Jaakko Jokinen

Design and Development of a Task Data Collection and Visualization Software: A Case Study at Posti Oyj

School of Technology and Innovations
Bachelor's Thesis
Automation and Computer Science

Vaasa 2026

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen akateeminen yksikkö
Tekijä: Jaakko Jokinen
Tutkielman nimi: Design and Development of a Task Data Collection and Visualization Software: A Case Study at Posti Oyj
Tutkinto: Tekniikan kandidaatti
Koulutusohjelma: Energia- ja informaatiotekniikka
Opintosuunta: Automaatio ja tietotekniikka
Työn ohjaaja: Timo Mantere
Valmistumisvuosi: 2026 **Sivumäärä:** 51

TIIVISTELMÄ:
Tämä työ tutkii ohjelmistoprototyypin suunnittelua ja kehitystä Posti Oyj:n sisälogistiikka liiketoiminnassa liittyen työtehtävien datan keräykseen ja visualisointiin. Työ keskittyy käytännölliseen ongelmaan varastoinnissa, jossa joitakin työtehtäviä ei saada tarkasti seurattua nykyisillä järjestelmillä. Tämä rajoittaa tarkan datan saatavuutta ja näin heikentää resurssien käytön suunnittelua ja operatiivista päätöksentekoa.

Työn tarkoituksena on suunnitella ja kehittää Posti Oyj:n sisälogistiikka liiketoiminnalle työtehtävien datan keräykseen ja visualisointiin tarkoitettu ohjelmistoprototyyppi, joka mahdollistaa näiden työtehtävien oikeanlaisen kirjauksen sekä niistä saadun datan tutkimisen raporteissa. Työ suoritetaan tapaustutkimuksena missä yhdistetään tieteellistä kirjallisuutta, Posti Oyj:n tarpeet sekä käytännöllisen ohjelmiston kehittäminen. Teoreettinen taustatutkimus kattaa työssä tarvittavia käsitteitä liittyen varastoaktiviteettien seurantaan, verkkoapplikaatio arkkitehtuuriin, käyttöliittymän suunnitteluun, tietoturvallisuuteen ja datan visualisointiin. Näitä käsitteitä käytetään tukemaan prototyypin suunnittelua.

Ohjelmisto kehitetään verkkopohjaisena prototyyppinä, erottaen käyttöliittymän (frontend), palvelinpuolen (backend) ja tietokannan (MySQL). Prototyyppi sisältää roolipohjaiset näkymät työntekijöille, managereille ja ylläpitäjille, jotta jokainen käyttäjäryhmä voi suorittaa heille tarpeellisia toimia. Palvelinpuoli vastaa turvallisesta kirjautumisesta, roolipohjaisesta oikeuksien valtuutuksesta, validoinnista, liiketoimintasäännöistä sekä tietokantayhteyksistä.

Prototyyppiä arvioidaan vaatimusten täyttymisellä sekä käyttäjätestauksella oikeassa Posti Oyj:n varastoympäristössä. Testauksen aikana varastotyöntekijät käyttävät prototyyppiä oikeissa työtilanteissa. Arviointi osoittaa, että prototyyppi tukee systemaattista datan keruuta ja antaa hyödyllistä dataa raportointiin ja operatiiviseen päätöksentekoon.

Työ osoittaa, että tunnistettu tarve tarkemmalle varastotyötehtävien seuraamiselle pystytään täyttämään digitaalisella ratkaisulla. Prototyyppi tarjoaa käytännöllisen alustan jatkokehitykselle, jatkotestaukselle sekä tuotantokäytölle.

AVAINSANAT: Warehousing, Logistics, Data collection, Data visualization, Resource management, Web application, Role-based, Prototype

Contents

Figures

Tables

Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Deployment
ECR	Elastic Container Registry
ECS	Elastic Container Service
ERD	Entity Relationship Diagram
HR	Human Resources
IoT	Internet of Things
JWT	JSON Web Token
MFA	Multi-Factor Authentication
MPA	Multi Page Application
NFC	Near Field Communication
RDS	Relational Database Service
REST	Representational State Transfer
SEO	Search Engine Optimization
SPA	Single Page Application
SQL	Structured Query Language
UAT	User Acceptance Testing
UI	User Interface
UX	User Experience
WMS	Warehouse Management System

1 Introduction

1.1 Background and objectives

Global trade is strongly dependent on the logistics field that enables the movement of goods around the world. The logistics sector contains many different activities such as warehousing, transportation, inventory management and order processing. (Verma et al., 2023) states that in the supply chain, warehousing and its proper management are the key to ensuring an uninterrupted flow of products. Verma introduces critical factors affecting warehouse operations. Some of them are manpower, warehouse productivity and forecasting (Verma et al., 2023).

Recent studies highlight the importance of data in developing warehousing. Accurate and relevant information of warehouse activities enables improved resource management, analyzing efficiency and operational decision-making (Baralis, 2024; Kellermayr-Scheucher et al., 2023). Simultaneously the logistics sector is transforming into the Industry 5.0 model that emphasizes real-time data collection, analytics and the collaboration between technology and human labor (Nagy et al., 2023).

This study is focused on Posti Oyj's warehousing, where all the warehousing activities can't be currently tracked or measured properly. Some of the activities are completely untracked and no data is collected from them. This complicates resource management, workforce efficiency analysis and forecasting warehousing trends. In the current state, decision-making is completely based on estimates and not accurate data.

The goal of this study is to design and implement a software solution for collecting and visualizing data related to warehousing activities. The aim of this solution is to generate more accurate information about the duration and quantities of these activities, and to support resource management, analytics and operational decision-making.

1.2 Research problem and questions

The main research problem of this study relates to the fact that some of the warehousing activities can't be reliably tracked with current systems. This leads to inadequate data that weakens resource management, analytics and operational decision-making.

The study aims to solve this problem by developing a software prototype that enables systematic data collection and visualization of these activities.

The research questions in this study are:

1. What kind of functional and non-functional requirements does a warehousing activity tracking system need?
2. What kind of design principles and technical solutions are best fitted for the implementation of this kind of software system?
3. How well does the implemented prototype solve the proposed problem and support resource management and analysis?

1.3 Scope and delimitations

The scope of the study includes the design and implementation of a software system for the needs of Posti Oyj. In this study, a functional prototype is developed that demonstrates the collection and visualization of data in a warehouse. The prototype is intended to be ready for testing in a controlled environment.

The study uses scientific research and articles about the best principles of software development, UI/UX design, system architecture and visualizing data. Based on this, the design plan for the software system is created.

The delimitations of this study include the further push to production, full deployment and measurement of long-term impact.

1.4 Research methods and materials

This study is conducted as a case study that examines the design and implementation of a software system for Posti Oyj's warehousing. The study uses a design-based approach where prior scientific research and practical requirements are combined.

The research process consists of four main stages. The first stage consists of reviewing prior literature about software development, UI/UX design, system architecture and data visualization. This forms the theoretical background for designing the software system.

In the second stage, the system requirements are defined based on the needs of the case organization and different user groups. These include the functional and non-functional requirements.

The third stage is based on the design and implementation of the software prototype that is based on the requirements and principles gathered from the literature review.

In the fourth stage, the implemented solution for the original problem is evaluated by comparing it to the given requirements and assessing its ability to support resource management, analytics and operational decision-making.

Materials used in the study consist of scientific literature, requirements obtained from the case organization and the implemented prototype.

1.5 Structure of the study

The study consists of seven main chapters. The introduction presents the background of the study, the research problem and questions, its scope, goal and research methods. The second chapter presents the theoretical background that covers the key principles of software development, UI/UX design, system architecture and data visualization. The third chapter introduces the case organization and defines the requirements for the software. The fourth chapter describes the design and architectural choices of the software. The fifth chapter consists of the implementation of the software. The sixth chapter evaluates the completed solution and its results. The seventh chapter presents the conclusions.

2 Theoretical background

2.1 Warehouse activity tracking and resource planning

Warehousing is a key part of a supply chain because it enables efficient storing, handling and moving of materials between different processes. The effectiveness of warehousing directly affects the performance, costs and service level of the supply chain (Verma et al., 2023).

Warehousing consists of many different activities such as picking, packing, loading, unloading and internal logistics. The time and resources needed for these activities vary, which makes tracking them crucial for effective performance. Recent studies show that optimizing warehousing activities requires precise and trustworthy data from operational processes (Almataani & Ullah, 2024).

Resource management is mostly based on historical and real-time data. Without precise information about the time and quantity of warehousing activities, organizations are forced to rely on estimates which weaken the quality of operational decision-making. Recent studies indicate that data-driven planning improves efficiency, lowers costs and can generate competitive advantage (Verma et al., 2023).

Simultaneously the logistics industry is transforming into even more digital operating models where real-time data collection and analytics are highlighted. Technologies such as WMS, IoT and automation enable improved insights into warehousing operations and support more efficient resource management (Alonge et al., 2023).

2.2 Web application architecture for internal systems

Modern web application architecture can be implemented in many different ways depending on the purpose, user quantities and technical requirements. Commonly used principles are, for example, multi-tiered architectures where the UI (frontend), system

logic (backend) and database are separated from each other, as well as simpler solutions where these parts are more closely connected to each other. This kind of modular architecture enhances the systems maintainability, scalability and security (Akpe et al., 2022).

In web applications the frontend is responsible for presenting information to the user and handling the inputs of the user. The backend is usually responsible for the system logic, handling requests, validation and the connections to other parts of the application. The database serves as a storage for data, where data can be fetched from and stored in. Even though these different parts of the application can be implemented in different ways, they are often seen as different logical layers that each have their own responsibilities. Studies show that this kind of differentiating of responsibilities improves the clarity of the system as well as eases its development and maintainability, especially as the system grows (Akpe et al., 2022).

The relationship between these layers can vary from system to system. Typically, the frontend communicates with the backend that handles the information and passes it onto the database. This prevents the database from being accessed directly from the frontend, which strengthens the security and maintainability of the system (Akpe et al., 2022). In some architectures the application can have extra layers such as API layers, integration services and microservices. The key principle in all this is that the different parts of the application communicate between each other by clearly defined interfaces.

The communication between the frontend and the backend can be implemented in many different ways, like by using RESTful APIs, GraphQL or other principles that offer a standardized way to handle requests and transfer data between the different parts of the system. The choice is often influenced by the complexity of the system, the needs for data querying and the community around the technology (Hernández et al., 2020). Also, communication between the backend and the database is a key part of the functionality of the system. The backend often communicates with the database, for example, with SQL queries or through different kinds of database libraries. The communication

method affects performance and maintainability of the system. Studies show that especially the structure of the queries and the principles of handling data are very important to the performance of the system (Shao et al., 2020).

Applications can be designed either to be multi page applications (MPA) or single page applications (SPA), where both of them have their advantages and disadvantages. MPA can be simpler to implement and control. They also give better opportunities for SEO. SPA is a common solution in applications where the user makes a lot of consecutive actions. SPA minimizes reloads of the page and enables a more fluent user experience (Phan, 2024). In internal systems SEO isn't a key requirement which makes the SPA approach a good choice. This way the focus is on performance and usability. Table 1 compares the MPA and SPA approaches from the view of an internal operational system.

Table 1. Comparison of MPA and SPA

Aspect	MPA	SPA
Page loading	Often loads a new page after user actions	Updates the page dynamically without reloading the whole page
User experience	Can feel slower in systems where the user makes many consecutive actions	Smooth and easier interactions
Search engine optimization	Fits for public pages where SEO is important	SEO isn't the primary focus of SPA
Complexity of implementation	Can be simpler to implement and manage	Requires more frontend logic and state control
Fit for this prototype	Weak fit because the user make consecutive actions and SEO isn't needed.	Strong fit because the system is internal and requires quick actions by the users

2.3 UI/UX design for role-based operational systems

User Interface and User Experience (UI/UX) design is a key part in the usability and effectiveness of a system, especially in operational systems where users perform repetitive activities. User-centered design emphasizes tailoring the software to the needs of the users, work environment and use cases (Granić, 2017).

In operational systems, users can usually be divided into different roles that have different goals and use cases. Recent studies show that the UI should support effective execution of activities as well as inspecting and analyzing data in different use cases (Punchoojit & Hongwarittorrn, 2017).

In addition, when designing the UI, different devices and use cases must be taken into consideration. Mobile and desktop usage sets different requirements for example for the speed of actions, the broadness of different views and displaying information. Key qualities of good UI are clarity, consistency and immediate feedback to the user. With these qualities, it is possible to minimize mistakes, improve efficiency and gain user satisfaction (Granić, 2017).

Different kinds of frameworks and libraries are often used when implementing the UI. Their purpose is to speed up development, support unified structures and make creating dynamic UIs easier. Recent studies highlight that when comparing these frameworks and libraries, usability, performance, scalability, trustworthiness, quality of documentation and support of the community should be taken into consideration. These factors affect directly to the quality of the development process and the final application (Kaur & Tiwari, 2023).

Studies also show that there isn’t a universally optimal solution for choosing frontend technologies. The right approach is based on, for example, the scale of the application, complexity, maintainability requirements as well as what kinds of use cases and users must the application support. This is why UI solutions are also important to think about

not only from the technical perspective but also from the usability and context perspective (Kaur & Tiwari, 2023).

2.4 Security and access control in enterprise web applications

Security is a central part of an organization-level software system, especially when operational and user-based data is being handled. Security is based on multiple sectors where the most important are authentication and authorization. Authentication verifies the identity of the user, whereas authorization defines what actions and information the user has access to (Pant et al., 2022).

Different kinds of authentication and authorization methods have their advantages and disadvantages. Password based authentication is easy to implement but is prone to weak passwords. MFA increases data security through added steps but can weaken the user experience. Token based authentication methods such as JWT support modern systems but require careful handling. In authorization, the role-based model is clear and very common but sometimes not flexible enough in complex systems. Choosing the right authentication and authorization methods is affected by the security requirements, usability and the sensitivity of the data (Pant et al., 2022).

Security isn’t only based on authentication and authorization but consists of many layers that support each other. Securing the dataflow, session management and safe handling of tokens are key principles in the overall safety of a system. Also input validation is important, because insufficient validation can lead to, for example, injection attacks and data breaches (Khan et al., 2022).

The system can also be implemented with security mechanisms such as request rate limiting and logging that help prevent misuse and detect anomalies. Studies show that security of the system should be considered throughout the whole development cycle because choices made in different points of the development cycle directly affect the security of the final system (Khan et al., 2022).

2.5 Data visualization and reporting in operational decision-making

Data visualization plays a major role in operational decision-making. It enables showing complex datasets clearly and supports daily activities of an organization. Visualization is especially used in production monitoring, performance tracking and controlling different processes. Furthermore, visualization can help visualize dependencies, follow activities and support users' decision-making in different operational environments. Interactive visualizations have become more important, because with them, the user is capable of inspecting data flexibly and form a overall picture of the state of the operations more effectively (Lindner et al., 2025).

The importance of visualization has also grown because it can improve the clarity and trustworthiness of data. Research shows that visualizations can help reduce opinion based decision-making and with this to support better economic decisions. This is why visualizations are a central part of modern reporting solutions in operational decision-making (Lindner et al., 2025).

Data can be visualized in many different ways, depending on what kind of data is wanted to be shown and how the user should interpret it. Common visualization formats are, for example, charts, graphs and maps that allow raw data to be transformed into more understandable forms. The meaning of visualization isn't only to show data, but to help the user to see relevant observations and form important insights for decision-making. Visual aspects like color, shape, location, width, length, height and size can emphasize meanings of the data and make it clearer. This is why the choices of visualization and visual aspects affect how easily the user can analyze data (Shakeel et al., 2022).

Reporting is an important part of organizational decision-making and business intelligence systems because it helps with collecting, analyzing and using data to support decision-making. Research shows that the effective usage of business intelligence systems is connected to how strongly different systems are part of the decision-making of the

organization. Well managed reporting enhances the accessibility of data, supports operational actions and helps organizations to take even more advantage of data in decision-making (Widjaja et al., 2025).

2.6 Similar systems and previous implementations

Previous research and implementations show that warehouse management, workforce planning and task tracking are increasingly tied to digital data collection and decision-making based on that. Kellermayr-Scheucher et al. discuss warehouse workforce management systems and address that workforce planning is an important part of warehousing, especially when demand and workforce need vary. Research shows that many organizations are still using spreadsheet tools like Excel, even though the ideal system should note the workers productivity, qualifications, forecasts, automatic planning and bottlenecks. This suggests that the warehousing business is in need of systems that are better at supporting operational planning (Kellermayr-Scheucher et al., 2023).

Operational planning based on data shows also in Baralis' study examining incoming volumes and workforce forecasting in a logistics warehouse. The study emphasizes that the variation of incoming volumes affects directly to workload, workforce needs and the operational effectiveness of the warehouse. Baralis also shows that automatic data collection can reduce manual updating of data, and the risk of human errors and insufficiencies. For this study, this supports the idea that tracking warehouse activities can improve later reporting, resource management and analytics (Baralis, 2024).

The study of a similar system by Guedes is a relevant point of comparison, because it examines the implementation of a digital system for tracking the warehouse workers' activities in a logistics center. Previous way of working was based on monitoring sheets filled out at the end of the day. This caused tracking based on estimates, transcription errors, administrative burden and weak real-time visibility. The digital solution took advantage of the existing infrastructure, for example, NFC-cards, end terminals and task reporting forms. Results of the prototype indicated better data collection, decreasing

manual work and real-time visibility. Guedes identified one area on development to be the accessibility of task logging. The workers shouldn’t have to walk long distances to the end terminals in the warehouse to start/stop a task. A solution proposal for this was either to add more terminals or integrate the system into the existing mobile devices of the workers (Guedes, 2025).

3 Case context and requirements

3.1 Case organization and current situation

This study was commissioned by Posti Oyj, the leading postal and logistics provider in Finland. Posti Oyj is under constant pressure to modernize its operations and to utilize digital technology (Singh, 2025). Outsourced in-house logistics is one of the sectors that Posti Oyj has had challenges in gathering accurate data to assist daily management. Currently, several logistics-related tasks cannot be tracked or measured because there are no corresponding transaction types in any system. As a result, there is no reliable way to create performance metrics or gain an overall understanding of the workload and resource usage related to these tasks. The customers of Posti Oyj have expressed a clear need for improved insights and more accurate ways of tracking in-house logistics activities. A tailored software solution is needed to solve these challenges. This solution is expected to enhance Posti Oyj's competitiveness within the logistics sector.

3.2 Stakeholders

Identifying the key users and stakeholders is crucial to tailoring the requirements to fulfill expected needs of everyone interacting with or affected by the software (Zahran & Widyarto, 2025). The primary stakeholders of this software include warehouse workers, managers and admins. The workers will use the software daily to record tasks. Managers will also use the software daily to analyze and inspect reports and work sessions. Admins will use the software regularly to maintain the system. Secondary stakeholders include logistics and HR departments of Posti Oyj who will use the collected data to analyze the in-house logistics resourcing and efficiency. Technical stakeholders include the developer of this study and the IT department of Posti Oyj. The developer is responsible for the design and development of the prototype. The IT department is responsible for supporting the development process, defining technical requirements and guiding their imple-

mentation, as well as the further integration of the software and the final push to production. Tertiary stakeholders include the customers and supply chain partners of Posti Oyj who will indirectly benefit from the improved logistics efficiency and service quality.

3.3 Functional requirements

The functional requirements define what the system should do and what kind of actions the users should be able to perform. Defining the requirements of the system is a critical part of designing it because it helps to make sure that all the key stakeholders and developers have a common understanding of the goals of the system, functionalities and limitations (Zahran & Widyarto, 2025). Also, according to Zahran and Widyarto, the requirement specification of the system works as the base of the whole software project (Zahran & Widyarto, 2025).

In this system, the functional requirements are based on three different user roles: workers, managers and admins. The workers log tasks, managers inspect and analyze collected data and admins maintain the different attributes in the system. Based on these roles, the functional requirements of this system are identified and are listed in Table 2 below.

Table 2. Functional requirements.

FUNCTIONAL REQUIREMENTS		
ID	USER	NEED
FR-01	All users	All users must be able to log in/out and authenticate
FR-02	All users	The system must show role specific view based on the user's role
FR-03	Worker	The worker must be able to choose the right location from a list
FR-04	Worker	The worker must be able to choose the right task from a list
FR-05	Worker	The worker must be able to start/stop a work session
FR-06	Worker	The worker must be able to pause/resume active sessions
FR-07	Worker	The worker must be able to add quantity and details to a stopped work session
FR-08	Worker	The worker must be able to view, edit and delete own sessions
FR-09	Manager	The manager must be able to view, edit, delete and restore all sessions
FR-10	Manager	The manager must be able to view session edit history
FR-11	Manager	The manager must be able to view daily, weekly, monthly and yearly reports from selected sorting options – (location, worker, time period/date, time spent/quantity)
FR-12	Manager	The reports must be shown in a table and a chart format
FR-13	Admin	The admin must be able to create, edit, delete and restore users
FR-14	Admin	The admin must be able to edit the name, username, password and role of existing users
FR-15	Admin	The admin must be able to create, delete, edit and restore tasks and locations
FR-16	Admin	The admin must be able to create and manage task categories for specific locations
FR-17	Admin	The admin must be able to assign correct tasks to correct locations
FR-18	Admin	The admin must be able to assign a task to a category in a location
FR-19	Admin	The system must prevent actions that would remove the last admin or let the user delete themselves

3.4 Non-functional requirements

Non-functional requirements define in what operating conditions the system should function and what kind of quality features it should fulfill. They don’t represent individual user actions but are connected, for example, to security, performance, usability, maintainability and limitations of the system. Together with the functional requirements they help to make sure that the system works in real usage (Zahran & Widyarto, 2025).

The non-functional requirements of this system are especially connected to safe login, user-role management, input validation, work session limits and safe data management. Because the system is used to log operational work, it should be clear and safe for daily usage and to support later reporting and analysis. The non-functional requirements are listed in Table 3 below.

Table 3. Non-functional requirements.

NON-FUNCTIONAL REQUIREMENTS		
ID	CATEGORY	NEED
NFR-01	Security	Authentication must be secure and session data must be protected
NFR-02	Security	Role-based authorization must be done in the backend
NFR-03	Security	Passwords must be stored safely and hashed
NFR-04	Security	User inputs and API parameters must be processed and validated before usage
NFR-05	Security	Database queries must be protected from injection attacks
NFR-06	Data accuracy	Work session time logic must prevent false start/stop/pause values
NFR-07	Business rule	Workers must not be able to start more than 5 simultaneous work sessions
NFR-08	Business rule	Long-lasting work sessions must be stopped automatically after 8h
NFR-09	Traceability	Work session edits, deletes and restores must be stored in edit history
NFR-10	Usability	UI must give immediate feedback from user actions
NFR-11	Usability	UI must be responsive and usable in desktop and mobile view
NFR-12	Performance	System must respond quickly enough for daily operational usage
NFR-13	Architecture	The system must use clear frontend, backend and database architecture
NFR-14	Architecture	The frontend must communicate with the backend with clear API endpoints
NFR-15	Deployment	The system must be deployable for production-like environment that matches Posti Oyj's infrastructure and deployment practices
NFR-16	Deployment	The system's source code and deployment must match Posti Oyj's version control and deployment practices
NFR-17	Data accuracy	The system must use soft-delete logic where important data is deactivated instead of deleted

3.5 Success criteria for the prototype

The success criteria define how the success of the prototype is being evaluated. In software development success doesn't only mean that the system is technically ready, but also that the solution meets the expectations and defined goals. Tamburri et al. describe the success of a software project as a state where the solution meets its expectations, whereas failure happens when these expectations are no longer met. There are many other aspects also that contribute to the success of the software, for example, system requirements, system design and other qualities of the project (Tamburri et al., 2020). This is why the success criteria of this prototype is connected to the requirements, expectations of the case organization and the recognized case issues.

In this study, the prototype isn't evaluated as a production-ready system, but as a proof-of-concept solution whose purpose is to show that the issues related to warehouse task logging, tracking and managing can be solved with a digital system. The prototype can be considered successful if it meets the functional requirements from the view of the worker, manager and admin, and the non-functional requirements like safe logging, role-based authorization and input validation. For success it is also important that the UI is clear for daily usage and that the collected data can be used in reporting. If these criteria are met, the prototype can be considered as a successful base for further development and later production use.

4 System design

This section describes how the prototype is designed based on the theoretical background and given requirements. The design choices are based on the principles in section 2 such as web application architecture, UI/UX design, security, reporting and visualization. The design is also based on the functional and non-functional requirements given in section 3. The goal of this section is to link the needs of the system to concrete design choices before describing the actual implementation. This is why the section focuses on the architecture of the system, responsibilities of different parts, design based on user roles and the technical principles that guide the implementation of the prototype.

4.1 Architecture overview

The system architecture is designed to be a three-tier architecture, where the UI, system logic and the database are separated as clear individual modules. The main parts of the system are the frontend, backend and a database. The frontend is responsible for the UI and directing the inputs of the user forward. The backend is responsible for business logic, input validation, role-based authorization, error handling and database connections. The database works as the system's central storage for data related to users, locations, tasks and work sessions.

The frontend is designed to not communicate with the database directly. All actions related to the database go through the backend. This solution ensures that the UI can be separated from the database and the backend will be responsible for checks relating to the database. This enhances the security and maintainability of the system because the logic relating to the user roles, inputs, business rules and errors can be managed in one place before storing or fetching data.

The communication between the frontend and the backend is designed to be implemented with RESTful APIs. This fits the system because the actions are made of clear

user actions like logging in, work session management, fetching reports and maintaining users, locations and tasks. Clear API structure also supports further development because the logic of the UI and the backend can be kept separate.

The frontend should follow a SPA approach. This choice supports the interactions of the users when they are making repetitive and fast actions during a workday. SPA minimizes page loads and enables smooth interactions with the system. Since the system is intended for internal operational use and not a public content-driven website, SEO is not a key requirement in the architectural choice.

The architecture should also support reporting and later analysis. The purpose of the system is to collect data from work sessions that can be later inspected based on, for example, location, worker, time period or quantity. This is also why data storing and management is designed to be through the backend and database. This keeps the work session data consistent and usable in reporting.

4.2 Frontend design

The frontend design must first clarify who uses the software, where the software is going to be used and how the software is going to be used. The primary users of the system are the warehouse workers, who are going to be using this software in the warehouse. They need to be able to quickly start and stop work sessions in a fast and moving work environment. Secondary users of the system are the managers. They will use the software mainly in the office to inspect and analyze reports and all work sessions. The third user group of the system are the admins that will also use the software in the office to manage users, roles and worksite information. Based on these user groups and their work environments, the frontend needs to be designed to be clear and easy to use during a busy workday.

The warehouse workers' view is mainly designed for mobile usage because the workers use the system in a mobile warehouse environment. The workers' main actions are

choosing the right location and task and starting, pausing, continuing and stopping a work session. These actions need to be quickly reachable and clear so that using the system doesn't slow down the actual job or cause data inaccuracies because of, for example, wrong choices, incorrect inputs or forgetting to log a work session. The view is also designed to have the option to inspect, edit and delete own work sessions. This enables the correction of falsely logged sessions and thus enhances the quality of the data.

For the managers, the view is designed for desktop usage to be able to efficiently inspect and analyze all the information in the reports and all work sessions. The reports' view for managers is designed to be able to show relevant and desired data in an easy-to-read form. To support this, the interface should offer an option for sorting and filtering only the wanted data to inspect. The information is designed to be shown in a comprehensive view, showing all the data for deep analysis. It's also shown in a suitable graph for easy and quick analysis.

The admins' view is designed for managing basic information. The admin-user is designed to be able to control users, locations, tasks, categories and relationships between locations and tasks. The view is designed to highlight clear forms, buttons and error prevention because the actions of the admin affect the whole structure of the system and choices visible to the workers.

The frontend is designed to be made using the same visual looks and branding of Posti Oyj. The primary colors are soft orange and dark grey for good contrast and easy readability. All inputs and interactions must be clear and deliver immediate feedback to the user, stating clearly that they have been pressed and what the result of that is. This reduces uncertainty and supports daily usage in a busy work environment.

The frontend-technology is designed to be based on JavaScript and React.js. React supports the SPA and enables a component-based UI where the role-based views can be

built from reusable components. This supports maintainability and reduces the need for repetitive use of UI code. The structure of the UI is designed so that different components are separated from each other.

The data fetching of the frontend is designed to happen through users' actions and the needs of the different views. All data isn't meant to be fetched all the time, but when the action of a user or a view needs it. This is designed to reduce unnecessary load and support the performance needed for operational usage.

4.3 Backend design

The backend is designed as the middle layer of the system, connecting the frontend and the database. Its responsibility is to receive the requests coming from the frontend, perform all the necessary checks, handle the business logic and return the requested data to the frontend. Designing the backend as a separate layer supports maintainability, security and clear definitions of responsibilities.

The APIs of the backend are designed based on sets of functionalities. These functionalities are, for example, authentication, work sessions, reporting, users, locations, tasks, categories and the links between tasks and locations. This structure enables the separation of the central user actions into logical API sets, which supports maintainability and further development.

The business rules are designed as the responsibility of the backend. These include, for example, checking the time values of work sessions, allowing a maximum of five simultaneously active work sessions, automatic stopping of too long work sessions and checking that the chosen task belongs in the chosen location. Performing these checks in the backend is important because the checks in the frontend are not enough to guarantee the quality of the data.

The backend is also designed to be responsible for the role-based authorization. The role of the user defines what actions they can perform. The check for the appropriate permissions needs to be done in the backend and not only in the frontend. This ensures that the worker, manager and admin are allowed to the right actions and data.

The backend is also designed to support reporting. Requests relevant to reporting include filtering, such as location, worker, time period or quantity. The responsibility of the backend is designed to process these filters, fetch the right information from the database and return it to the frontend in a form where it can be used in a comprehensive view and a graph.

4.4 Database design

The database is designed as the permanent data storage system. Because the system's data consists of clear and linked sets like users, locations, tasks, categories and work sessions, the database solution is designed to be a MySQL database. This solution supports managing the relationships between datasets and filtering needed in reporting.

The database structure is designed to support the three main roles of the system. The database should store the data related to work sessions, such as location, task, start and stop time, breaks, quantity and more information for workers. For managers, the data is designed to be fetched for reports and inspecting all work sessions. For admins, the database is designed to include all the changeable information, such as users, locations, tasks, categories and the link between tasks and locations.

The database is also designed to support traceability and safe deletion. Work session edits, deletes and restores should be stored in edit history. This enables later checks to see what has been done to the data. In important data tables, soft-deleting principles are used where the data isn't permanently deleted but marked as active or inactive. This supports the quality of data, trustworthiness of reporting and enables information to be restored later.

The database is also designed to support reporting. Work session data is stored with enough attributes to be filtered and sorted, for example, by worker, location, time period or quantity. This is essential because the goal of the system is to generate more trustworthy data about workload and resource usage than now.

4.5 Security design

The security design of the system is based on the fact that the system uses sensitive user-based and operational data. This is why the security design is noted in every part of the system: the frontend, the backend and the database. Key design choices are safe authentication and authorization, input validation, safe data processing and prevention of misuse.

Authentication is designed to verify the identity of the user before using the system. The user needs to log in before using protected actions. Session data should also be handled securely. Passwords are designed to be safely stored and hashed for improved security.

Authorization is designed to be based on roles. The worker can handle their own work sessions, the manager can inspect and handle all work sessions and form reports, the admin can handle the basic information of the system. The role-based UI enhances usability but the actual check for authorization should be done in the backend, so that the user cannot bypass it through the frontend.

Input validation is designed to be included in both the frontend and backend. Inputs given by the user like quantities, time values and text should be checked before processing. The backend is also designed to prevent database queries from injection attacks. These design choices help prevent false data inputs and improve security.

The security design also includes the business rules. The worker should only be allowed to have five active work sessions simultaneously, work sessions should stop automatically when running too long and admins should not be able to delete or remove permissions from the last active admin. These rules protect the system from misuse and support data quality.

4.6 Deployment design

The deployment design is based on the requirements of the system and guidance from the IT team of Posti Oyj. Because the system is designed for internal usage of Posti Oyj and that it can be easily developed further later, the deployment design must match the current deployment, CI/CD and cloud-infrastructure practices of Posti Oyj.

The deployment is designed based on version control, workflow-based deployment and AWS-based cloud-infrastructure. The system is designed to be containerized so that it can be packed as a single unit and run in a deployment environment. From the AWS services, the main ones are a service fit to run containers, a service to store containers and a MySQL compatible database service.

The deployment design also includes environment-based configuration. Database connections, credentials and other environment-based information aren’t designed to be hard coded in the system but are controlled through the deployment environment. This supports more secure and maintainable deployment.

As a whole, the deployment design forms a model where the version control, workflow-based deployment, running a containerized system and the AWS infrastructure support moving the prototype into production.

5 Implementation

This section describes how the design of the system in section four was implemented. The section focuses on the key technical aspects in the system like frontend structure, API structure of the backend, database, the deployment process and testing. This section is not meant to be used as a user guide but to show how the design choices were formed into a working prototype.

5.1 Frontend implementation

The frontend was implemented as a React-based SPA solution. The implementation follows the component and role-based structure made in the design phase. Figure 1 shows the login view for all users, from where the user is directed to the right view based on their role. The role-based views were implemented for workers, managers and admins. The specific views support the necessary actions needed for the role.

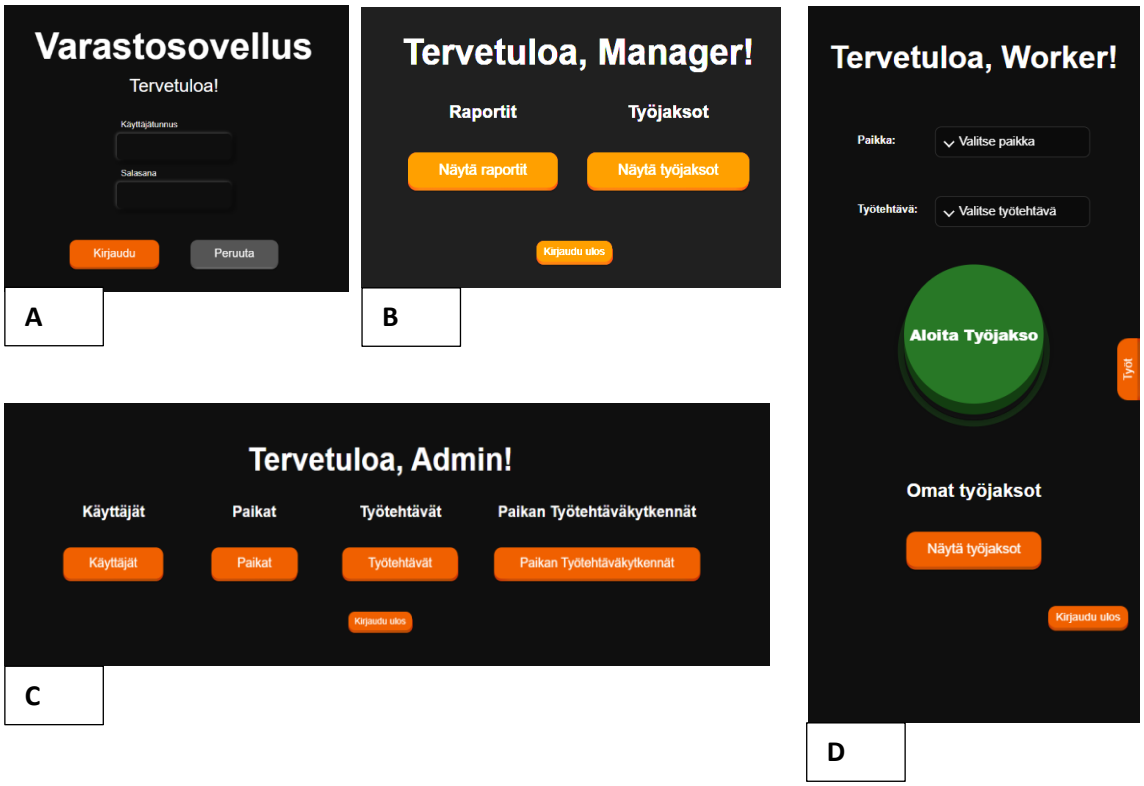

Figure 1. Different main menu views. A) Login menu, B) Main menu for manager, C) Main menu for admin, D) Main menu for worker.

The view of the worker was implemented with an emphasis on mobile usage. Figure 2 shows the main functions of the worker view, where the worker chooses the location and a task, and starts the work session, manages active sessions and can inspect, edit or delete their own sessions. The task list updates based on the chosen location, only showing the tasks available for that location. Active work sessions are shown in a right-side drawer where the worker can pause, continue and stop active sessions. When stopping a session, the worker can also add quantities and more information. This implementation was aimed to support fast usage in a warehouse and to minimize false work session logs.

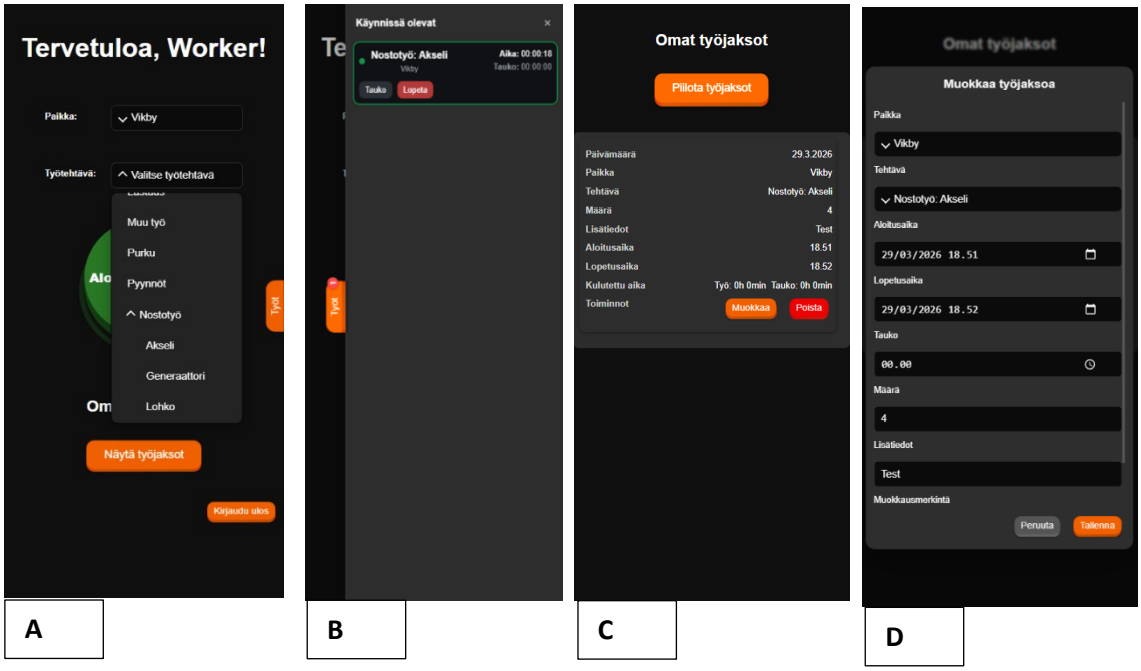

Figure 2. Functions of the worker view. A) Worker choosing a location and a task, B) Active work sessions of the worker, C) Own sessions of the worker, D) Editing a work session.

The view of the manager was implemented for reporting and inspecting all work sessions. Figures 3-5 show the main functions of the manager view, including report filtering, all work session inspection and work session edit history. In the report view, the manager can filter data according to the location, worker, time period and quantity. The results are shown both in a table and a pie chart so that the data can be analyzed comprehensively and to get a quick overall picture. The implementation for the view of the manager

also included the possibility to inspect, edit, delete and restore all work sessions. The manager can also view the edit history.

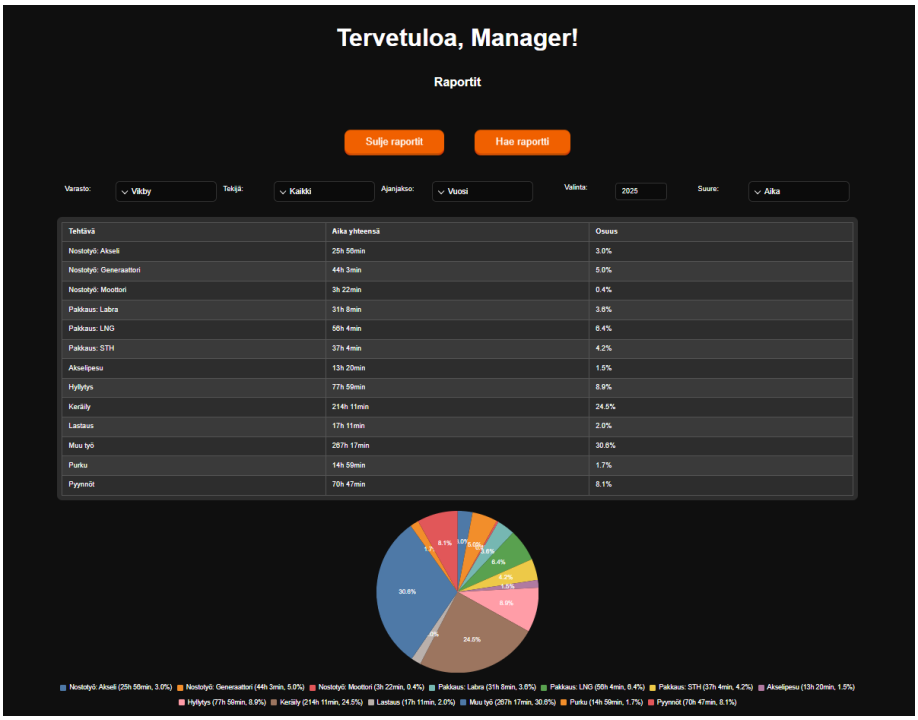

Figure 3. Manager example report.

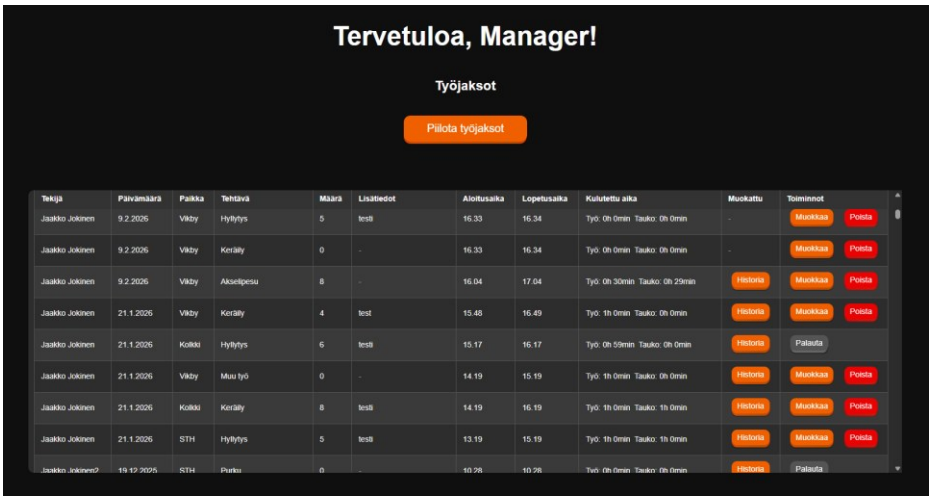

Figure 4. Manager's all work sessions view.

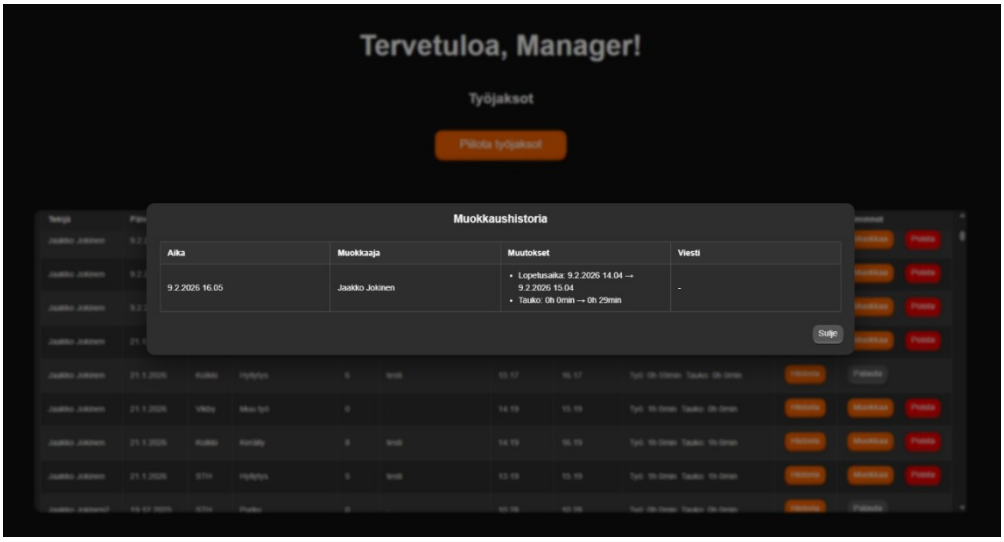

Figure 5. Work session edit history.

The view of the admin was implemented to control basic system information. Figures 6-9 show the main functions of the admin view, where admin can create, edit, delete and restore users, locations, tasks, and categories. The admin can also link tasks to locations and categories. With these implementations, the data shown in the UI can be kept under control and the role-based views functionality is based on managing basic system information.

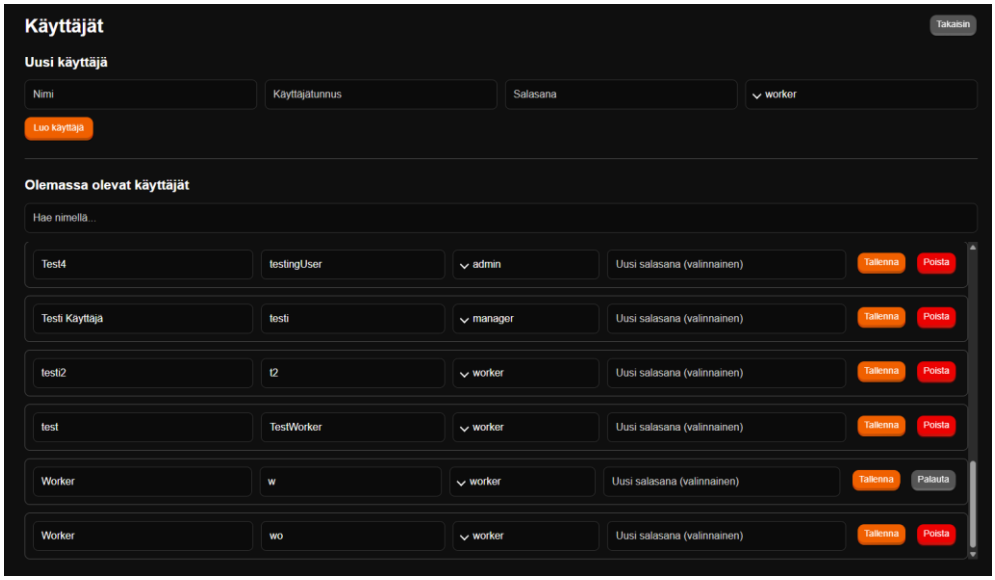

Figure 6. Admin user interactions.

Figure 7. Admin location interactions.

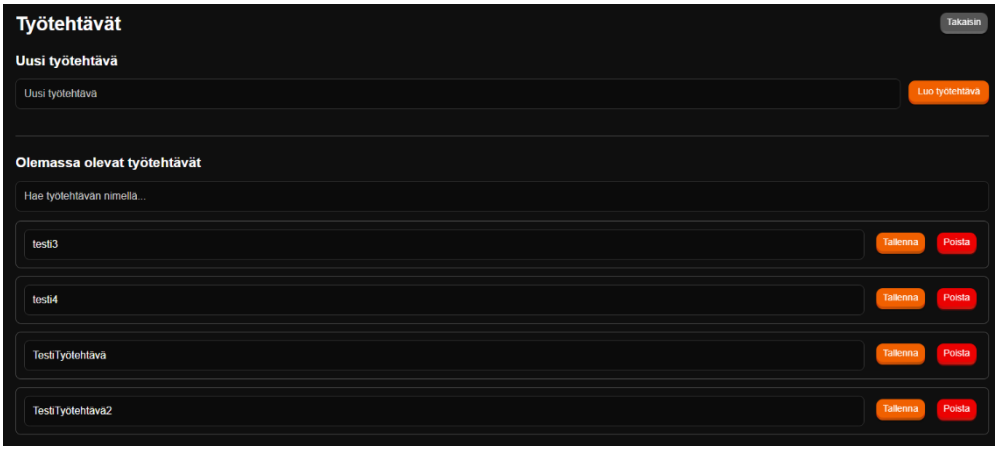

Figure 8. Admin task interactions.

Figure 9. Admin assigning tasks to locations.

The technical implementation of the frontend included splitting the structure where the UI consists of main components and smaller reusable components and functions. Figure 10 shows an example of the reusable frontend components. This means that the reused selectors, list components and control elements were made each as their own components. This solution decreased redundancy and made making changes easier without needing to change the whole system logic.

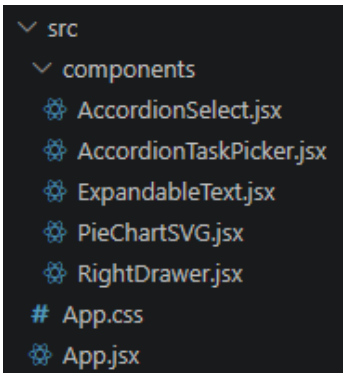

Figure 10. Reusable components.

The state management was implemented using React's native hooks (useState, useEffect and useMemo) without any large global state management library. This was reasonable because the states of the system are divided into view-based components (login, worker's view, manager's view and admin's view). This kind of implementation keeps the data flow clear and decreases unwanted complexity compared to a heavier state management solution.

The communication between the frontend and the backend was implemented using the RESTful APIs. A helper function has been made for handling JSON-type requests, credentials and uniform error handling. This way all of the HTTP methods follow the same request pattern, which increases maintainability and readability. Fetching data has been bound to user actions and view transitions to avoid unnecessary load.

The responsiveness of the system was handled with CSS. The view of the worker is based on quick actions with a mobile view. The actions of the manager and admin are based on easy readability and analysis on a larger desktop view. Overall usability of the system was improved with unified buttons and field components, clear alerts to give the user immediate feedback of their actions and a unified colour scheme according to the visual looks and branding of Posti Oyj.

5.2 Backend implementation

The backend was implemented by using the RESTful APIs. The APIs were divided into clear wholes, such as authentication, work sessions, reporting, users, locations, tasks and categories. This structure made the backend implementation clearer because the different functionalities could be separated as their own logical parts. The frontend uses these APIs for logging in, work session management, fetching reports and admin actions.

The authentication was implemented by login functions where the credentials given by the user are compared to the credentials of the user in the database. Passwords are processed as hashed and not stored in plain text. After a successful authentication, session handling is managed with JWT-tokens. The frontend gets the right view based on the role of the user. The backend checks the authentication with safe API routes before the user can make any actions.

Authorization was implemented as role-based. The backend checks if the user has the right authorization to perform an action. The workers have the right to control their own work sessions, managers can control and inspect all work sessions and reports, and the admins can control users, locations, tasks, categories and links between locations and tasks. Implementing the role-based authorization in the backend is important because only the role-based views in the frontend aren’t enough to prevent unauthorized API requests.

The business rules for the work sessions are also implemented in the backend. When a worker starts a work session, the backend checks, for example, the rights of the user, chosen location, the task, possible other active sessions and that the worker doesn't have too many simultaneous sessions active. When stopping, pausing, continuing or editing a work session, the backend takes care of the correctness of time values. This implementation improves data quality and consistency.

The API routes for reporting are implemented so that the frontend can fetch data based on given filters. The backend processes, for example, the location, worker, time period and quantity. Then the backend forms the necessary database queries and returns the results to the frontend. This implementation enables that the same work session data can be used in a table as well as in a pie graph.

Implementation for the input validation and error handling are implemented in the backend as a part of every essential function. The backend checks, for example, mandatory fields, correctness of the characters, time values and authorization. In error situations the backend returns a clear HTTP answer to the frontend. Based on that the frontend can give a clear alert to the user.

Database queries are implemented in the backend so that the frontend isn't able to communicate directly with the database. The queries use SQL-queries formed by the backend and in important functions the quality of the data, soft delete logic and edit history storing are noted. This implementation supports security, traceability and trustworthiness.

5.3 Database implementation

The database solution is implemented with a MySQL relational database. The database structure is based on the main entities of the system that are users, locations, tasks, categories and work sessions. The database also needed tables for storing links between

locations and tasks, as well as work session edit history. Figure 11 shows the relational database structure of the prototype. A relational database fits this implementation because the tables have clear connections.

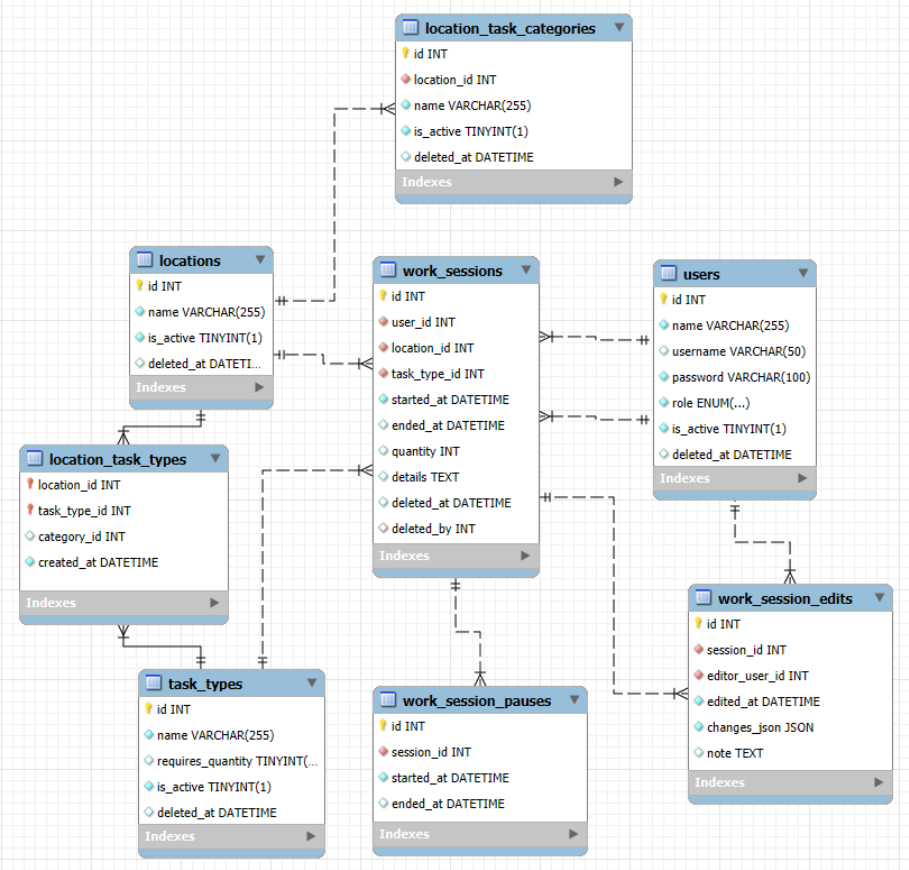

Figure 11. Database ERD.

The user table stores the user's basic information, login credentials and role. Based on the role, the system can direct the user to the correct view and filter in the backend what actions the user is authorized to do. Locations, tasks and categories are stored in their own tables, and they can be controlled from the view of the admin. Linking tasks to locations was implemented in a different table so that the user can only see correct tasks for each location.

Work sessions are stored in their own table. Information related to the work sessions is the user, location, task, start time, end time, quantity, details and possible deletion. With

this information, the work session data can be used to inspect work sessions and form reports.

Soft-delete logic is also implemented in the database. Important data, such as users, locations, tasks and work sessions aren't deleted permanently but the data is marked as deleted or passive. This enables restoring information and reduces the risk that important data is lost permanently.

Tracing the edit history of work sessions is implemented with the work session edits table. When a work session is edited, deleted or restored, it leaves a mark. The history includes the editor, time of edit and the information of what was edited. This improves the transparency of the system and enables managers to inspect work session edits.

5.4 Deployment and CI/CD

The deployment is implemented according to the guidance and requirements of Posti Oyj. The deployment noted the version control-, CI/CD- and cloud-infrastructure principles of the organization. This is why the deployment is based on GitHub, GitHub Actions workflows and AWS. The goal of the deployment implementation is to build a process where the system can be containerized, published and run in a production-like environment as controlled and reliable as possible.

The deployment process begins when the source code changes are pushed into the GitHub repository. When the changes are pushed to the repository, a GitHub Actions workflow triggers the build process where it builds a Docker image from the system and pushes it to an AWS ECR repository. The container-based implementation enables the system to be packed into a clear whole.

When the Docker-image has been built and successfully pushed into the ECR, another GitHub Actions workflow triggers that deploys the container from the ECR into the AWS ECS environment. In ECS the system is deployed through the ECS Cluster, ECS Service and

ECS Task. The traffic of the user is guided to the system through the Application Load Balancer and Target Group. This way the traffic can be controlled, and the users don't need to be connected straight to the container.

The system uses the AWS RDS MySQL database, to which the ECS Task makes a connection using environment variables. The connection information of the database, such as host, name, user or password aren't hard coded into the system but handled in the deployment environment. This kind of implementation increases security and maintainability. The RDS databases data can be used later in Databricks for analytics.

This deployment implementation forms a clear CI/CD pipeline where the source code change goes through the GitHub repository, GitHub Action workflows, Docker-image, AWS ECR and AWS ECS into a production-like environment. Figure 12 shows this CI/CD pipeline. This kind of implementation supports the deployment practices of Posti Oyj and enables the further development of the prototype and the push into production.

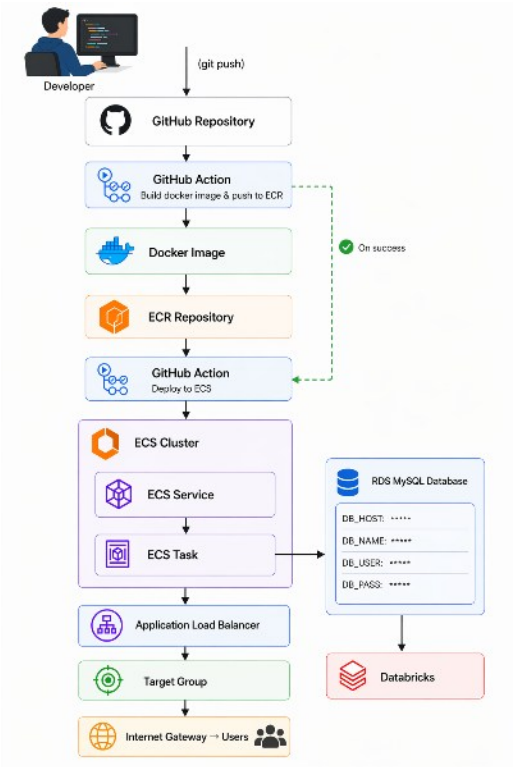

Figure 12. CI/CD Pipeline. Note. This figure has been generated with AI (OpenAI, 2026)

5.5 Testing

The testing was done as a part of an agile development process. The prototype was in use and tested in one of the in-house logistics operative units of Posti Oyj during development where the new functionalities were tested in use together with the warehouse workers. The prototype was developed one functionality at a time. After each new feature or change, the prototype was tested in practice. The testing was focused on a UAT approach, where the evaluation was based on the fact whether the implemented functions or changes would fit the users' needs, requirements or success criteria.

Because the functional scope of the system was limited it was also possible to test all of the central functionalities connected to the new functionality or change. This way the testing also worked as a type of regression testing because it was made sure that the change didn't break anything. The testing included, for example, logging in, role-based views, work session start, work session pause, work session continuation, work session stop, editing own sessions, filtering reports and admin actions.

The testing of the backend focused especially on authentication, role-based authorization, input validation and error handling. The testing made sure that the secure API routes couldn't be used without logging in, users with incorrect role can't make unauthorized requests and that false requests return a clear HTTP message to the frontend to process.

The purpose of the testing wasn't to build a large automatic testing system but to verify the functionality of the prototype, usability and fulfilment of the requirements during development. Because the system had a limited functional scope, the manual UAT testing was a good option to spot mistakes and to make sure that the prototype is ready for further development.

6 Evaluation and results

This section evaluates the results of the prototype against the given requirements and success criteria. The evaluation considers both the testing made during development and the practical usage of the prototype in a real in-house logistics operative unit environment of Posti Oyj. Because the prototype was in real use with real workers, the evaluation isn't only based on technical inspection but also practical functionality in an operational environment.

6.1 Evaluation method

The evaluation method is based on fulfilment of the requirements, success criteria and UAT testing. The prototype was tested during development in one of the in-house logistics operative units of Posti Oyj where the users used it in real work situations. After every new functionality, it was tested and also all other central functionalities to make sure that the whole system worked.

The final prototype was tested for one month. The evaluation examined whether the workers can log work sessions fluently, can the manager inspect all work sessions and reports, and that can the system support role-based usage and secure actions.

6.2 Results and fulfilment of success criteria

Based on the evaluation, the prototype fulfils the central functional requirements. The users can log in and are directed to the right role-based view. The workers can choose a location and a task, start, pause, resume and stop work sessions, inspect and edit their own work sessions. This supports the main goal of the system, more accurate and trustworthy logging of tasks.

For the manager the prototype enables inspecting work session data and to form reports. The reports can be filtered based on wanted data, and they are presented in a table and

a pie chart. Also, the functional requirements for the admin are fulfilled. The prototype enables managing the users, locations, tasks, categories and links between locations and tasks.

The non-functional requirements are also filled with the prototype. The system includes secure login, role-based authorization, validation in the backend, soft delete logic and storing edit history. The UI supports the workers' mobile view and the desktop view for managers and admins.

During the UAT, the prototype received positive feedback from the users. The system was thought of as useful because it enabled tracking and inspecting tasks where there were no clear existing data. Also, in the UAT, the prototype gathered useful data and this was very much appreciated, especially by the customer of that Posti Oyj's in-house logistics operative unit. Because the prototype was in testing for a month without any problems, it can be considered as a successful proof of concept solution and ready for further development.

6.3 Limitations and reliability of the results

Even though the prototype was tested in a real in-house logistics operative unit environment, there are still limitations in the results. The UAT was only done in one in-house logistics operative unit, so that the results don't clearly show how the system would function in multiple locations and with a larger quantity of users and data. Furthermore, the prototype isn't yet evaluated as a full scale production system.

The reporting benefits can be considered promising because the system enables collection and inspection of work session data. The true impacts on resource planning and operational decision-making can be evaluated later when the system is in larger use and with increased quantity of data.

The reliability of the results is also impacted by the fact that the data produced by the system is only dependent on the right and consistent usage of the workers. Even though the system reduces false work session logs with technical checks, in the end the correctness of the data requires proper onboarding and training of workers and that the work sessions are being logged truthfully.

Even with these limitations, the trustworthiness of the evaluation is increased by the fact that the prototype was tested with real users in a real warehousing environment for a month. Based on the results, the prototype proves that problems related to task tracking, data collecting and reporting can be solved with a digital system. Final deployment to production still needs further development, testing and possible UAT in different in-house logistics operative units.

7 Conclusions

The goal of this study was to design and implement a software prototype for the collection and visualization of Posti Oyj's in-house logistics data. The starting point was the problem in which not all warehousing tasks could be properly tracked with existing systems. This weakened the management and forecasting of resources and workload.

The study identified three central user roles: workers, managers and admins. The workers needed a clear way to track tasks, the managers a clear way to inspect collected data and the admins the tools to manage basic system information. From the identified non-functional requirements, the most important were security, role-based authorization, input validation, data correctness, traceability and usability.

The prototype was implemented as a three-tier web application that consists of a React-based frontend, the API layer of the backend and a MySQL database. The frontend was designed as a SPA solution where the view of the worker supports mobile usage and fast work session logging. The views of the manager and admin were designed for desktop usage. The backend is responsible for authentication, authorization, business logic, validation and database connection. The deployment was implemented with the guidance and requirements of Posti Oyj with the help of CI/CD workflows, containerizing and AWS.

Based on the evaluation, the prototype fulfilled the given functional and non-functional requirements. The workers could start, pause, continue, stop and inspect their work sessions. Managers could inspect work session data and form reports based on filters. The admins could control the basic information of the system.

The prototype was tested in a real warehousing environment with real users, which enhances the reliability of the results. During the testing the prototype got positive feedback and gathered useful data that was appreciated, especially by the customer of that in-house logistics operative unit.

Some limitations are also included in the study. The prototype was only tested in a single in-house logistics operative unit, so the results don’t yet prove the functionality of the system on a larger scale. Furthermore, the reliability of the data is dependent on the truthful and correct use by the workers. For the prototype to be production-ready, it needs further development and testing.

As a whole, the study showed that problems related to inaccurate task-data collection and the underuse of such data can be addressed with digital systems.

References

Akpe, O. E. E., Kisina, D., Owoade, S., Uzoka, A. C., Ubanadu, B. C., & Daraojimba, A. I. (2022). Systematic review of application modernization strategies using modular and service oriented design principles. *International Journal of Multidisciplinary Research and Growth Evaluation, 2*(1), 995–1001. https://doi.org/10.54660/.IJMRGE.2022.2.1.995-1001

Almataani, N. A. R., & Ullah, D. A. (2024). Storage operations in warehouses: an review. *International Journal of Social Sciences and Management Review, 7*(01), 2024. https://doi.org/10.37602/IJSSMR.2024.7112

Alonge, E. O., Eyo-Udo, N. L., Ubanadu, B. C., Daraojimba, A. I., Balogun, E. D., & Ogunsola, K. O. (2023). Real-time data analytics for enhancing supply chain efficiency. *Journal of Supply Chain Management and Analytics, 10*(1), 49–60. https://doi.org/10.54660/.IJMRGE.2021.2.1.759-771

Baralis, G. (2024). *Forecast inbound volumes and workforce needs in a logistics warehouse* (Doctoral dissertation, Politecnico di Torino). Retrieved 12.1.2026 from https://webthesis.biblio.polito.it/32039/1/tesi.pdf

Granić, A. (2017, December). Technology in use: The importance of good interface de sign. In *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions)(ICTUS)* (pp. 43–49). IEEE. https://doi.org/10.1109/ICTUS.2017.8285972

Guedes, G. P. (2025). *Digital transformation of workforce task monitoring in a logistics operational center* (Master's thesis, Universidade do Porto, Portugal). Retrieved 20.1.2026 from https://repositorio-aberto.up.pt/bitstream/10216/168193/2/733055.pdf

Hernández, L. M., Cadena, A. H., Vázquez, J. N., Magaña, J. Á., & Zea, J. M. (2020). REST (Representational State Transfer) architecture for enterprise web application development. *Innovación Y Desarrollo Tecnológico Revista Digital, 12*(3), 219–27. Retrieved 18.2.2026 from https://iydt.wordpress.com/wp-content/uploads/2020/07/3_11_rest-representational-state-transfer-architecture-for-enterprise-web-application-development.pdf

Kaur, G., & Tiwari, R. G. (2023). Comparison and analysis of popular frontend frame works and libraries: An evaluation of parameters for frontend web development. In *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)* (pp. 1067–1073). IEEE. https://doi.org/10.1109/ICESC57686.2023.10192987

Kellermayr-Scheucher, M., Niedermeier, M., & Brandtner, P. (2023). Applications and perceptions of workforce management systems for warehouse operation-results and findings from expert interviews. *Procedia Computer Science, 219*, 255–262. https://doi.org/10.1016/j.procs.2023.01.288

Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2022). Systematic literature review on security risks and its practices in secure software development. *IEEE Access, 10*, 5456–5481. https://doi.org/10.1109/ACCESS.2022.3140181

Lindner, F., Reiner, G., & Keil, S. (2025). A behavioral perspective on visualization in manufacturing and operations management: A review, framework, and research agenda. *Operations Management Research, 18*(1), 317–352. https://doi.org/10.1007/s12063-024-00534-9

Nagy, G., Bányainé Tóth, Á., Illés, B., & Varga, A. K. (2023). The impact of increasing dig itization on the logistics industry and logistics service providers. *Multidiszciplináris Tudományok: A Miskolci Egyetem Közleménye, 13*(4), 19–29. https://doi.org/10.35925/j.multi.2023.4.3

OpenAI. (2026). *ChatGPT* (GPT-5.5 Thinking, 23.4.2026) [large language model]. Retrieved 27.4.2026 from https://chat.openai.com/chat

Pant, P., Rajawat, A. S., Goyal, S. B., Bedi, P., Verma, C., Raboaca, M. S., & Enescu, F. M. (2022). Authentication and authorization in modern web apps for data security using Nodejs and role of dark web. *Procedia Computer Science, 215*, 781–790. https://doi.org/10.1016/j.procs.2022.12.080

Phan, K. (2024). *A comprehensive study on single-page applications: Pros, cons, and practical guidelines*. (Bachelor's thesis, Haaga-Helia University of Applied Sciences). Theseus. https://urn.fi/URN:NBN:fi:amk-2024112730662

Punchoojit, L., & Hongwarittorrn, N. (2017). Usability studies on mobile user interface

design patterns: A systematic literature review. *Advances in Human-Computer Interaction, 2017*(1), 6787504. https://doi.org/10.1155/2017/6787504

Shakeel, H. M., Iram, S., Al-Aqrabi, H., Alsboui, T., & Hill, R. (2022). A comprehensive state-of-the-art survey on data visualization tools: Research developments, challenges and future domain specific visualization framework. *IEEE Access, 10*, 96581–96601. https://doi.org/10.1109/ACCESS.2022.3205115

Shao, S., Qiu, Z., Yu, X., Yang, W., Jin, G., Xie, T., & Wu, X. (2020, September). Database-access performance antipatterns in database-backed web applications. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 58–69). IEEE. https://doi.org/10.1109/ICSME46990.2020.00016

Singh, J. (2025). *The impact of increased competition and digitalization on Posti delivery strategies* (Bachelor's thesis, Centria University of Applied Sciences). Theseus. https://urn.fi/URN:NBN:fi:amk-2025061122267

Tamburri, D. A., Palomba, F., & Kazman, R. (2020). Success and failure in software engi neering: A followup systematic literature review. *IEEE Transactions on Engineering Management, 68*(2), 599–611. https://doi.org/10.1109/TEM.2020.2976642

Verma, A., Tripathy, S., & Singhal, D. (2023). The significance of warehouse manage ment in supply chain: An ISM approach. *Decision Making: Applications in Management and Engineering, 6*(1), 92–110. Retrieved 12.1.2026 from https://distantreader.org/stacks/journals/dmame/dmame-417.pdf

Widjaja, A. A., Ghapanchi, A. H., & Bingley, S. (2025). Exploring the antecedents to the effective use of business intelligence: A systematic review approach. *Information Systems Management, 42*(4), 525–545. https://doi.org/10.1080/10580530.2025.2479737

Zahran, A., & Widyarto, S. (2025, April). The importance of comprehensive software re quirements developing an effective software requirements specification (SRS) document. In *Proceedings of the Informatics Conference* (Vol. 11, No. 22, pp. 20–24). Retrieved 3.2.2026 from https://ojs.journals.unisel.edu.my/index.php/icf/article/view/403