the transition to handling real customer data underscores the need for strict compliance with data protection regulations. Robust measures such as data encryption, access control, and anonymization are imperative to safeguard the privacy and rights of individuals whose data is being processed. The checklist concerning confidential business information is as following:

Ethics Checklist Relevance:

- Does your planned activity involve processing of personal data? : **Yes**
- Does it involve the processing of special categories of personal data? :**Yes**
- Does it involve profiling, systematic monitoring of individuals, or processing of large scale of special categories of data or intrusive methods of data processing? : **Yes**
- Is it planned to export personal data from the EU to non-EU countries? **Yes**

Given the relevance of these factors, strict adherence to GDPR was mandatory. This regulation requires that all personal data be processed lawfully, fairly, and transparently. To comply, the dashboard application must implement data encryption to protect data during storage and transmission. Access controls should be established to ensure that only authorized personnel can access sensitive data. Additionally, anonymization techniques must be applied to personal data to minimize the risk of identifying individuals from the processed data.

9.2. Transparency of data

Transparency emerges as another critical ethical consideration for the dashboard application. Users must have visibility into how their data is utilized and accessed. Following the merger of the dashboard with the OccaSee platform, ensuring transparency becomes essential. Each user should only have access to the plans and dashboards pertinent to them upon logging into the application. This ensures that users are aware of the data being processed and have control over their interactions with the platform. The checklist concerning transparency of data is as following:

Ethics Checklist Relevance:

- Does it involve profiling, systematic monitoring of individuals, or processing of large scale of special categories of data or intrusive methods of data processing? : **Yes**
- Does it involve profiling, systematic monitoring of individuals, or processing of large scale of special categories of data or intrusive methods of data processing? : **Yes**
- Does this activity involve further processing of previously collected personal data? : **Yes**

To ensure transparency, clear and concise communication with users about how their data is being used was vital. Users should be informed about the purposes of data processing, the types of data being processed, and the entities that have access to their data. Implementing a user-friendly interface that provides this information and allows users to manage their data preferences was crucial. Giving clear indications

9.3. Accountability and responsibility

Moreover, accountability and responsibility are integral aspects of addressing ethical concerns in the development and deployment of the dashboard application. We must establish clear roles and processes for data governance within the organization. This includes maintaining detailed records of data processing activities, conducting regular assessments to identify and mitigate risks, and ensuring that group members are adequately trained in data protection and ethical data management practices. The checklist concerning accountability and responsibility is as following:

Ethics Checklist Relevance:

- Does it involve the processing of special categories of personal data? : **Yes**
- Does this activity involve further processing of previously collected personal data? : **Yes**

9.4. Mitigating the risks

Addressing these ethical concerns necessitates a proactive approach that prioritizes the rights, privacy, and well-being of individuals whose data is being processed. For the confidential business information part, we already know that OccaSee implements a firm data protection framework.

OccaSee ensures safe authentication through a robust system where the database never stores the actual password, but only stores the hash of the password. The password hash is generated using a secure hashing algorithm, making it computationally infeasible for an attacker to retrieve the original password even if the hash is compromised. Additionally, the hash of the password is changed whenever the user logs in. This process, known as "hashing with salt," involves combining the password with a unique value before hashing it, which further guarantees the security of the data by ensuring that each password has a unique hash even if the same password is used by multiple users. For transparency, OccaSee implements a structured user authorization system with different levels of access. This ensures that users can view and interact with data that is relevant to their roles and responsibilities. We have also applied transparency during the user tests, as we have informed the testers about where and how the results will be used.

In conclusion, ethical considerations are paramount in the development and deployment of the dashboard application, particularly given its role in handling confidential business information for OccaSee. By adhering to ethical principles, OccaSee can navigate the complexities of data processing responsibly, fostering trust, transparency, and accountability in its interactions with customers and stakeholders.

10 Conclusion and takeaways

This report aimed to describe and elaborate on the decisions that were made during the development of the dashboard application and how we ensured that the application provided insights to help the client make the best decisions in supply chain management.

Our approach for the application was eliciting the requirements and creating a simple design by prototyping. We have cleared out the requirements by communicating with the client, while showing our initial design for any feedback or comments. With the requirements and design given, actual implementation has been done using different frameworks such as Django and React. Lastly, evaluation was done using user experiments in order to search for any drawbacks of the application and enhance user experience.

The result of this process is an interactive dashboard application that meets the specified requirements. Key features include a customizable, user-specific dashboard with components displaying various data about production plans, such as resource occupancy and plan effectiveness. Users can add, remove, and edit components, filter data, and save their dashboard layouts. The application updates data every 10 minutes and supports multiple time bucket sizes (1h, 8h, 12h, 1d, 3d, 7d). Evaluation results showed that the application successfully provided the necessary insights, enabling effective decision-making in supply chain management.

Despite initial challenges such as unfamiliarity with frameworks and ensuring compatibility with existing systems, the team overcame these obstacles through structured learning, seeking guidance from the company and TA, and maintaining open lines of communication. Regular feedback from Occator helped refine the application, resulting in a successful implementation that met the project goals and enhanced the client's supply chain management capabilities.

Looking forward, we recommend several areas for future development:

- Creating a wider range of components to provide even deeper insights about a plan.
- Extending the application to integrate with more data sources for a more comprehensive overview.
- If we were to start a new project, we would further emphasize early user testing to catch potential issues sooner.

In conclusion, the project DASH+ has been a valuable learning experience and has resulted in a useful tool for supply chain management. Our approach and methodology proved effective, and the resulting application meets the client's needs and requirements while providing a foundation for future improvements. The development of the dashboard application not only showcased our technical skills but also underscored the importance of iterative development and user feedback in creating impactful software solutions.

Bibliography

[1] P. Eijgenraam, *Private communication*, Apr. 2024.

[2] S. S. ., “Scrum methodology,” *International Journal of Engineering and Computer Science*, vol. 5, no. 6, May 2016. [Online]. Available: https://www.ijecs.in/index.php/ijecs/article/view/1989.

[3] Occator. “Occasee production planning visibility - communication and availabililty.” (), [Online]. Available: https://occasee.occator.com/.

[4] P. M. G. Knolmayer and A. Zeier, *Supply chain management based on SAP systems: Order management in manufacturing companies*. 2002, ISBN: 978-3-642-08625-0. DOI: 10.1007/978-3-540-24816-3.

[5] E. P. Alex Banks, *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, Inc., 2017, ISBN: 9781491954621.

[6] Django. “Django: The web framework for perfectionists with deadlines.” (), [Online]. Available: https://www.djangoproject.com/.

[7] J. Grady, *System Requirements Analysis*. Elsevier, 2010, ISBN: 9780120885145.

[8] T. Kravchenko, T. Bogdanova, and T. Shevgunov, “Ranking requirements using moscow methodology in practice,” in *Computer Science On-line Conference*, Springer, 2022, pp. 188–199.

[9] “Ranking requirements using moscow methodology in practice,” Springer, Cham, 2022. DOI: https://doi.org/10.1007/978-3-031-09073-8_18.

[10] D. S. Taley and B. Pathak, “Comprehensive study of software testing techniques and strategies: A review,” *Int. J. Eng. Res*, vol. 9, no. 08, pp. 817–822, 2020.

[11] J. Ahmad, A. ul Hasan, T. Naqvi, and T. Mubeen, “A review on software testing and its methodology,” *Manag. J. Softw. Eng*, vol. 13, no. 1, pp. 32–38, 2019.

[12] A. Ehsan, M. A. M. Abuhaliqa, C. Catal, and D. Mishra, “Restful api testing methodologies: Rationale, challenges, and solution directions,” *Applied Sciences*, vol. 12, no. 9, p. 4369, 2022.

[13] J. Lee, S. Kang, and P. Jung, “Test coverage criteria for software product line testing: Systematic literature review,” *Information and Software Technology*, vol. 122, p. 106 272, 2020.

[14] K. Meinke, “Code coverage and test automation: State of the art,” *arXiv preprint arXiv:2108.11723*, 2021.

[15] T. Collins. “What is system testing? (examples, use cases, types).” (), [Online]. Available: https://www.browserstack.com/guide/what-is-system-testing.

[16] C. Sin, “Ux design principles — a scrapbook of examples,” *Medium*, 2018.

[17] J. Nielsen, “How to conduct a heuristic evaluation,” *Nielsen Norman Group*, vol. 1, no. 1, p. 8, 1995.

Appendices

A
Risk Analysis & Feasibility Study

This chapter covers the descriptions of risk analysis and the feasibility study. In section A.1, we introduce the types of risks encountered during the development process and explain how they were evaluated. Additionally, section A.2 illustrates the project's feasibility in terms of technical aspects and timeline.

A.1. Risk Analysis

Ensuring the success of the DASH+ product requires a comprehensive consideration of potential risks throughout its life cycle: before, during, and after development. The following risks have been identified and evaluated:

- **Limited Access to Actual Company Database:** Currently, the team is working with mock data instead of the actual company database. This limitation means that the application can only be tested using mock data, raising concerns about its performance and efficiency in real-life settings with live data. To mitigate this risk, extensive testing and validation procedures will be implemented once access to the live database is granted.
- **Unfamiliarity with Required Frameworks:** The application needs to be built using frameworks that the team is not well-versed in. Although the company can provide support, the learning curve may pose challenges during development. This risk can be minimized through targeted training sessions, thorough documentation review, and seeking assistance from experts within the company.
- **Misalignment of Client Expectations:** There is a potential risk that the client's expectations may not align with the application's functionalities. While this risk is considered low due to the team's proactive approach in maintaining regular communication with the client, it can be further mitigated by employing iterative development methodologies, frequent prototyping, and regular feedback sessions to ensure alignment.
- **Unintentional Disclosure of Confidential Information:** There is a risk of inadvertently sharing confidential information outside the authorized channels. To minimize this risk, the team will strictly use secure communication platforms such as Microsoft Teams for all project-related discussions and ensure that no sensitive information is shared outside the client and TU Delft.
- **Client Delays in Providing Requirements:** Delays from the client in supplying necessary requirements could hinder the commencement and progress of the application's implementation. However, during these delays, the team can utilize the time to further confirm and solidify the requirements and design of the application. This additional time allows for more in-depth discussions and clarification, ensuring that the final product aligns closely with the client's true needs and expectations. This risk can be mitigated by establishing clear timelines, maintaining open lines of communication, and setting up regular check-ins with the client to ensure timely provision of requirements.

A.2. Feasibility Study

The feasibility study is a crucial step in assessing the viability and potential success of a project. This section will list a study based on different aspects of feasibility in order to determine whether the project is worth pursuing. We will focus on technical and schedule-wise aspects of feasibility.

Assessing the technical feasibility involves evaluating whether the project can be successfully implemented considering the technological requirements and the team's expertise. In this context, the project requires proficiency in Django for backend development and React for frontend development, both of which are unfamiliar to us.

The schedule feasibility analysis focuses on determining whether the project can be completed within the allocated time frame of 10 weeks, encompassing all stages from initial client meetings to the delivery of the final product.

By conducting a thorough analysis of both technical and schedule feasibility, the project team can make informed decisions regarding the viability of pursuing the project within the given constraints. This study serves as a foundation for identifying potential challenges and implementing strategies to overcome them, ultimately increasing the likelihood of project success.

When looking at the solution outline we can assess the technical feasibility of each solution:

- **Effective Information Delivery:** Implementing effective information delivery requires knowledge about Django and React, which the team currently lacks. To address this gap, we will engage in targeted learning sessions and utilize available online resources and tutorials. Additionally, collaborating with experienced developers within Occator can provide valuable insights and accelerate the learning process. Despite the initial lack of expertise, we are confident in our ability to achieve effective information delivery through dedication and continuous learning.
- **Effective Visualization as a Dashboard:** Creating a visually appealing and functional dashboard requires expertise in React and data visualization techniques. Since our application is an extension of Occator's existing application, we are required to follow the established design guidelines. While this constraint limits creativity, it enhances feasibility by providing a clear design reference. We will focus on mastering the specific components and visualization libraries used in the existing application. Regular feedback from Occator will ensure that the new dashboard aligns with their standards and enhances user experience.
- **Compatibility with Existing Application:** Ensuring compatibility with Occator's existing application involves a deep understanding of its architecture, data structures, and integration points. This requires technical expertise in backend development and API integration, areas where the team currently faces challenges due to unfamiliarity with the existing system. To overcome this, the team will prioritize studying Occator's documentation and establishing clear communication channels with their technical staff. Regular meetings and code reviews with Occator's development team will be essential to identify and resolve compatibility issues early in the development process, preventing major problems later on.

We are now going to consider the schedule feasibility of each solution:

- **Effective Information Delivery:** The process of categorizing, integrating, and delivering information requires careful planning and communication with Occator. During the 10 weeks of the project, we should allocate enough time each week to show Occator what and how we deliver information to its clients. Information is only effective when it resonates with the client.
- **Effective Visualization as a Dashboard:** Building a good dashboard requires iterative design and development cycles. Since the dashboard should follow the design of the existing application, making drafts of the dashboard and getting it approved by Occator should be done early on so that both parties have the same vision of the end product. During the project, we should have regular meetings regarding the design to ensure the current product is still aligned with their vision and make changes accordingly if needed.
- **Compatibility with Existing Application:** Ensuring compatibility during early development is crucial for the schedule feasibility of the project. If this is not done correctly, there is a possibility that we might

be pressed for time.

Regarding the feasibility of this project, the team definitely thinks the project is viable within the given time constraints. Taking these potential challenges into account during the project and tackling them accordingly should bring us closer to a successful project.

B
Requirements

Requirement analysis is crucial for gathering stakeholders' expectations for the final product and ensuring a unified understanding between the development team and the client. This process ensures that the final product aligns with user requirements and business goals. The initial requirements were provided by the client and evolved through discussions. This chapter delineates the functional and non-functional requirements for a dashboard application.

B.1. Terminology

- **Plan:** A production plan (a schedule).
- **Planboard:** A tab in OccaSee that shows a plan.
- **Component:** A graph, representation, table, or numeric value that contains data about a plan.
- **Dashboard/Board:** A tab in OccaSee that displays multiple components.

B.2. Functional Requirements

Must have

- The application has a dashboard.
- Dashboards are user-specific (saved per user).
- The application has components that show different data about the plan.
- A component has a title and a description.
- Each component must be able to show data in different time buckets.
- The size of time buckets can be selected by the user: 1h, 8h, 12h, 1d, 3d, 7d.
- The component must have a filter to filter plans per product, resource, or custom-made selection.
- Users can see a dropdown list of available components.
- Users can add new components to the dashboard from a dropdown list of available components.
- Users can delete a component from the dashboard.
- The dashboard will automatically save all the user's dashboard changes
- The dashboard fetches new data and renders it every 10 minutes.
- Users must be able to edit the layout of the dashboard.
- There must be a component showing resource occupancy in a selected time bucket with percentage values for: productive, non-productive (maintenance and downtime), setup, changeover, idle
- There must be a component showing information about the effectiveness of the plan.
- The components are movable and resizable

Should have

- Users can create a new board and switch between boards using a dropdown list.
- Users can name the dashboard
- There should be a component calculating KPI information following a predefined formula.
- The application must make the results exportable to PDF.

- Users can switch between plans.
- A group of users can share boards, utilizing the group functionality already supported in the OccaSee application.

Could have

- Users can import data (txt, XML, PDF, etc.) from external sources.
- There could be a component showing information about the quality.

B.2.1. Won't have

- The dashboard will not automatically refresh every time there is a change in the database. This would cause a very laggy environment.
- The component showing and comparing two full dashboards at the same time will not be implemented.
- The dashboard will not have slots for resizing and moving components.
- There could be a component showing forecast coverage.

B.3. Non-functional Requirements

Non-functional requirements specify the system's operational capabilities and constraints, ensuring that our dashboard application is robust, user-friendly, and compatible with various environments and platforms. Below are the key non-functional requirements for the application, explained briefly.

B.3.1. Development Tools

The choice of development tools was selected considering the original OccaSee application's languages to ensure consistency, maintainability, and to leverage the team's expertise.

- **Frontend Development: JavaScript (React) and Redux** - React and Redux are widely-used frameworks that offer efficient state management and component-based architecture. This choice allows for scalable, maintainable, and efficient frontend development, ensuring a responsive and dynamic user interface.
- **Backend Development: Python (Django)** - Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It includes numerous built-in features for handling security, database access, and application management, which simplifies backend development and ensures robust application performance.
- **Global Variables** - Utilizing global variables provided by the system, where possible, ensures consistency across the application, reducing redundancy and potential errors. It also helps in maintaining a unified codebase, making the application easier to manage and update.

B.3.2. Project Organization

Organizing project files in a structured manner is essential for maintaining clarity and efficiency in the development process.

- **Project File Organization** - Adopting a project organization similar to OccaSee provides a familiar structure for the development team, enhancing productivity and reducing the learning curve. It also ensures that best practices and successful patterns from OccaSee are reused, promoting consistency and quality in the project.

B.3.3. Database Management

Effective database management ensures data integrity and seamless integration with the application's components.

- **Mocked Database** - Using the provided mocked database while keeping the original models unchanged ensures that development and testing can proceed without affecting the live data. This approach helps in maintaining data integrity and allows for robust testing scenarios, reducing the risk of errors in the production environment.

B.3.4. Compatibility and Adaptability

Ensuring compatibility with multiple operating systems and adaptability to different screen sizes is crucial for providing a versatile user experience.

- **Operating System Compatibility** - The application must be able to run on the latest versions of Windows and Mac OS. This ensures that a broad user base can access the application, regardless of their preferred operating system, thereby increasing the application's usability and market reach.
- **Screen Size Adaptability** - The application should be adaptable to multiple screen sizes. This requirement ensures that users can access the dashboard on various devices, from desktop monitors to tablets and mobile phones, providing a seamless and consistent user experience across different platforms.

By addressing these non-functional requirements, we ensure that the application is built on a solid foundation, offering reliability, compatibility, and ease of use. This approach helps in delivering a high-quality product that meets user expectations and industry standards.

C

Medium Fidelity

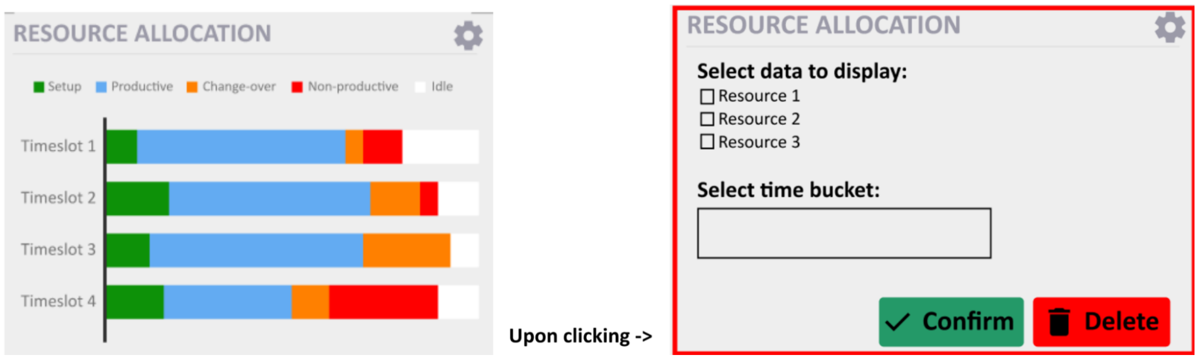
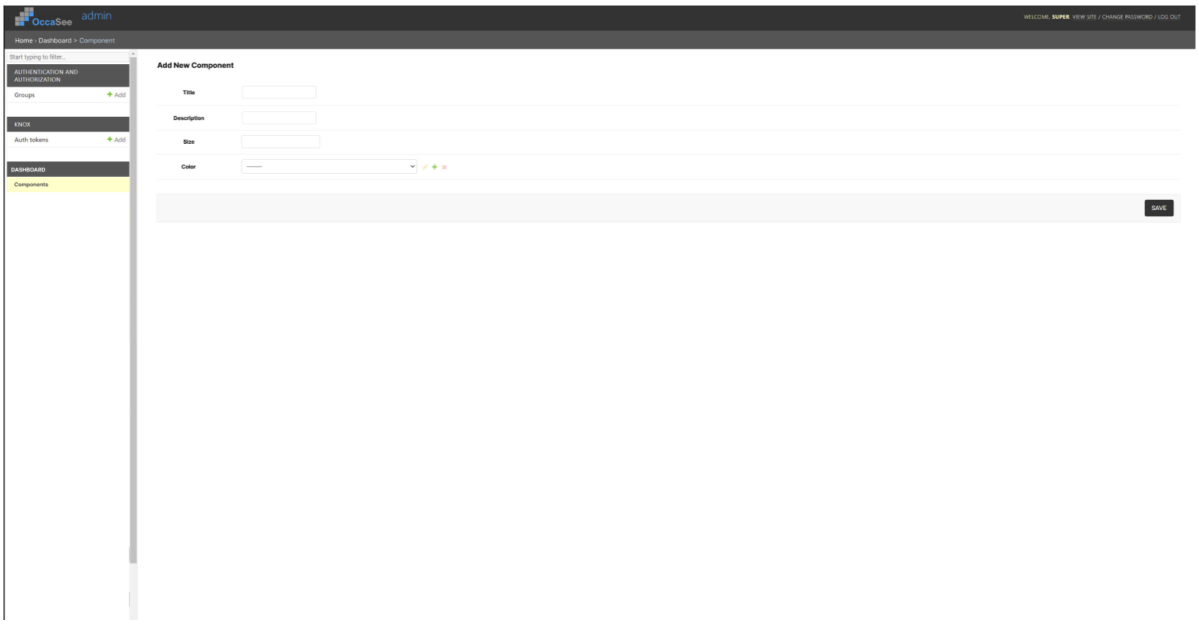


Figure C.1: Example functionality: editing a component


OccaSee admin
WELCOME, SUPER. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Dashboard > Component
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
KNOX
Auth tokens + Add
DASHBOARD
Components
SAVE

Add New Component

Title	
Description	
Size	
Color	———

Figure C.2: Admin add component

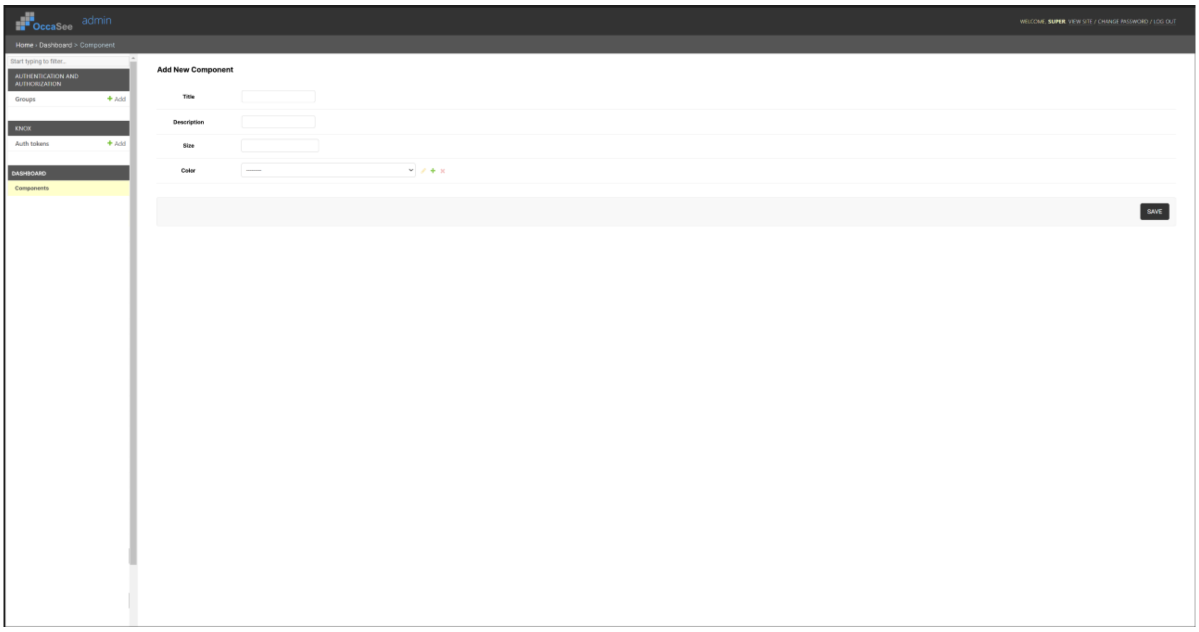

OccaSee admin
WELCOME, SUPER. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Dashboard > Component
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups
Add
KNOX
Auth tokens
Add
DASHBOARD
Components
Add New Component
Title
Description
Size
Color
SAVE

Figure C.3: Admin add component

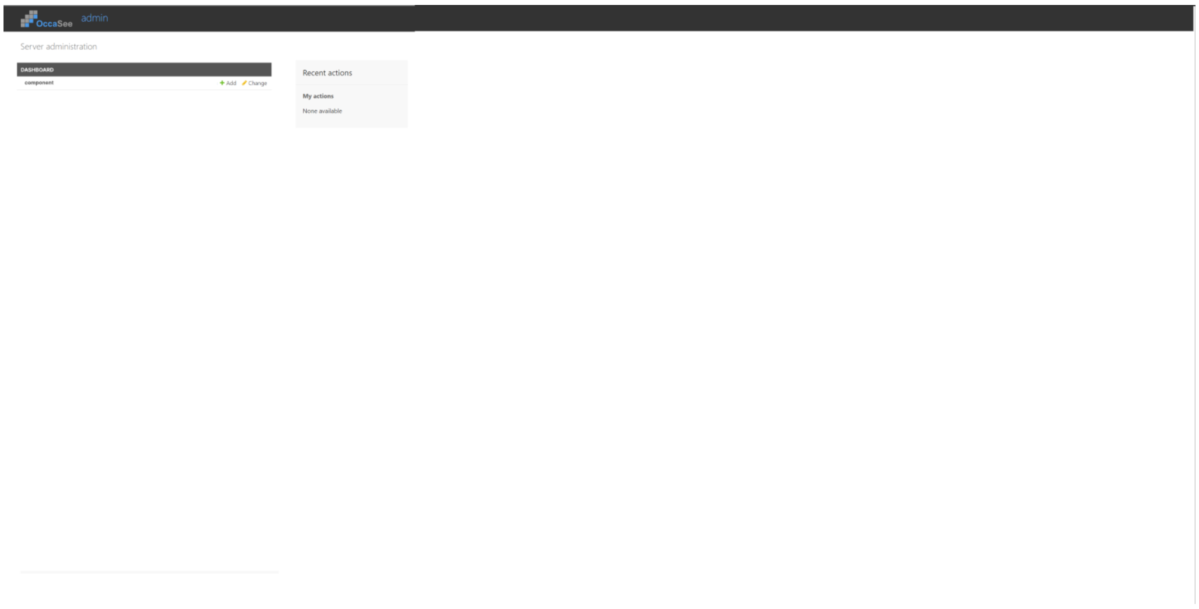
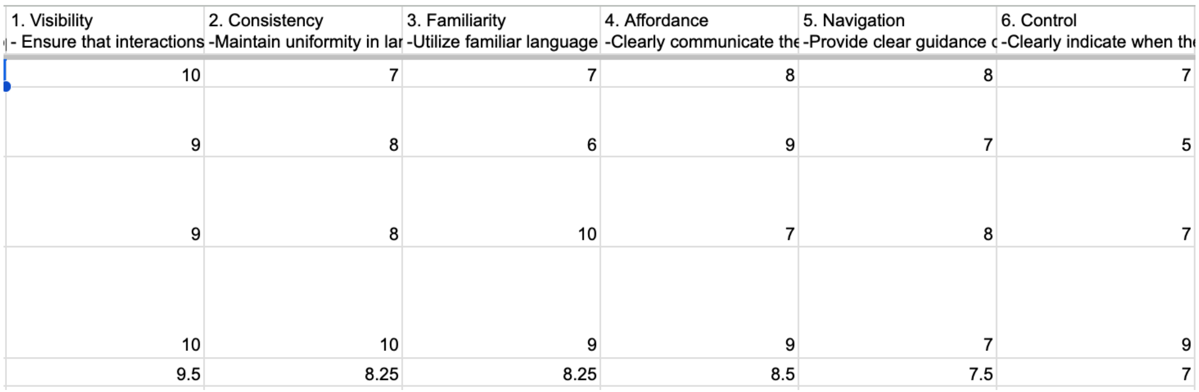


Figure C.4: Admin dashboard-component

D
Evaluation Results & Design principle

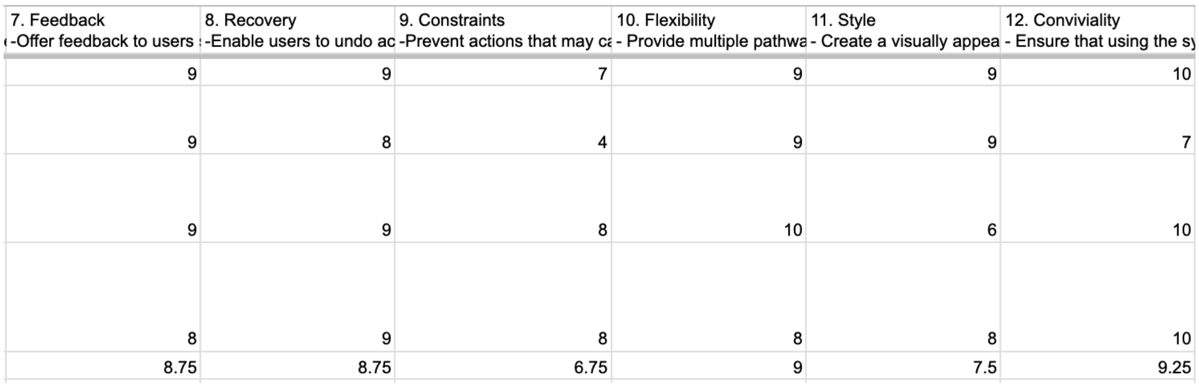
This is the list of the design principles, along with our interpretation.

- Visibility: Ensure that interactions with the system produce observable outcomes.
- Consistency: Maintain uniformity in language, design, and concepts throughout the application.
- Familiarity: Utilize familiar language and symbols or provide clear introductions to aid user understanding.
- Affordance: Clearly communicate the purpose and usage of tools within the application.
- Navigation: Provide clear guidance on how users can navigate within the application.
- Control: Clearly indicate when the user or the system is in control. Clearly indicates when a textbox, dropdown, etc are selected
- Feedback: Offer feedback to users so they can discern the effects of their actions.
- Recovery: Enable users to undo actions or recover from errors easily.
- Constraints: Prevent actions that may cause the program to crash or behave unexpectedly.
- Flexibility: Provide multiple pathways to achieve goals, allowing users to adapt their approach.
- Style: Create a visually appealing and stylish design.
- Conviviality: Ensure that using the system is a friendly and enjoyable experience without disruptions.



1. Visibility - Ensure that interactions	2. Consistency -Maintain uniformity in lar	3. Familiarity -Utilize familiar language	4. Affordance -Clearly communicate the	5. Navigation -Provide clear guidance c	6. Control -Clearly indicate when th
10	7	7	8	8	7
9	8	6	9	7	5
9	8	10	7	8	7
10	10	9	9	7	9
9.5	8.25	8.25	8.5	7.5	7

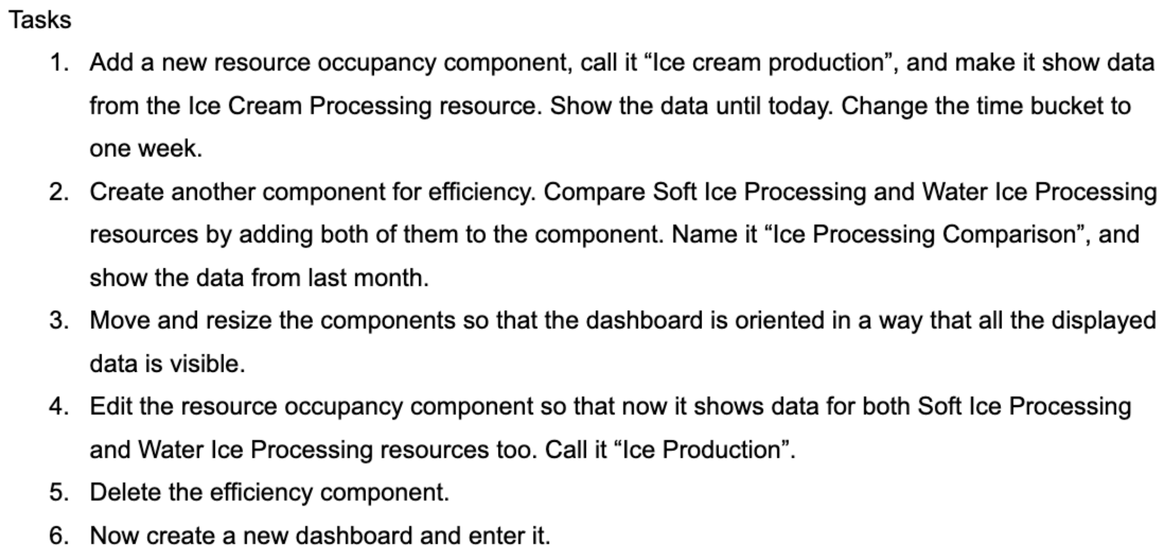
Figure D.1: Heuristic Evaluation



7. Feedback -Offer feedback to users	8. Recovery -Enable users to undo ac	9. Constraints -Prevent actions that may ca	10. Flexibility - Provide multiple pathwa	11. Style - Create a visually appea	12. Conviviality - Ensure that using the sy
9	9	7	9	9	10
9	8	4	9	9	7
9	9	8	10	6	10
8	9	8	8	8	10
8.75	8.75	6.75	9	7.5	9.25

Figure D.2: Heuristic Evaluation

Figure D.1, D.2 illustrates the average result of the user testing in terms of design principles. [16]

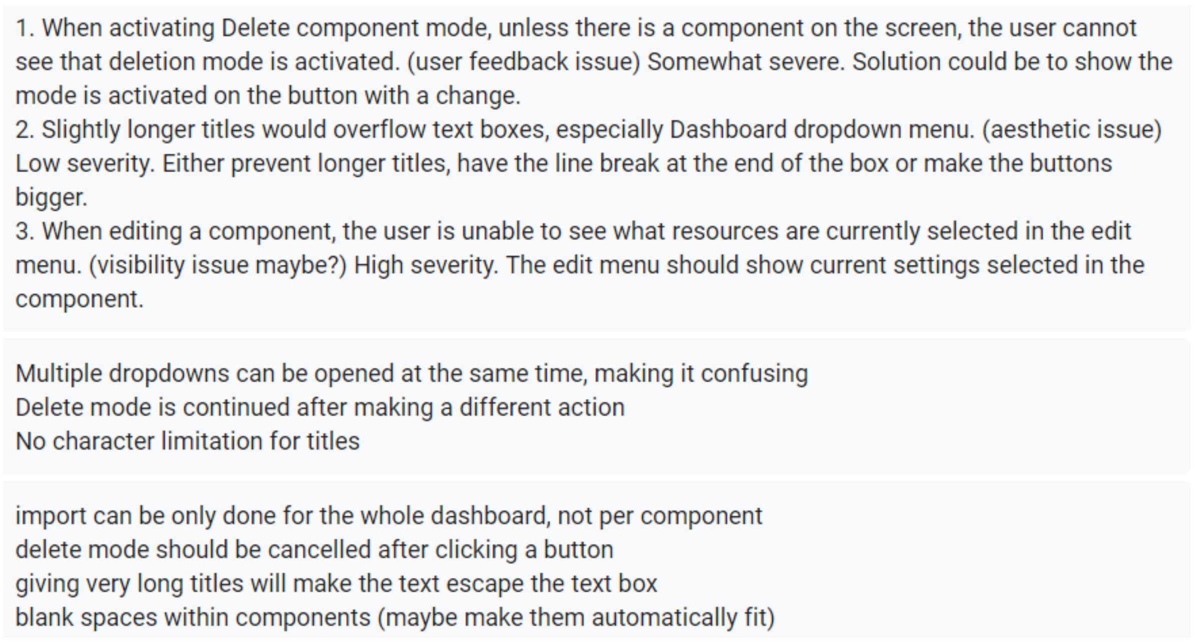


Tasks

1. Add a new resource occupancy component, call it “Ice cream production”, and make it show data from the Ice Cream Processing resource. Show the data until today. Change the time bucket to one week.
2. Create another component for efficiency. Compare Soft Ice Processing and Water Ice Processing resources by adding both of them to the component. Name it “Ice Processing Comparison”, and show the data from last month.
3. Move and resize the components so that the dashboard is oriented in a way that all the displayed data is visible.
4. Edit the resource occupancy component so that now it shows data for both Soft Ice Processing and Water Ice Processing resources too. Call it “Ice Production”.
5. Delete the efficiency component.
6. Now create a new dashboard and enter it.

Figure D.3: User testing task

Figure D.3 is the list of instructions given to the user who tests the application.


1. When activating Delete component mode, unless there is a component on the screen, the user cannot see that deletion mode is activated. (user feedback issue) Somewhat severe. Solution could be to show the mode is activated on the button with a change.
2. Slightly longer titles would overflow text boxes, especially Dashboard dropdown menu. (aesthetic issue) Low severity. Either prevent longer titles, have the line break at the end of the box or make the buttons bigger.
3. When editing a component, the user is unable to see what resources are currently selected in the edit menu. (visibility issue maybe?) High severity. The edit menu should show current settings selected in the component.

Multiple dropdowns can be opened at the same time, making it confusing
Delete mode is continued after making a different action
No character limitation for titles

import can be only done for the whole dashboard, not per component
delete mode should be cancelled after clicking a button
giving very long titles will make the text escape the text box
blank spaces within components (maybe make them automatically fit)

Figure D.4: Evaluation result

Figure D.4

E
Data Calculation Logic

Database Entities

Production Batch

- start_inflow: The expected start time of inflow
- end_inflow: The expected end time of inflow
- start_outflow: The expected start time of outflow
- end_outflow: The expected end time of outflow
- actual_start_inflow: The actual start time of inflow
- actual_end_inflow: The actual end time of inflow
- actual_start_outflow: The actual start time of outflow
- actual_end_outflow: The actual end time of outflow
- setup: The setup time for the resource in minutes
- changeover: The changeover time for the resource in minutes
- resources: The resource which the production batch is working on
- product: The production being produced in this production batch
- quantity: The expected amount of quantity to be produced
- produced_quantity: The actual amount of quantity to be produced

Resource

- id: The identification key or the resource
- name: The name of the resource

Product

- code: The unique code of product
- description: The description of the product

Nominal Speed

- resource: The resource that this nominal speed represents
- product: The product that this nominal speed represents
- nominal_speed: The nominal speed, in the unit of hours

E.1. Resource Occupancy

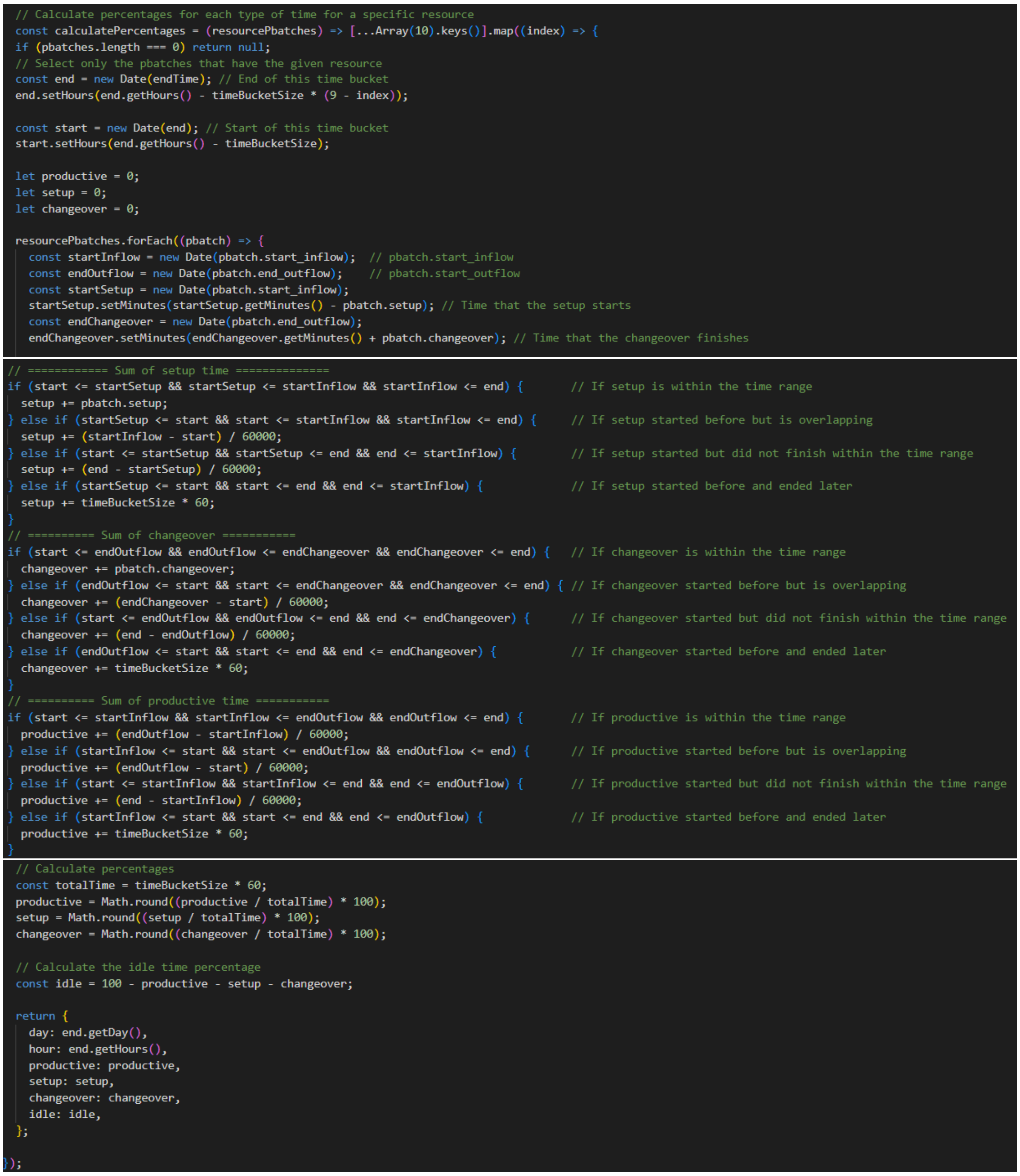
E.1.1. User Input Fields

1. Resource
2. Start time
3. Time bucket

E.1.2. Fetched Fields

1. pbatch.start_inflow
2. pbatch.end_outflow
3. pbatch.setup
4. pbatch.changeover

E.1.3. Calculation



```
 // Calculate percentages for each type of time for a specific resource
 const calculatePercentages = (resourcePbatches) => [...Array(10).keys()].map((index) => {
 if (pbatches.length === 0) return null;
 // Select only the pbatches that have the given resource
 const end = new Date(endTime); // End of this time bucket
 end.setHours(end.getHours() - timeBucketSize * (9 - index));

 const start = new Date(end); // Start of this time bucket
 start.setHours(end.getHours() - timeBucketSize);

 let productive = 0;
 let setup = 0;
 let changeover = 0;

 resourcePbatches.forEach((pbatch) => {
   const startInflow = new Date(pbatch.start_inflow);  // pbatch.start_inflow
   const endOutflow = new Date(pbatch.end_outflow);    // pbatch.start_outflow
   const startSetup = new Date(pbatch.start_inflow);
   startSetup.setMinutes(startSetup.getMinutes() - pbatch.setup); // Time that the setup starts
   const endChangeover = new Date(pbatch.end_outflow);
   endChangeover.setMinutes(endChangeover.getMinutes() + pbatch.changeover); // Time that the changeover finishes
```

```
// =========== Sum of setup time =============
if (start <= startSetup && startSetup <= startInflow && startInflow <= end) {       // If setup is within the time range
  setup += pbatch.setup;
} else if (startSetup <= start && start <= startInflow && startInflow <= end) {     // If setup started before but is overlapping
  setup += (startInflow - start) / 60000;
} else if (start <= startSetup && startSetup <= end && end <= startInflow) {        // If setup started but did not finish within the time range
  setup += (end - startSetup) / 60000;
} else if (startSetup <= start && start <= end && end <= startInflow) {             // If setup started before and ended later
  setup += timeBucketSize * 60;
}
// ========== Sum of changeover ==========
if (start <= endOutflow && endOutflow <= endChangeover && endChangeover <= end) {   // If changeover is within the time range
  changeover += pbatch.changeover;
} else if (endOutflow <= start && start <= endChangeover && endChangeover <= end) { // If changeover started before but is overlapping
  changeover += (endChangeover - start) / 60000;
} else if (start <= endOutflow && endOutflow <= end && end <= endChangeover) {      // If changeover started but did not finish within the time range
  changeover += (end - endOutflow) / 60000;
} else if (endOutflow <= start && start <= end && end <= endChangeover) {           // If changeover started before and ended later
  changeover += timeBucketSize * 60;
}
// ========== Sum of productive time ===========
if (start <= startInflow && startInflow <= endOutflow && endOutflow <= end) {       // If productive is within the time range
  productive += (endOutflow - startInflow) / 60000;
} else if (startInflow <= start && start <= endOutflow && endOutflow <= end) {      // If productive started before but is overlapping
  productive += (endOutflow - start) / 60000;
} else if (start <= startInflow && startInflow <= end && end <= endOutflow) {       // If productive started but did not finish within the time range
  productive += (end - startInflow) / 60000;
} else if (startInflow <= start && start <= end && end <= endOutflow) {             // If productive started before and ended later
  productive += timeBucketSize * 60;
}
```

```
 // Calculate percentages
 const totalTime = timeBucketSize * 60;
 productive = Math.round((productive / totalTime) * 100);
 setup = Math.round((setup / totalTime) * 100);
 changeover = Math.round((changeover / totalTime) * 100);

 // Calculate the idle time percentage
 const idle = 100 - productive - setup - changeover;

 return {
   day: end.getDay(),
   hour: end.getHours(),
   productive: productive,
   setup: setup,
   changeover: changeover,
   idle: idle,
 };

});
```

Figure E.1: Implementation of resource occupancy calculation

E.2. Effectiveness

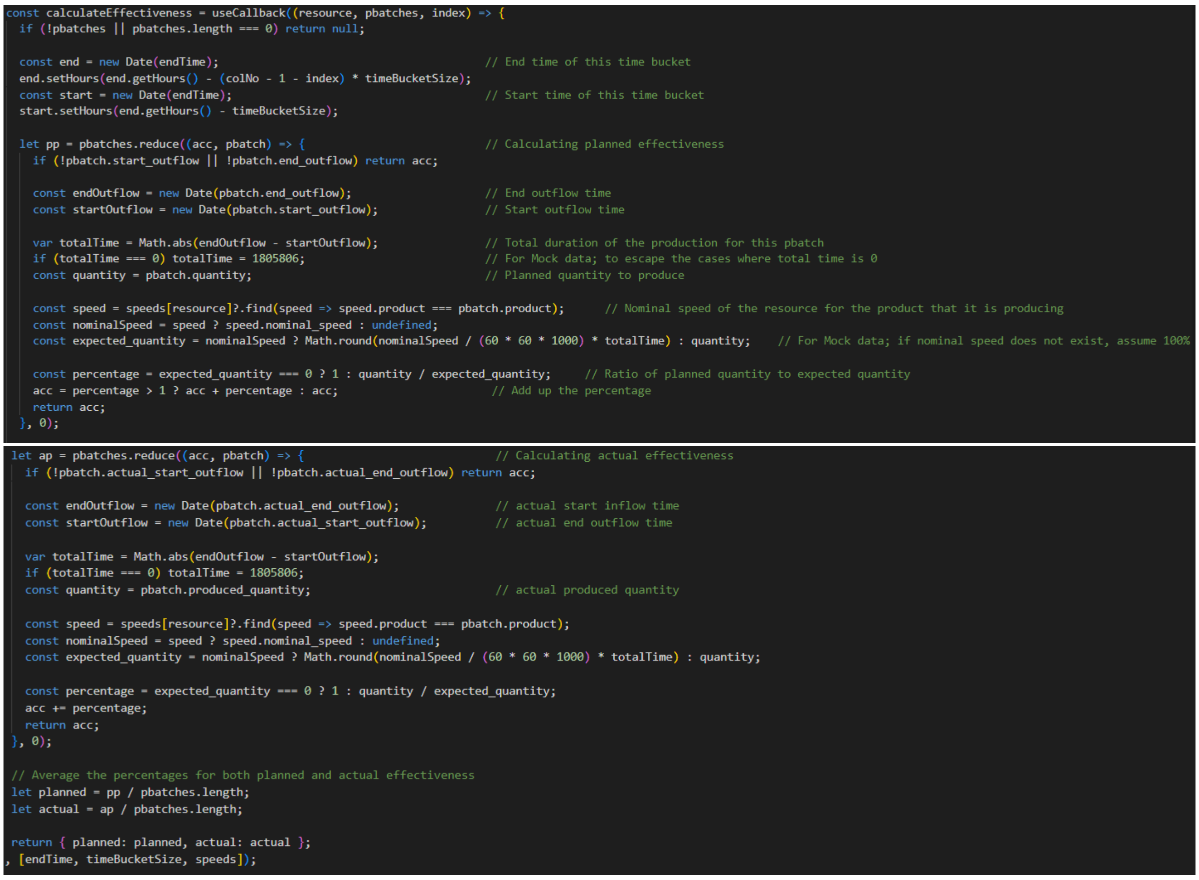
E.2.1. User Input Fields

1. Resource
2. Start time
3. Time bucket

E.2.2. Fetched Fields

1. pbatch.start_outflow
2. pbatch.end_outflow
3. pbatch.actual_start_outflow
4. pbatch.actual_end_outflow
5. pbatch.quantity
6. pbatch.produced_quantity
7. pbatch.product
8. nominalSpeed.nominal_speed

E.2.3. Calculation



```
const calculateEffectiveness = useCallback((resource, pbatches, index) => {
  if (!pbatches || pbatches.length === 0) return null;

  const end = new Date(endTime);                                    // End time of this time bucket
  end.setHours(end.getHours() - (colNo - 1 - index) * timeBucketSize);
  const start = new Date(endTime);                                  // Start time of this time bucket
  start.setHours(end.getHours() - timeBucketSize);

  let pp = pbatches.reduce((acc, pbatch) => {                       // Calculating planned effectiveness
    if (!pbatch.start_outflow || !pbatch.end_outflow) return acc;

    const endOutflow = new Date(pbatch.end_outflow);                // End outflow time
    const startOutflow = new Date(pbatch.start_outflow);            // Start outflow time

    var totalTime = Math.abs(endOutflow - startOutflow);            // Total duration of the production for this pbatch
    if (totalTime === 0) totalTime = 1805806;                       // For Mock data; to escape the cases where total time is 0
    const quantity = pbatch.quantity;                               // Planned quantity to produce

    const speed = speeds[resource]?.find(speed => speed.product === pbatch.product);      // Nominal speed of the resource for the product that it is producing
    const nominalSpeed = speed ? speed.nominal_speed : undefined;
    const expected_quantity = nominalSpeed ? Math.round(nominalSpeed / (60 * 60 * 1000) * totalTime) : quantity;    // For Mock data; if nominal speed does not exist, assume 100%

    const percentage = expected_quantity === 0 ? 1 : quantity / expected_quantity;     // Ratio of planned quantity to expected quantity
    acc = percentage > 1 ? acc + percentage : acc;                   // Add up the percentage
    return acc;
  }, 0);
 let ap = pbatches.reduce((acc, pbatch) => {                        // Calculating actual effectiveness
   if (!pbatch.actual_start_outflow || !pbatch.actual_end_outflow) return acc;

   const endOutflow = new Date(pbatch.actual_end_outflow);          // actual start inflow time
   const startOutflow = new Date(pbatch.actual_start_outflow);      // actual end outflow time

   var totalTime = Math.abs(endOutflow - startOutflow);
   if (totalTime === 0) totalTime = 1805806;
   const quantity = pbatch.produced_quantity;                       // actual produced quantity

   const speed = speeds[resource]?.find(speed => speed.product === pbatch.product);
   const nominalSpeed = speed ? speed.nominal_speed : undefined;
   const expected_quantity = nominalSpeed ? Math.round(nominalSpeed / (60 * 60 * 1000) * totalTime) : quantity;

   const percentage = expected_quantity === 0 ? 1 : quantity / expected_quantity;
   acc += percentage;
   return acc;
 }, 0);

 // Average the percentages for both planned and actual effectiveness
 let planned = pp / pbatches.length;
 let actual = ap / pbatches.length;

 return { planned: planned, actual: actual };
, [endTime, timeBucketSize, speeds]);
```

Figure E.2: Implementation of effectiveness calculation

E.3. Start Time Variance

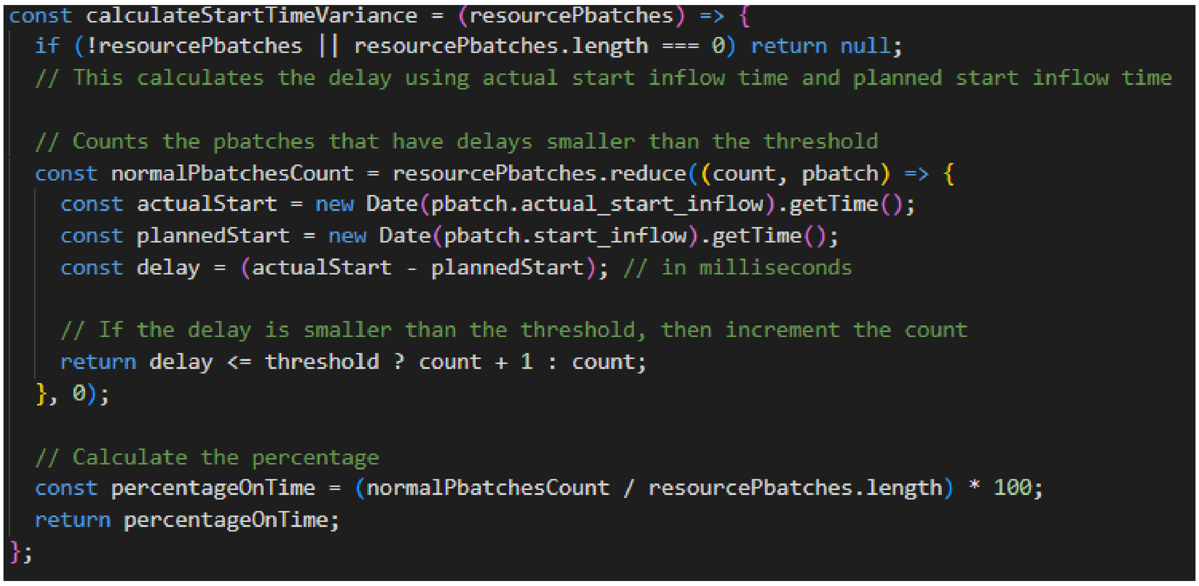
E.3.1. User Input Fields

1. Resource
2. Start time
3. Time bucket
4. Threshold

E.3.2. Fetched Fields

1. pbatch.start_inflow
2. pbatch.actual_start_inflow

E.3.3. Calculation



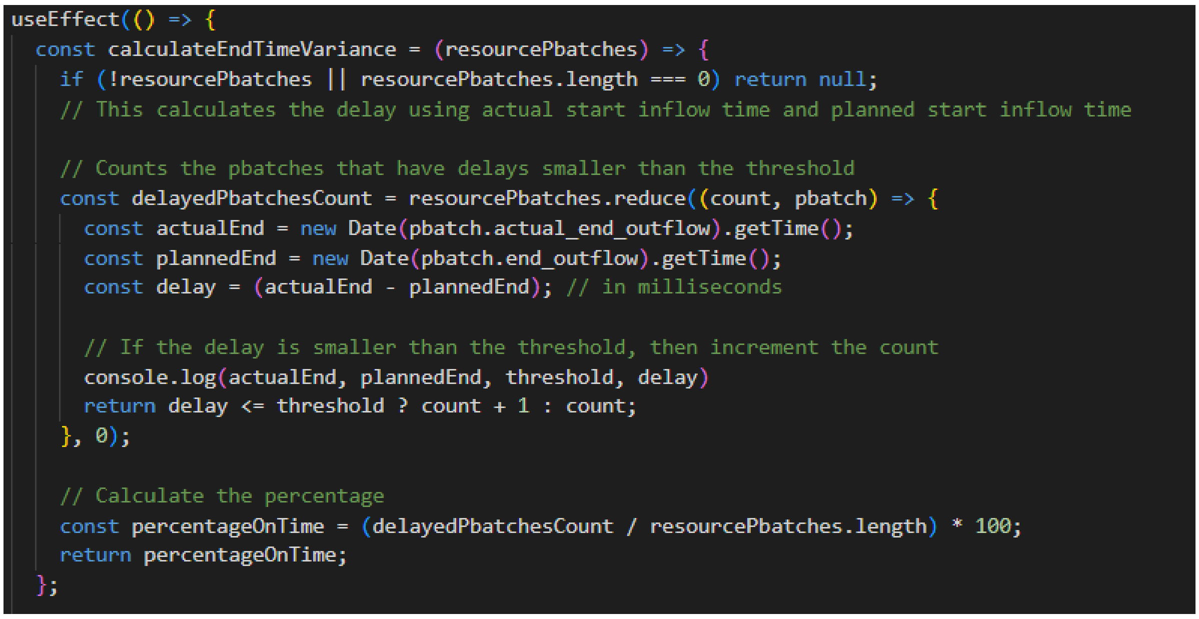
```
const calculateStartTimeVariance = (resourcePbatches) => {
  if (!resourcePbatches || resourcePbatches.length === 0) return null;
  // This calculates the delay using actual start inflow time and planned start inflow time

  // Counts the pbatches that have delays smaller than the threshold
  const normalPbatchesCount = resourcePbatches.reduce((count, pbatch) => {
    const actualStart = new Date(pbatch.actual_start_inflow).getTime();
    const plannedStart = new Date(pbatch.start_inflow).getTime();
    const delay = (actualStart - plannedStart); // in milliseconds

    // If the delay is smaller than the threshold, then increment the count
    return delay <= threshold ? count + 1 : count;
  }, 0);

  // Calculate the percentage
  const percentageOnTime = (normalPbatchesCount / resourcePbatches.length) * 100;
  return percentageOnTime;
};
```

Figure E.3: Implementation of start time variance calculation

E.4. End Time Variance

1. pbatch.end_outflow
2. pbatch.actual_end_outflow

E.4.1. Calculation



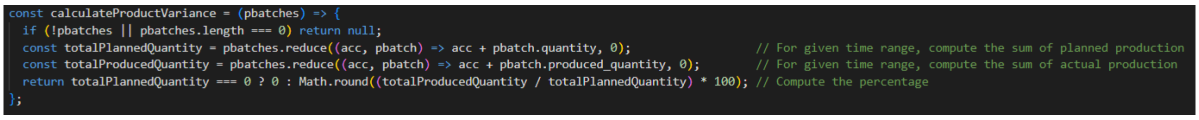
```
useEffect(() => {
  const calculateEndTimeVariance = (resourcePbatches) => {
    if (!resourcePbatches || resourcePbatches.length === 0) return null;
    // This calculates the delay using actual start inflow time and planned start inflow time

    // Counts the pbatches that have delays smaller than the threshold
    const delayedPbatchesCount = resourcePbatches.reduce((count, pbatch) => {
      const actualEnd = new Date(pbatch.actual_end_outflow).getTime();
      const plannedEnd = new Date(pbatch.end_outflow).getTime();
      const delay = (actualEnd - plannedEnd); // in milliseconds

      // If the delay is smaller than the threshold, then increment the count
      console.log(actualEnd, plannedEnd, threshold, delay)
      return delay <= threshold ? count + 1 : count;
    }, 0);

    // Calculate the percentage
    const percentageOnTime = (delayedPbatchesCount / resourcePbatches.length) * 100;
    return percentageOnTime;
  };
```

Figure E.4: Implementation of end time variance calculation

E.5. Product Variance

1. pbatch.quantity
2. pbatch.produced_quantity

E.5.1. Calculation



```
const calculateProductVariance = (pbatches) => {
  if (!pbatches || pbatches.length === 0) return null;
  const totalPlannedQuantity = pbatches.reduce((acc, pbatch) => acc + pbatch.quantity, 0);            // For given time range, compute the sum of planned production
  const totalProducedQuantity = pbatches.reduce((acc, pbatch) => acc + pbatch.produced_quantity, 0);   // For given time range, compute the sum of actual production
  return totalPlannedQuantity === 0 ? 0 : Math.round((totalProducedQuantity / totalPlannedQuantity) * 100); // Compute the percentage
};
```

Figure E.5: Implementation of product variance calculation

F

Test Results

Name	**Stmts**	**Miss**	**Cover (%)**
__init__.py	2	2	0
asgi.py	4	4	0
core/__init__.py	0	0	100
coreapp/__init__.py	0	0	100
coreapp/admin.py	1	0	100
coreapp/api_tests/__init__.py	0	0	100
coreapp/api_tests/test_api_component_effectiveness.py	91	5	95
coreapp/api_tests/test_api_component_occupancy.py	86	5	94
coreapp/api_tests/test_api_component_quantity_variance.py	85	2	98
coreapp/api_tests/test_api_dashboard.py	74	5	93
coreapp/api_tests/test_api_nominal_speed.py	46	3	93
coreapp/api_tests/test_api_order_view_set.py	48	3	94
coreapp/api_tests/test_api_pbatch_view_set.py	82	7	91
coreapp/api_tests/test_api_start_end.py	126	9	93
coreapp/apps.py	4	0	100
coreapp/migrations/0001_initial.py	12	0	100
coreapp/migrations/0002_pbatch_shiftpattern_trafficlight_and_more.py	6	0	100
coreapp/migrations/0003_remove_componentperfectorder_product_and_more.py	4	0	100
coreapp/migrations/0004_componentdeliverytime_is_live_and_more.py	4	0	100
coreapp/migrations/0005_componentdeliverytime_time_bucket_and_more.py	4	0	100
coreapp/migrations/0006_componentdeliverytime_estimated_delivery_time.py	4	0	100
coreapp/migrations/0007_componentend_componentproductionquantity_and_more.py	5	0	100
coreapp/migrations/0008_componentproductionquantityvariance_and_more.py	5	0	100
coreapp/migrations/0009_rename_actual_end_pbatch_actual_end_inflow_and_more.py	4	0	100
coreapp/migrations/__init__.py	0	0	100
coreapp/model.py	6	6	0
coreapp/serializers.py	1	1	0
coreapp/serializers/__init__.py	0	0	100
coreapp/settings.py	19	0	100
coreapp/settings_test.py	2	0	100
coreapp/tests/__init__.py	0	0	100
coreapp/tests/test_component.py	128	128	0
coreapp/tests/test_dashboard.py	38	38	0
coreapp/utils/__init__.py	0	0	100
coreapp/views/__init__.py	0	0	100
manage.py	11	2	82
models/__init__.py	0	0	100
models/asgi.py	4	4	0
models/core/__init__.py	7	0	100
models/core/auth.py	42	3	93
models/core/bills.py	51	34	33
models/core/bookmarks.py	29	29	0
models/core/components.py	99	5	95
models/core/dashboard.py	12	1	92
models/core/documents.py	53	53	0
models/core/execution_log.py	6	6	0
models/core/locations.py	17	2	88
models/core/notifications.py	115	48	58
models/core/orders.py	77	48	38
models/core/products.py	63	5	92
models/core/remark_group.py	15	3	80
models/core/remarks.py	78	25	68
models/core/resource.py	73	13	82
models/core/syncs.py	62	62	0
models/generic/__init__.py	0	0	100
models/generic/models.py	21	8	62
models/group_hierarchy/__init__.py	1	0	100
models/group_hierarchy/models.py	49	29	41
models/pbatch/__init__.py	3	0	100
models/pbatch/bookings.py	143	143	0
models/pbatch/documents.py	24	24	0
models/pbatch/inventory.py	72	26	64
models/pbatch/pbatches.py	197	108	45
models/pbatch/remarks.py	27	27	0
models/pbatch/shifts.py	72	29	60
models/pbatch/trafficlights.py	29	2	93
models/serializer.py	69	0	100
models/settings.py	19	19	0
models/urls.py	19	0	100
models/views.py	168	13	92
models/wsgi.py	4	4	0
wsgi.py	4	4	0
TOTAL	**2626**	**997**	**62%**

Table F.2: Coverage Report for backend

File	**% Stmts**	**% Branch**	**% Funcs**	**% Lines**	**Uncovered Line #s**
All files	62.47	50.75	56.72	61.89	
src	0	0	0	0	
App.js	0	0	0	0	18-103
index.js	0	100	100	0	9-21
jest.config.js	0	100	100	0	1
reportWebVitals.js	0	0	0	0	1-8
withNavigation.js	0	100	0	0	4-7
src/components/AddDataForm	0	0	0	0	
AddEditData.jsx	0	0	0	0	12-43
src/components/AddDataForm/Forms	61.43	41.66	56.66	62.41	
EffectivenessForm.jsx	93.61	61.76	85.71	95.34	44-45
EndTimeVarianceForm.jsx	0	0	0	0	12-183
InventoryForm.jsx	0	0	0	0	12-138
ProductVarianceForm.jsx	93.47	61.76	85.71	95.23	41-42
ResourceOccupancyForm.jsx	93.61	61.76	85.71	95.34	44-45
StartTimeVarianceForm.jsx	95	65.85	88.23	96.42	46-47
src/components/Button	100	100	100	100	
DeleteButton.jsx	100	100	100	100	
SaveButton.jsx	100	100	100	100	
src/components/DeleteConfirmation	100	100	100	100	
DeletePopup.jsx	100	100	100	100	
src/components/DialGauge	0	0	0	0	
DialGauge.jsx	0	0	0	0	13-228
index.js	0	0	0	0	
src/components/Dropdown	17.43	12.9	15.78	17.59	
ComponentDropdown.jsx	0	0	0	0	7-73
DashboardDropdown.jsx	0	0	0	0	17-157
ExportDropdown.jsx	73.07	66.66	75	73.07	39-49
src/components/TopBar	72.72	44.44	66.66	70	
TopBar.jsx	72.72	44.44	66.66	70	29-31, 47-49, 53-54, 72
src/components/Utils	100	80	100	100	
percentageTranslator.js	100	80	100	100	5-6, 19-20
src/store	0	100	100	0	
index.js	0	100	100	0	16-30
src/store/actions	93.51	82.5	95.65	93.79	
componentActions.js	86.2	77.63	90.32	85.48	36, 157-158, 174, 178, 183-209
dashboardActions.js	100	90	100	100	46, 105
pbatchEFActions.js	100	100	100	100	
pbatchETActions.js	100	100	100	100	
pbatchPVActions.js	100	100	100	100	
pbatchSTActions.js	100	100	100	100	
productsActions.js	100	100	100	100	
resourcesActions.js	100	100	100	100	
speedActions.js	93.75	80	100	100	11, 28
src/store/reducers	100	96.47	100	100	
componentReducer.js	100	90	100	100	13
dashboardReducer.js	100	90	100	100	17
pbatchEFReducer.js	100	100	100	100	
pbatchETReducer.js	100	100	100	100	
pbatchPVReducer.js	100	100	100	100	
pbatchROReducer.js	100	100	100	100	
pbatchSTReducer.js	100	100	100	100	
pbatchTimebucketReducer.js	100	100	100	100	
productsReducer.js	100	100	100	100	
resourcesReducer.js	100	80	100	100	13
speedReducer.js	100	100	100	100	

Table F.4: Coverage Report for frontend

G Project Plan

The following subsection outlines the project plan created at the outset of our project, based on our initial knowledge and research of the system we were to implement. It includes the problem analysis, feasibility study, risk analysis, requirements, and development methodology. Through feedback received on this initial plan, we have gained a better understanding of the project and refined our upcoming plans. This feedback has been significant in adapting our requirements and successfully implementing the final product.

—-

G.1. Problem Analysis

G.1.1. Problem Statement

The main aim of the project is to create a dashboard application that assists existing supply chain schedulers. Occator is a supply chain consultant that provides the overview of the planboard including schedules, inventory availability, orders, performances, and remarks to the clients. The key problems we aim to address is: How to effectively provide all necessary information to the client. This includes integrating and presenting a broad spectrum of data such as schedules, inventory availability, orders and performance metrics in an actionable format. Leveraging a balanced scorecard approach can aid in categorizing these performance metrics into dimensions including customers, internal processes, innovations, and finance, that provides overview of current scheduler to the clients[1]. Visualizing the performance of the current scheduler. The dashboard should avoid containing too much data. Also, instead of displaying all static data, the application has to provide more dynamic information, based on the importance and degree of impact on performance. Additionally, the dashboard must be adaptable and flexible enough to function effectively across different schedules. Creating a dashboard that is compatible with existing SCM (supply chain management) schedulers. As the performance from the provided scheduler has to be shown, the team has to decide which data to show and use to evaluate. The design of the dashboard should also align with the scheduler.

G.1.2. Problem Domain

There will be mainly two stakeholders in this problem - namely Occator, project manager, and project member. Occator is responsible for designing and implementing the solution, ensuring the solution meets the clients' needs and aligns with industry standards. The project manager is the client who requests service from Occator, and has the goal of achieving efficient production, scheduling and optimizing their supply operations. Their satisfaction with the solution is the most important factor for the success of Occator's consultancy services. Lastly we have the project members, who will be the end users of the application. They will have restricted availability to the application (e.g. read only) compared to project managers. Their input and feedback are essential in ensuring that the solution effectively addresses the specific problem and needs in the production process. Therefore, close collaboration and communication among both stakeholders are required to develop a firm solution that maximizes productivity while meeting the clients' requirements.

The proposed product is a comprehensive dashboard in order to address the diverse needs of the stakeholders within the process of supply chain in the process industry. It serves as a centralized hub for project overview, milestone tracking, progress updates and production metrics aggregation. Users can access real-time updates on production activities, including scheduled tasks, ongoing processes, and any detected violations such as inventory shortage or production delays. This facilitates swift issue detection and resolution, enhancing overall project efficiency and performance. Occator can leverage the dashboard not only to provide clients with project overviews but also to foster communication and collaboration among stakeholders, thereby aligning with the broader objective of improving resource utilization and productivity in the process industry.

G.1.3. Existing Solutions

An existing solution for the supply chain management dashboard is SAP SCM. SAP SCM provides comprehensive functionalities that allow management of all supply chain processes in a single platform. Its main feature is its centralized scheduling assistant which helps effectively manage resources, inventories, and orders: it allows users to visualize potential violations like inventory shortages, making risk management more accurate and timely [2]. Furthermore, SAP SCM provides interactive plan boards for direct manipulation of orders and schedules, making real-time scheduling changes simpler. As the main problem of the project is creating such an interactive application, many inspirations could be taken from SAP SCM.

Another existing solution for this project's requirement is Klipfolio. Klipfolio is a data visualization tool that can assist the role of a supply chain manager and scheduler. It offers robust, real-time tracking that can readily present critical supply chain metrics on a customizable dashboard. This aids in efficient planning, monitoring inventory levels, managing resources, and assessing performance. Moreover, it provides alerts for any constraints, such as inventory shortages, to prompt immediate action. The data visualization and constraint management functionalities could be benchmarked in this project to implement a more efficient and effective method in the solution.

G.2. Solution Outline

In order to address three key problems - 1) effective information delivery, 2) effective visualization, 3) compatibility with existing applications - the use of "GQM(Goal-Question-Measurement)"[3] model provides a structured and strategic idea. Now we will go through how GQM model mitigates each problem:

Figure 1 : GQM model (Janes, A. et al. (2013) Effective dashboard design, Cutter IT Journal.) For effective information delivery, as it is mentioned in the problem statement, integration and categorization is a key. By setting specific goals that information needs to be delivered to Occator's client, and then creating questions based on integrating and categorizing the information, the GQM model helps our data collection and planning to reflect what actual needs of the users. For effective visualization, GQM will help us to pinpoint crucial questions and filter unnecessary information. By focusing only on essential data, the dashboard remains streamlined and impactful, avoiding information overload and enhancing user decision-making through clear, concise visual representations. Lastly, to satisfy compatibility, it is important to fully understand the existing application's structure before we build our new dashboard. If we try to squeeze in what we want to the application, it will not be helpful for Occator and Occator's clients. Therefore, the GQM model is useful in addressing our key problems. However, it is crucial to avoid developing our own questions independently. Since the requirements are provided by Occator, our focus should remain strictly on these stipulations. By properly applying the GQM model based on Occator's requirements, we can enhance the clarity of communication between Occator and our team. This approach will help in aligning our objectives, ensuring a unified and coherent goal throughout the project.

G.3. Feasibility study & Risk analysis

G.3.1. Feasibility study

The feasibility study is a crucial step in assessing the viability and potential success of a project. This subsection will list a study based on different aspects of feasibility in order to determine whether the project is worth pursuing. We will focus on technical and schedule-wise aspects of feasibility. Assessing the technical feasibility involves evaluating whether the project can be successfully implemented considering the technological requirements and the team's expertise. In this context, the project requires proficiency in Django for backend development and React for frontend development, both of which are unfamiliar to the team. The schedule feasibility analysis focuses on determining whether the project can be completed within the allocated time frame of 10 weeks, encompassing all stages from initial client meetings to the delivery of the final product By conducting a thorough analysis of both technical and schedule feasibility, the project team can make informed decisions regarding the viability of pursuing the project within the given constraints. This study serves as a foundation for identifying potential challenges and implementing strategies to overcome them, ultimately increasing the likelihood of project success[4].

When looking at the solution outline we can assess the technical feasibility of each solution: Effective information delivery: Implementing effective information delivery requires knowledge about Django and React, which the team lacks. However, they are confident that this is feasible. Effective visualization as a dashboard: Creating a visually appealing and functional dashboard takes expertise in React and data visualization. Since the team's application extends from Occator's application, we need to follow their design. This means there is less room for creativity, but this does improve feasibility since we have an example design to take inspiration from. Compatibility with existing application: Ensuring compatibility with Occator's existing application involves understanding its architecture, data structure and integration points. This requires technical expertise in backend development and API integration. Since the team is still unfamiliar with the backend of the existing application, they may face challenges to easily integrate their application. Thorough documentation and communication with Occator is essential to minimize compatibility issues early on during development and will pose a big problem if not done correctly.

We are now going to consider the schedule feasibility of each solution: Effective information delivery: The process of categorizing, integrating and delivering information requires careful planning and communication with Occator. During the 10 weeks of the project we should allocate enough time each week to show Occator what and how we deliver information to its clients. Information is only effective when it resonates with the client. Effective visualization as a dashboard: Building a good dashboard requires iterative design and development cycles. Since the dashboard should follow the design of the existing application, making drafts of the dashboard and getting it approved by Occator should be done early on so that both parties have the same vision of the end product. During the project we should have regular meetings regarding the design to ensure the current product is still aligned with their vision and make changes accordingly if needed. Compatibility with existing application: Ensuring compatibility during early development is crucial for the schedule feasibility of the project, if this is not done correctly there is a possibility that we might be pressed for time.

Regarding the feasibility of this project, the team definitely thinks the project is viable within the given time constraints. Taking these potential challenges into account during the project and tackling them accordingly should bring us closer to a successful project. 3.2. Risk analysis One of the keys to succeeding in creating a successful product is considering all potential risks that come into play, before, during and after the development of DASH+. These are the risks the team considered:

The team does not have access to actual company data. When the team is developing functionalities, the team can only test them using testing data. Therefore, it will not be clear if the application works well and efficiently in real-life settings with live data. This could be minimized by making use of testing data that already exists in the company because their simulated data closely resembles the actual dataset. The team could test the application by either using the simulated data directly or creating a testing data based on it. The team does not always have access to the source code of the company. It is only available on devices from the company itself. When it is required, the team should be at the office. The application requires to be built upon frameworks that are not well-known by the team; while the company can provide assistance with this, there may still be challenges during the development process. The client's expectation might not be aligned with the use of the application. The risk of this happening is low, since the team will be talking to the client and keeping them updated to any changes throughout the development process using various methods such as prototyping. The team could unintentionally share confidential information externally. The team could minimize this by communicating any project-relevant information through Microsoft Teams and not uploading or sharing the progress outside the client and TU Delft. The client's delay in providing requirements may hinder the team from commencing actual implementation of the application.

G.4. Requirements

This section lists the functional and non-functional requirements of the application. The requirements will be organized using the MoSCoW method, as it is a highly scalable method and is especially effective in the earlier stage of the project. The “Must have” requirements are essential features and functionalities without which the application cannot fulfill its primary purpose. The “Should have” requirements are important but not critical functionalities that enhance the application’s usability and effectiveness. While the MoSCoW method typically includes ’Could have’ and ’Won’t have’ sections, our requirement list does not feature them. This is because the client has provided us with a definitive list of requirements that are essential. Lastly, the “non-functional” requirements encompass various aspects including the selection of the framework to be used, the organization of the application’s architecture, and guidelines for database usage. These aspects are critical for ensuring the efficiency, maintainability, and scalability of the application beyond its functional features.

“Must-have” requirements: User can see the dashboard, when the user clicks the dashboard tab User can select the main (default) plan and switch between plans User can add new graphs in a plan User can see a dropdown list of available graphs

Admin can login to an admin panel Admin can enter the ‘Dash’ subsection in the admin panel Admin can see the list of graphs that are available to the user Admin can add or edit (title, description, axis, color, etc) graphs, which will be displayed on the dashboard for the users to add their plans

The application must show resource occupancy in a selected time bucket with percentage values for: productive non-productive (maintenance and downtime) setup changeover idle The application must have a filter to filter plans per product, resource, or custom made selection The application must show calculated KPI information following a predefined formula The application must support the admin to modify and configure the formula that calculates the KPI information The application should forecast coverage

“Should-have” requirements: plans can be themed (effectiveness plan, adherence plan, quality plan) User can import data(txt, xml, pdf, etc) from external source User can compare imported actual produced quantity with planned produced quantity The application must show information about the effectiveness of the plan (regarding production speed and machine/operating efficiency) The application must report on the idle time of a machine The application must show information for adherence if the plan is producing what should be produced and when it should be produced ?Create information for Adherence The application must display information about the quality if the plan is producing the specifications The application must make the results exportable to Excel and PDF

“Non functional” requirements: The frontend development should be done on JavaScript (React) and Redux The frontend development should use global variables provided if possible The frontend development should organize the project files in a way similar to OccaSee

The backend development should be done using Python (Django) The backend development should organize the project files in a way similar to OccaSee The backend development should use the provided mocked database, while keeping the original Models unchanged unless necessary

G.5. Development Methodology

In this section, the general methodology used in this project to obtain the final product will be discussed. The method that will be used throughout the development is the Scrum method, which is a kind of Agile development methodology. Scrum, an agile software development framework, emphasizes adaptability to changing requirements, known as “requirements volatility”. Unlike traditional forecasting methods, Scrum allows for possible changes in customer requirements during development. This approach recognizes the uncertainty of pre-defining the problem thus it prioritizes responses to evolving needs, technological developments, and market developments[6]. This approach allows the team to plan each sprint per week, including the iterative development cycles, frequent collaboration, product increment, and feedback for those specifically from the client. To review the sprint, a discussion with the client is scheduled for every Monday with the flexibility of adjusting by some special circumstances. By applying this methodology, a flexible and adaptable development environment is expected. The team will use tools such as GitLab, provided by TU Delft, for version control and project management, enabling seamless collaboration and tracking of progress. As the importance of the definition of done came to mind, some criteria for the completion of features and merge requests to ensure consistency and quality throughout development were established. Each task will be considered done when it meets predefined acceptance criteria, passes the code review, and has been tested successfully. For that, several criteria are discussed as follows: Each team member initially on their branch, and merge later At least 2 people approve the merge request All merge requests should be done in the ‘dev’ branch first If all members approve, merge the ‘dev’ branch to the ‘main’ Before merging to the ‘dev’ branch, the frontend part should use the ‘frontend’ branch, and the backend part should use the ‘backend’ branch. The ‘frontend’ branch should be merged with at least 1 approval The ‘backend’ branch should be merged with at least 1 approval

Other tools are suggested to use for communications too. For important communications, including confidential ones, the communication will be done on Microsoft Teams; this is specifically for the client. For contact with the TA, Mattermost will be used while the others will be done via email, WhatsApp, and Discord. Every Tuesday, there will be a meeting with the TA to discuss the progress and consult any problems the team experiences with the project management. Coach meetings will commence when the team needs feedback or advice regarding deliverables along the course. All meetings will have chairs and notetakers, following the Scrum methodology with a prepared agenda.

If the demand for changes is requested, whether or not to make changes will be decided during the sprint planning session with the client. Each change will be evaluated based on its alignment with project goals, impact on existing features, and feasibility within the project timeline.

For an efficient development process, the team distributed the work to each member. First of all, 3 of the team members will primarily focus on the frontend of the application, while the other 2 members are developing the backend and the database. The frontend development will be based on React, and the backend development will be based on Python Django Web Framework. As the project progresses, team members will share responsibility for all parts of the project. To ensure an effective and efficient team working process, the team agrees to follow a code of conduct. To sum it up, the team established shared values and norms to guide their collaboration and project development process. The team’s shared values prioritize fairness, respect, transparency, and clear communication. The team aims for a minimum grade of 8.0 and commits to delivering a high-quality dashboard application for Occator. For communication and collaboration, Microsoft Teams, Mattermost, and GitLab will be used. By adopting a SCRUM approach for project management and decision-making, make weekly client and TA meetings to track progress and resolve issues. Roles within the team, such as chairperson and note-taker, rotate weekly to ensure equal participation. The team emphasizes open communication, mutual respect, and conflict resolution, with a focus on prevention and support. Expect advice from the TA on project organization and feedback on deliverables. Success factors include balanced team composition, strong communication, active participation, shared vision, and diversity of backgrounds and experiences. Norms for evaluation include task completion, respectful behavior, adherence to requirements, clear documentation, teamwork, doing everything on time, and communication of significant changes. The original code of conduct containing details is shared among group members and is accessible to all.

G.6. Bibliography

[1] J. P. C. Kleijnen and M. Smits, “Performance metrics in supply chain management,” Journal of the Operational Research Society, vol. 54, no. 5, pp. 507–514, May 2003, doi: 10.1057/palgrave.jors.2601539

[2] G. Knolmayer, P. Mertens, and A. Zeier, Supply chain management based on SAP systems: Order management in manufacturing companies. 2002. [Online]. Available: http://ci.nii.ac.jp/ncid/BA55180663

[3] A. Janes, A. Sillitti, G. Succi, FHV Vorarlberg University of Applied Sciences, and Centre for Applied Software Engineering, “Effective dashboard design,” Jan. 2013. [Online]. Available: https://www.researchgate.net/profile/Alberto-Sillitti/publication/286996830$_{E}ffective_{d}ashboard_{d}esign/links/57c699e208aec24de0414df1/Effective-dashboard-design.pdf$

[4] D. J. Bowen et al., “How we design feasibility studies,” American Journal of Preventive Medicine, vol. 36, no. 5, pp. 452–457, May 2009, doi: 10.1016/j.amepre.2009.02.002

[5] Vestola, M., A Comparison of Nine Basic Techniques for Requirements Prioritization. rep. 2010

[6] Sachdeva, S., Scrum Methodology. Int. J. Eng. Comput. Sci, 5(16792), 16792-16800, 2016

H
Work distribution

Justin Jo

- Creating and implementing frontend logic for data calculation in different components.
- Designing and creating exportable components.
- Designing and managing frontend architecture.
- Creating Redux store and reducer logic.
- Implementing time bucket calculation.
- Implementing graph visualization.

Michel Chen

- Implementing and reformatting code in the frontend.
- Implementing API calls on the frontend.
- Refactoring and styling frontend components.
- Testing frontend code.
- Adding content in the wiki.

Kyongsu Kim

- Creating and implementing frontend components with data calculation.
- Creating data visualizations (Graphs, Table with bar chart).
- Implementing API calls on the frontend (components, dashboard, pbatch for some components, resources).
- Switch dashboard.
- Deleting components.
- Bug fixes in the frontend (data overwriting for multiple data fetch, some data not fetching correct data).

Junwon Yoon

- Implementing, testing, and reformatting the classes from the backend.
- Setting up API endpoints for different classes.
- Database management (setup, data addition, query fixation, etc).
- Adding content to the README file.
- Adding some features and bug fixes in the frontend.

Kwangjin Lee

- Implementing, testing, and reformatting the classes from the backend.
- Setting up API endpoints for different classes.
- Database management (setup, data addition, query fixation, etc).
- Bug fixes in the frontend.

- Designing some parts of the frontend.
- Creating queries for API endpoints.