Pile Driving Monitoring System

Van Dijk Heitoezicht Platform Development Team

Report submitted as the final deliverable for course:

Software Project (CSE2000)

Group 15A

Matei Aruxandei

Bogdan Duminica

Stefan Voicila

Andrei Mosescu

Mihai Marcu

Client: **Van Dijk Heitoezicht en Funderingsexpertise BV**
Client Representative: **Rob van den Hoven**
Teaching Assistant: **Yigit Çolakoğlu**, Coach: **Inald Lagendijk**

Date: **20 June 2025**

Preface

This report was prepared by five second-year Computer Science students from TU Delft as part of the CSE2000 Software Project course.[1]

We want to express our gratitude and appreciation to everyone who supported us throughout the project's timeline. In particular, we would like to thank:

- Client Rob van den Hoven for always providing prompt feedback and actively engaging with us throughout the project.
- Koen Faro, the client's technical specialist, for always providing expert support and helping us overcome technical challenges whenever we got stuck.
- TU Delft support through our teaching assistant Yigit Çolakoğlu for providing constructive feedback and our coach Inald Lagendijk for valuable implementation suggestions and guidance.
- Technical Writing and Teamwork lecturers from TU Delft for sharing their expertise and engaging lectures.

The team during a trip to a construction site in Zoeterwoude.
From left to right: **Bogdan Duminica**, **Stefan Voicila**, **Mihai Marcu**, **Andrei Mosescu**, **Matei Aruxandei**.

[1] **Total word count: 11,340**
Statement on the use of generative AI: In writing this text, we used generative AI to improve the grammar, style, and/or spelling of the text.

Summary

Pile-driving monitoring on Dutch construction sites remains largely manual: supervisors record hammer counts, depths, delivery notes, and plan progress by hand. This fragmented workflow is time-consuming, error-prone and hard to integrate, which slows decision-making and increases costs. Our project automates data capture, processing, and visualization to streamline supervision and reporting and thus makes the process faster and more reliable for the supervisors.

The core engineering challenge was to ingest diverse inputs, real-time sensor streams (ESP32-based accelerometer and audio), scanned receipts, and DWG/PDF pile plans and convert them into actionable insights. We developed a microservices platform (React frontend, Node.js/WebSocket back-end, PostgreSQL) that can handle thousands of simultaneous users and provide sub-second dashboard updates.

Key features include:

- A live, WebSocket-powered dashboard showing blows per 250 mm, failure alerts and interactive maps of pile layouts.
- An OCR pipeline with configurable regex rules and optional LLM refinement for extracting invoice and delivery fields from photos.
- Automated DWG to DXF conversion and parsing to identify pile IDs, types and coordinates, with a manual review interface for corrections.
- Role-based WebAuthn authentication (Administrator, Supervisor, Reporter) ensuring secure, tailored access.
- Manual-entry pages for hammer counts and incident logs, preserving audit trails.

Rigorous testing (unit and integration) ensured >95 % accuracy in OCR and parsing, <1 s dashboard latency, and 100 % RBAC enforcement. Containerized with Docker and orchestrated via Terraform, the solution scales horizontally and supports zero-downtime updates.

This integrated platform turns manual pile supervision into a data-driven, reliable process, saving time, reducing errors and empowering all stakeholders with real-time project visibility.

Future work includes adding new pile types and metadata extraction, enhancing DWG parsing with legend inference, and strengthening sensor security via mTLS.

Contents

Chapter 1

Introduction

In 2020 alone, the construction industry reportedly lost $1.84 trillion due to poor data management [1]. One of the key factors contributing to such losses is inefficiency on construction sites, which directly impacts project duration and cost. In pile driving operations, efficient and accurate data collection is essential for maintaining quality control and project oversight. However, much of this data is still collected manually on-site, increasing the risk of human error, inconsistencies, and incomplete documentation. These inefficiencies are further compounded by the time construction professionals spend managing data: a study by Autodesk reports that managers and executives spend an average of 11.5 hours per week searching for and analyzing project information [2].

To address the challenges of inefficient data management in pile driving operations, our team collaborated with Van Dijk Heitoezicht B.V. to develop a digital monitoring platform. This report outlines the design and implementation of that platform, with a focus on how it improves data collection, processing, and visualization in the field. The system is built using a modular microservices architecture and includes a React-based frontend that provides real-time dashboards, interactive maps, and role-specific views. Key features include robust optical character recognition of delivery receipts to digitize any kind of field, automated sensor data delivery, and the extraction of pile information (IDs, types, and coordinates) from engineering documents.

This report is structured as follows: Section 2 presents the problem analysis, including stakeholders, requirements, and risks. Section 3 describes the implemented solution, covering both the web application and sensor processing components. Section 4 discusses how we ensured the system's quality through testing, tooling, and stakeholder feedback. Ethical considerations related to data collection and automation are addressed in Section 5. Finally, Sections 6 and 7 describe our development process and potential future improvements.

Chapter 2

Problem Analysis

This chapter analyzes the key challenges faced by Van Dijk Heitoezicht in supervising pile-driving activities on Dutch construction sites. Although pile foundations are essential due to unstable soil conditions, the current workflows for data collection and reporting remain manual, fragmented, and error-prone. This leads to inefficiencies, limited data accessibility, and delayed decision-making for all stakeholders involved.

We begin by describing the core problems in the existing workflow (Section 2.1) and identifying the main stakeholders and their needs (Section 2.2). We then review existing tools and their limitations (Section 2.3), outline the requirement analysis process and functional priorities (Section 2.4), and conclude with a feasibility and risk assessment (Sections 2.5 and 2.6).

2.1 Problem Description

Van Dijk Heitoezicht faces four major efficiency challenges in their current pile-driving supervision process: **manual data entry**, **fragmented documentation systems**, **lack of real-time progress tracking** and **poor data accessibility for stakeholders**. These issues significantly impact the quality, reliability, and accessibility of project information.

As an independent supervisor of pile-driving operations across the Netherlands, Van Dijk plays a crucial role in ensuring safe and compliant foundation work. Despite the technical complexity and importance of these activities, the current supervision and documentation process remains largely manual and fragmented. Supervisors and workers rely on a combination of Word documents, Excel sheets, and even handwritten notes to track daily operations. This includes recording ground resistance, the number of hammer hits per 250mm for prefabricated piles, and the amount of concrete used in cast-in-place piles. These repetitive and error-prone processes consume valuable time and distract workers and supervisors from more critical, on-site responsibilities.

In addition to these technical measurements, supervisors are also tasked with documenting delivery information such as delivery reference numbers, material quantities, and pile types, which may be relevant to both internal reporting and client transparency. However, this data is often found in separate sources or not systematically recorded, which makes it difficult for stakeholders to access or analyze relevant project insights in a timely manner.

Another area of inefficiency is the tracking of pile completion progress. On most sites, a physical pile plan is used, where piles are manually crossed off as they are completed. While this approach has worked traditionally, it lacks flexibility and real-time visibility. Any updates or errors must be manually corrected, and it is difficult to gain a clear overview of the current project status or potential issues.

The underlying issue across all of these tasks is the decentralization of project information. Different parties (supervisors, clients, engineers, and site workers) have different

needs, yet often depend on disjointed systems to gather and interpret critical data. This fragmentation increases the risk of communication gaps, delays in reporting, and inconsistencies in documentation.

Addressing these problems would not only reduce the time and effort required from supervisors and site workers but also improve the quality, reliability, and accessibility of project data for all stakeholders. A more streamlined process allows for faster decision-making, clearer communication, and a more accurate understanding of project progress and challenges.

2.2 Stakeholders

Understanding the needs of key stakeholders is essential for designing a system that addresses the real-world challenges of pile-driving supervision. Each group interacts with project data in different ways, and their specific issues shaped the core functionality and priorities of our solution.

The primary stakeholders are the pile-driving supervisors at Van Dijk Heitoezicht which are contracted as specialized supervisors. They are responsible for overseeing the pile-driving process from start to finish, ensuring piles are constructed according to technical and regulatory specifications. Their tasks include measuring ground resistance (calendering), recording pile data, managing incidents, coordinating with various parties, and maintaining daily logs. These activities are currently handled manually across fragmented systems, increasing workload and risk of errors. Supervisors would benefit from a more reliable, efficient, and accessible way to collect and access data directly on-site, needs that have played a central role in shaping our system design.

Structural engineers form another key stakeholder group. They depend on accurate and timely field data to validate whether ground resistance aligns with pre-construction cone penetration tests. When discrepancies arise, they must make rapid technical decisions, such as adjusting pile specifications. Delayed or incomplete data can lead to misinformed decisions, project delays, or safety risks. Our solution aims to streamline access to up-to-date field data, supporting faster and more informed decision-making.

Project contractors are also heavily involved. As they oversee overall project execution, they require real-time insight into progress, such as pile completion, delivery records, and encountered issues. At present, much of this information is only available through delayed, end-of-day reports, limiting their ability to make timely adjustments to schedules and logistics. Our system provides live data dashboards and improved data flow, enabling more effective planning and execution. As Van Dijk Heitoezicht's core value lies in providing trustworthy supervision, our solution emphasizes data integrity, traceability, and system reliability.

More broadly, local municipalities and regulatory bodies may depend on documentation compiled by supervisors to confirm compliance with building codes and safety standards. Inaccuracies or data loss due to system failure could impact not only project approval but also the company's reputation and legal standing.

Since Van Dijk Heitoezicht's core service is professional supervision and has the responsibility of ensuring accurate and continuous data for quality on construction sites, the reliability of their tools is essential. If reporting fails or data is incomplete, the company risks reputational damage and potential loss of future contracts. Ensuring a high level of reliability, usability, and data integrity is therefore critical to supporting both day-to-day operations and long-term business continuity.

2.3 Existing Solutions

2.3.1 Existing system for tracking pile driving

An existing solution relevant to pile-driving supervision is the Pile-Driving Companion. This application allows supervisors to manually input measurement data on-site, ultimately exporting it as a CSV file, and also includes basic audio recording capabilities to detect pile-driving events using a phone microphone [2]. However, this solution has notable limitations: the output is limited to simple CSV files, which require significant additional processing before the data can be integrated into project reports or shared meaningfully with stakeholders. Additionally, it does not support the extraction of critical data, such as delivery details or pile plan information, which remain manual and fragmented tasks. In addition, using a phone microphone for measurements can cause accuracy issues, as construction sites often have significant background noise, potentially compromising data reliability.

2.3.2 Existing system for construction site dashboards and reports

Another product, FieldWire, is a broader project management solution that offers real-time dashboards and PDF or DWG file handling. Although FieldWire provides robust general management tools, it lacks specific functionality tailored for pile-driving tasks, such as sensor-based data recording and automatic incident logging relevant to pile-driving activities. Its generic design means that it does not adequately address the unique technical and regulatory requirements faced by pile-driving supervisors. Furthermore, general-purpose applications may include features that supervisors find cumbersome or unnecessary, increasing complexity and the likelihood of user errors.

2.4 Requirement Analysis

2.4.1 Requirement Elicitation Process

In the first week, we met with the client to define the application's main objectives, success criteria, and stakeholder needs. This was followed by two site visits, where we observed the pile-driving process, gathered example field data, and aligned on the desired system features. These insights informed both the scope of our solution and the design of early detection and visualization components.

To ensure traceability and regulatory compliance, we documented each requirement in a centralized backlog with version control. We held bi-weekly review sessions with the coach to validate our interpretations, refine ambiguous requirements, and incorporate feedback on error tolerances, data privacy, and performance benchmarks.

2.4.2 Solution Strategy and Functional Requirements

To ensure the solution directly addressed the core challenges identified in the existing workflow (manual documentation, fragmented systems, lack of real-time updates, and poor data accessibility) we worked closely with the client to define a realistic and impactful system scope.

Given the team's limited experience with embedded hardware and the unpredictable nature of site visits, we agreed to focus first on delivering a functional and modular web

application. This would form the foundation of the system, providing immediate value while allowing for future sensor integration.

The application would serve as a central platform for supervisors and stakeholders to interact with project data efficiently and reliably. Based on stakeholder input and field observations, we defined the following core capabilities and key requirments:

- **Real-time monitoring dashboard:** A live interface displays key pile-driving metrics, such as hammer blows per 250 mm and failure alerts,with updates pushed via WebSockets every 5 seconds. This directly addresses the need for up-to-date insight into field activity and reduces dependence on manual status reports.
- **Interactive pile plan parsing:** Users can upload DWG or PDF drawings of pile layouts. The system automatically extracts pile IDs, types, and coordinates, which users can verify and correct in an intuitive interface. This supports faster preparation and progress tracking while minimizing manual data entry.
- **Flexible document OCR:** A configurable OCR tool extracts fields (e.g., bon numbers, material quantities) from scanned delivery receipts or reports. This replaces error-prone manual transcription and supports various document formats through tunable extraction rules.
- **Secure and scalable backend:** The backend includes WebAuthn-based login, role-based access control, and fast, permission-checked APIs for managing projects, users, and annotations, ensuring secure and responsive system behavior.

By aligning the functional design with specific stakeholder needs and project constraints, this solution not only improves current workflows but also consists the basis for future enhancements, such as automated anomaly detection based on sensor input.

2.5 Feasibility Study

2.5.1 Feasibility

The ten-week timeline (April–June 2024) was designed to balance frontend, processing, and hardware tasks. In weeks 1–3, we delivered the basic user interface and showed our concept identification algorithms using phone and inertial recordings. During weeks 4–7, we focused on creating the OCR pipeline, DWG/PDF parsing capability, and finalizing the web application's feature set, while establishing clear integration points for sensor modules.

Week 8 concentrated on integrating and validating our initial sensor prototypes in a controlled environment, simulating site conditions. Week 9 was dedicated to full end-to-end testing of data flows, performance optimization, and deliverable preparation. Week 10 will wrap up final stress testing, stakeholder training sessions, and compliance verifications against building regulations.

2.5.2 Available Data from Construction Site

At project start, we relied on recordings made during our two site visits: high-fidelity audio and video acquired on a smartphone, as well as inertial sensor logs approximating

the vibrations and movements that the final ESP32-based devices[2] would record. These samples enabled early training and validation of our anomaly detection routines and helped tune our preprocessing pipeline.

Furthermore, the customer provided CAD[3] drawings in DWG and PDF formats, which served as the foundation for our plan-parsing interface. We examined multiple drawing versions to understand layering conventions, coordinate systems, and annotation practices, informing our parsing logic and error-correction UI design.

2.5.3 Changes in Case the Project Is Infeasible

We, along with the client, knew from the beginning that properly integrating live sensor data would be difficult due to anticipated hardware limits, access constraints, and performance considerations. Given these uncertainties, we actively developed alternative methods and held discussions with our stakeholders to clarify expectations and explore potential solutions. As a result, we created thorough documentation and algorithms for anomaly detection, preparing to analyze manually provided audio and inertial data via CSV imports.

Finally, we demonstrated partial functionality by successfully integrating a single ESP32 sensor into our platform, which can transmit inertial data and count the number of impact occurrences. This solution effectively acted as a proof of concept, indicating the possibility for extended, fully integrated sensor capabilities in future versions.

2.6 Risk Analysis

We identified many important risks to the successful conversion of prototype recordings to actual sensor data from the start of the project, including background noise, fluctuating ground conditions, and low signal-to-noise ratios, which could reduce algorithm performance. To address these risks, we modified preprocessing parameters and personally tested various anomaly detection algorithms, ranging from isolation forests to dynamic sequence-based methods, to determine optimal performance under different kinds of noise situations. In addition, we used adaptive filtering techniques and improved inertial-only detection methods to provide reliable data processing.

Parsing DWG proved difficult because of custom formatting and coordinate inconsistencies. [3] We addressed these concerns by using a two-stage parsing approach: an initial automatic extraction followed by supervisor-assisted rectification procedures, ensuring data quality and compliance to site plans.

We carefully evaluated and adapted numerous OCR configurations to improve performance, taking into account document quality, lighting circumstances, and template diversity. Additionally, we provided customers with the option of using LLM assistance to identify and recover lost data as needed, resulting in high precision values.

Throughout the development process, our client frequently proposed new feature requests or user interface changes that differed from the initially agreed-upon designs. Each time, our team worked actively with the client, having dedicated talks to clearly explain our original choices and demonstrate why the initial design decisions best fit the project's functional and usability needs.

[2]See the official product page at `https://www.espressif.com/en/products/socs/esp32`

[3]Computer-Aided Design

At the end of each development week, we held careful progress review meetings with the client. These meetings served not only to display our work and obtain iterative feedback, but also to expressly explain which activities could be completed within the time frame provided. A notable example came at the conclusion of week seven, when multiple requests for changes came out as a result of a basic misunderstanding of previously agreed-upon functionality and expectations. Our team carefully addressed each point of doubt, extensively describing the current implementation state, justifying some design decisions, and outlining why certain requested changes were either unneeded or impracticable given the remaining timetable. This complex conversation cleared up any ambiguities and matched the client's expectations with the project's true scope and objectives.

Chapter 3

Description of the Implemented Solution

This chapter details the software solution we developed to address the challenges outlined in Chapter 2. It connects the project's initial requirements with the implemented software solution, explaining how each design decision addresses a specific challenge in pile-driving supervision. Our goal was to design and implement a scalable, modular, and user-friendly platform that supports pile-driving supervision by enhancing data collection, centralized documentation, and real-time project monitoring.

We begin by presenting the overall architecture of the system, including the technologies chosen for the frontend, backend, and database, as well as the reasoning for adopting a microservices-based design. The system-level diagram provides a high-level overview of the platform's infrastructure, illustrating the main components and their general data flow without getting into low-level technical details.

Subsequent sections describe the core functionalities of the web application, including manual data entry, dashboard visualization, role-based access control, document OCR, automated reporting, and interactive pile plan management. Each component is developed to meet specific stakeholder needs, while ensuring scalability, security, and flexibility for future enhancements.

3.1 Outline of The Architecture

In this section, we discuss the technologies used for the frontend and backend, the database solution, the architectural design pattern, and the system design that incorporates all of this. We describe the overall architecture of the web application that we developed.

3.1.1 Selecting Technologies for the Web Application

Making the right technology choices is crucial to support a scalable, maintainable real-time monitoring system with a user-friendly interface for pile-driving supervisors, engineers and clients. Table 1 compares our leading frontend framework choices against the most important criteria for our project, while Table 2 summarizes our backend platform evaluations.

For the frontend, we required a responsive, modern UI that remains easy to maintain and test as the project evolves. ReactJS excelled across component architecture, ecosystem maturity and testing support[4], that is why it was chosen as our primary framework.

To link the frontend and backend we needed a reliable, scalable layer that could deliver real-time data to many users. We compared three candidates: Node.js with Express,

[4]According to the 2024 Stack Overflow Developer Survey, React was the most popular front-end framework with 39.5% usage among professional developers (n $\geq$ 65,000). See `https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190`.

Criteria	ReactJS	Angular	Vue.js	Svelte
Component Architecture	Excellent	Excellent	Good	Good
Learning Curve	Fair	Poor	Excellent	Excellent
Ecosystem & Resources	Excellent	Excellent	Good	Fair
Testing Support	Excellent	Good	Good	Fair
Industry Adoption	Excellent	Good	Good	Fair

Table 1: Frontend Framework Comparison

Criteria	Node.js + Express	Python + Flask	Go
Single-Language Stack	Excellent	Poor	Poor
Concurrency (real-time I/O)	Excellent	Poor	Excellent
Learning Curve	Good	Excellent	Good
Ecosystem	Excellent	Good	Fair
Community & Resources	Excellent	Good	Good
Testing Consistency	Excellent	Poor	Fair

Table 2: Backend Platform Comparison

Python with Flask and Go (see Table 2). We settled on using RESTful APIs built with Node.js/Express. Its event-driven, non-blocking I/O works really well with high workloads, while using the same programming language on both frontend and server side lets us reuse validation logic, integrates smoothly with React and keeps our test suite simple.[5]

Additionally, we use Docker to containerize the application for easier deployment and scalability.[6]

3.1.2 Selecting the Database

For the database we chose PostgreSQL because it provides a reliable and scalable solution that would work great for our project. Additionally, PostgreSQL has a really large and strong community and is widely adopted, which ensures available resources and long-term maintainability. Moreover, the official PostgreSQL Docker image simplifies deployment a lot, aligning with our goal of containerized, scalable services.

We also considered MongoDB, but its document-oriented design has no native SQL, which complicates complex joins and ad-hoc queries; therefore, we excluded it.

Also, it is a great decision because it supports ACID transactions and robust constraint

[5]According to The Node.js Performance Report (2024) by Rafael Gonzaga, Node.js efficiently processes real-time, concurrent RESTful API requests and integrates seamlessly with JavaScript front-end frameworks like React. See `https://downloads.ctfassets.net/hspc7zpa5cvq/7rCLml0etL4snP0MnX2q62/929dfdc419fcf4189a4220e85d991dfd/The_Node.js_Performance_Report.pdf`.

[6]Docker has become an industry-standard container platform, ensuring consistent environments from development through production—making it a must for modern, scalable applications. See `https://www.docker.com/why-docker`.

enforcement. Its advanced features, such as JSONB support, also give us flexibility to store less structured data, which might be needed for the sensor data. A schema for our database can be seen in Figure 1.[7]

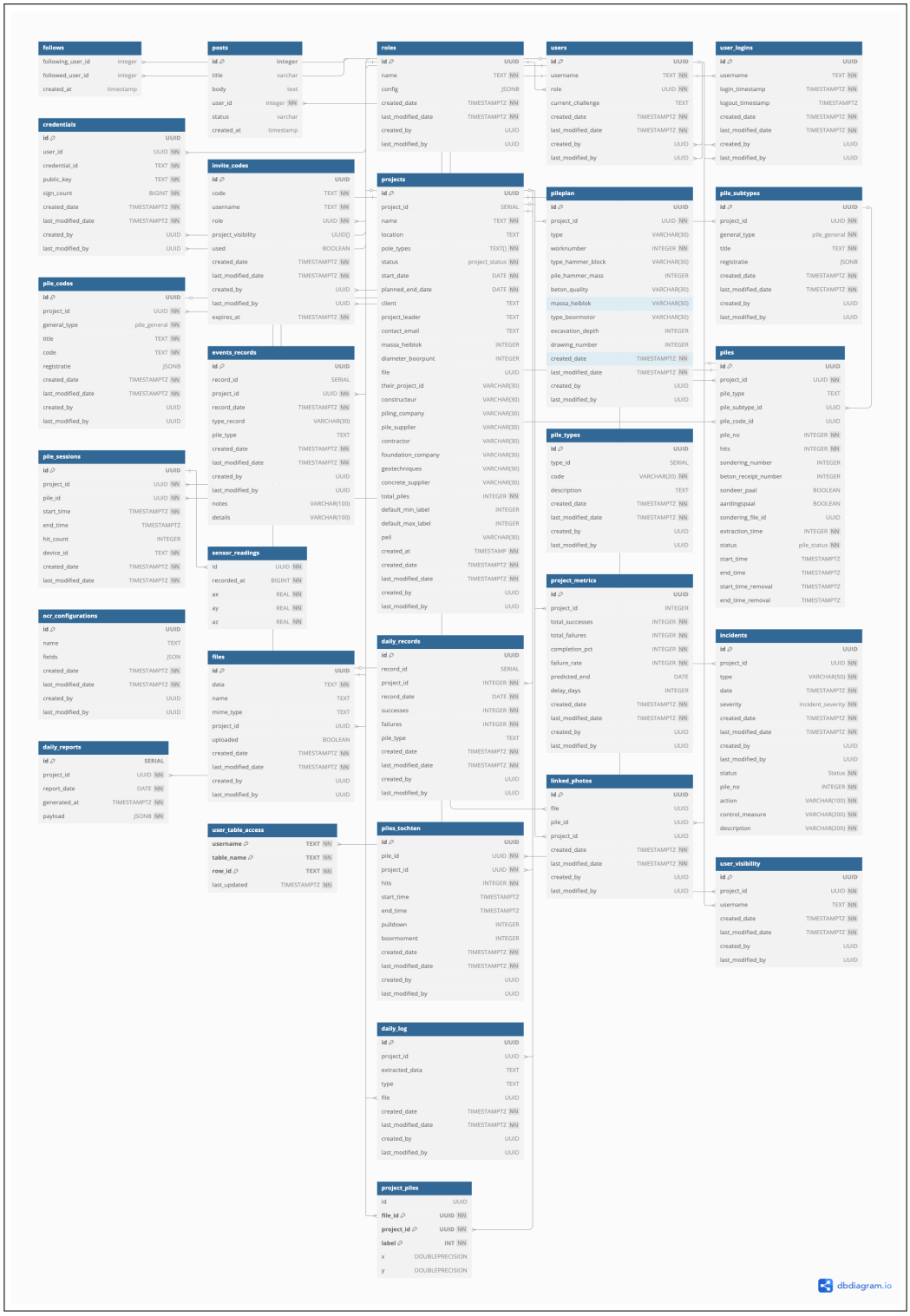


Figure 1: Database visual diagram

[7]The DB-Engines Ranking (June 2025) places PostgreSQL as the fourth most popular database system with a positive growth trend compared to MySQL and MongoDB, see full ranking charts at `https://db-engines.com/en/ranking` and trend graphs at `https://db-engines.com/en/ranking_trend`.

3.1.3 Selecting the Architecture Pattern

Since we had to combine different components to make this application, we needed an architecture that supports modularity, independent development, and easy scaling. That is why we adopted a microservices architecture, which breaks down the complex system into smaller services. This ensures that we can develop, test and deploy individual components independently.

We decided to use Docker containers to isolate services. This containerization is contributing to scalability since the services can be replicated or edited without affecting the whole system. Docker also helps us to manage microservices infrastructure by making different environments more consistent and by making the deployment process simpler. Moreover, because failures do not propagate throughout services, it also improves fault tolerance. In week 5, we also introduced Terraform as our infrastructure-as-code platform, which allows us to explicitly provision and scale only the services that require additional capacity, thus enhancing our ability to grow and manage the system with minimal manual effort.

3.1.4 Final System Design

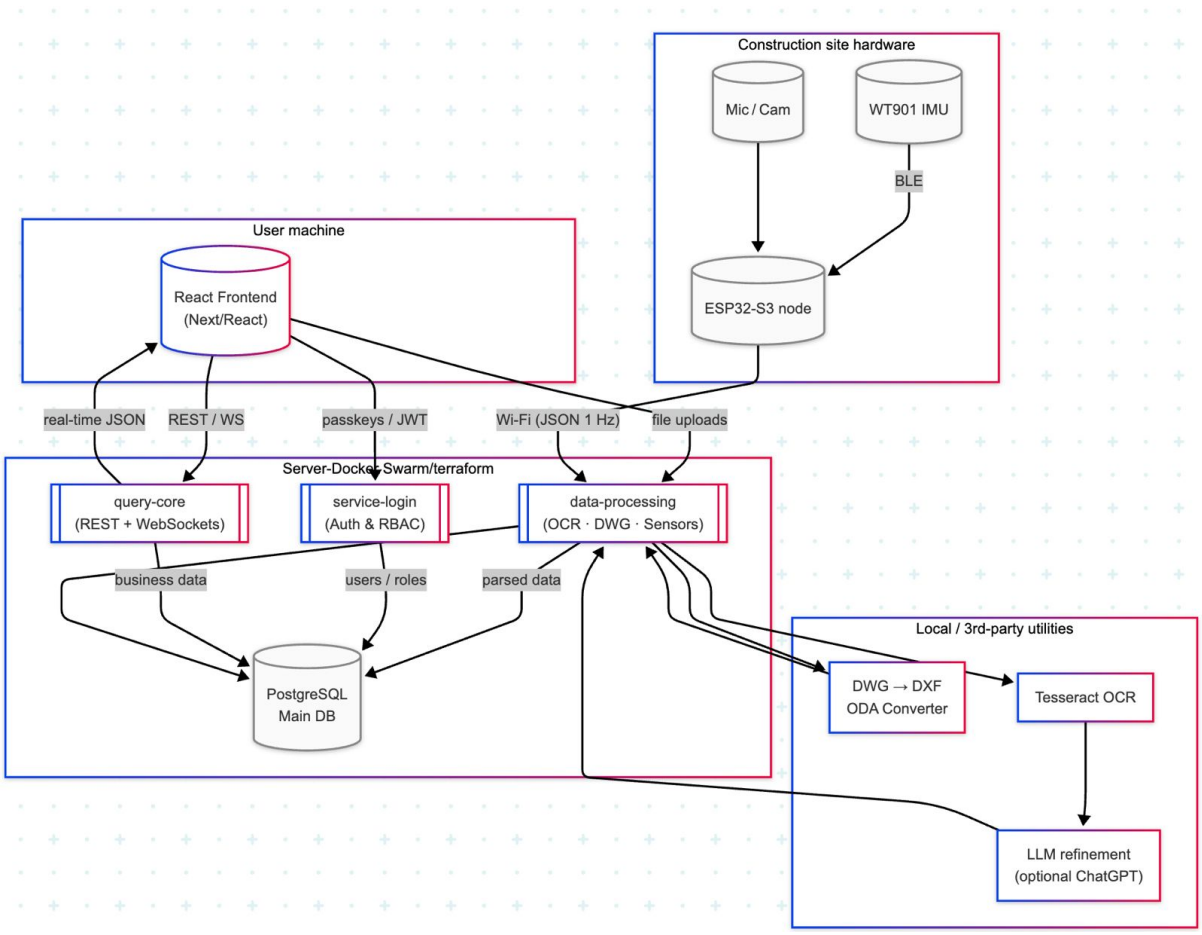


Figure 2: System-design layout for the pile-driving monitoring platform. Boxes denote distinct components; arrows depict the data flow.

The figure illustrates the complete pile-driving monitoring platform. The architecture matches our `docker-compose` file and is organised into four cooperating blocks whose boundaries align with individual containers: *field hardware, four micro-services, a database container, and the React front-end.* Each arrow in the figure represents an actual message path in the running system.

- **Construction-site hardware.** A robust ESP32-S3 gateway is connected to an inertial sensor mounted on the concrete pump or, optionally, to an audio sensor located near the machine and receives 200 Hz acceleration data (or audio frames) over BLE. The gateway timestamps the most recent samples every second, wraps them in a compact JSON payload, and sends the result to the backend via Wi-Fi. If no trusted SSID is identified, the ESP establishes its own access point, allowing personnel to enroll recent credentials on-site.

- **Back-end services (Swarm / Terraform).** Four stateless containers implement all server logic:

  `service-login` Performs WebAuthn passkey registration, issues JWTs, and enforces RBAC in 20 ms per request.

  `query-core` Exposes public REST endpoints and WebSocket hubs that power the live dashboards.

  `data-processing` Ingests sensor JSON, DWG/PDF uploads, and receipt images; runs OCR, DWG $\rightarrow$ DXF parsing, and anomaly detection before writing normalised artefacts back to storage.

  `frontend` A React server-side-render process that serves the compiled React bundle, giving faster first paint and SEO benefits.

  All cross-service traffic is JWT-signed HTTP; long-running tasks (OCR, DWG conversion) are dispatched asynchronously but results always return through `data-processing`, so upstream code remains independent of the parsing engine.

- **Persistence containers.**

  – A dedicated **PostgreSQL** container stores structured entities (projects, piles, hits, receipts, audit log). JSONB columns capture semi-structured payloads such as OCR results or plan geometry.

- **Auxiliary parsing utilities.** `data-processing` shells out to a local *ODA Converter* for DWG $\rightarrow$ DXF and to *Tesseract OCR* for receipt text. Low-confidence fields may be cleaned by an optional LLM refinement step, reaching $\geq 95\%$ accuracy without manual templates.

- **End-to-end data flow.**

  1. *Capture* – the ESP32 timestamps each reading and streams 1 Hz JSON to `data-processing`.
  2. *Validate & enrich* – the service attaches project context, writes the sample to PostgreSQL, and notifies `query-core` so dashboards refresh in under five seconds.

3. *Document parsing* – a supervisor uploads a DWG or receipt; the corresponding helper converts/reads it and merges the extracted metadata into the pile record.
4. *Visualise* – the React front-end holds a WebSocket open for live counts, uses REST for heavy queries, and sends file uploads directly to `data-processing` with a progress bar.
5. *Report generation* – on request the front-end orchestrates `report-maker`, which combines relational data with stored attachments and streams a 20-page PDF back to the browser in roughly three seconds.

All containers are declared in Terraform modules and updated via a rolling-update strategy, guaranteeing zero downtime. Because every service is stateless and shares the same JWT/RBAC layer, additional replicas can be scheduled automatically when multiple construction sites operate in parallel, leaving the sensor firmware and front-end untouched.

This layered yet loosely-coupled design achieves the project's key goals: sub-second dashboard latency, secure passkey log-ins, high-accuracy document parsing, and effortless horizontal scaling.

3.2 Web Application

3.2.1 Manual recording and processing of data from the sensors

In many cases, site supervisors still require a fallback to manual data entry when sensor hardware is unavailable or being tested. Our application therefore provides a dedicated interface on the Piles page where users can enter the total number of hammer strokes, as well as the count per each 250 mm section, exactly as they would on paper. Similarly, any unexpected incidents, such as equipment malfunctions or ground anomalies can be logged directly from the Incidents page, ensuring that no event goes unrecorded even if automatic detection is offline.[4]

To link operational data with logistics and documentation, users may upload or attach purchase receipts, delivery notes, or site pictures of specific events directly to a pile record. Uploaded pile-plan files (PDF or DXF) are associated with the project, and the UI allows supervisors to update each pile's status, start and stop a live timer for the drilling process, and correct any automatically-detected parameters. All manual entries and attachments are versioned in our audit log, providing full traceability across the project's lifecycle.

Behind the scenes, as sensor data becomes available, it is sent in real time from the ESP32 units to our PostgreSQL database. When a user clicks "View Hits" for a certain pile, the backend obtains the raw time series and performs our anomaly detection and peak-counting algorithms. The algorithm returns both a table of counts and a rendered graph in under four seconds.[5] This hybrid approach guarantees that supervisors always have accurate, up-to-date information, whether they rely on automated sensors or manual entry. For more details, we added the file in which we documented our research in the Appendix .

3.2.2 Dashboard Visualization and Aggregation

One of the central components of our pile-driving portal is the dashboard system, which serves as the main interface for monitoring progress, managing data, and responding

to incidents. It is designed to support both high-level project aggregation and detailed project-specific insights. This section provides a structured walkthrough of its key elements.

Architecture and Structure

The dashboard follows a project-oriented architecture. While organization-wide oversight is supported through aggregation views, the majority of the system's functionalities are scoped per project. As such, the dashboard is conceptually divided into two main layers:

- A *General Dashboard*, providing statistics and summaries across all projects.
- *Per-Project Dashboards*, focusing on the detailed status and controls for each specific project.

General Dashboard Overview

The first interface users typically see is the *Projects List*, which summarizes platform-wide activity. It displays cards showing the number of total and active projects, pile-related counts, emergency tallies, and the overall progress percentage. Below this, a progress bar visualizes total pile completion (e.g., *3 of 210 piles placed – 1% progress*).

A searchable and sortable table provides further project details: name, location, pile types, project status, and a small progress bar. Each entry includes a `Details` button that links to the corresponding per-project dashboard.

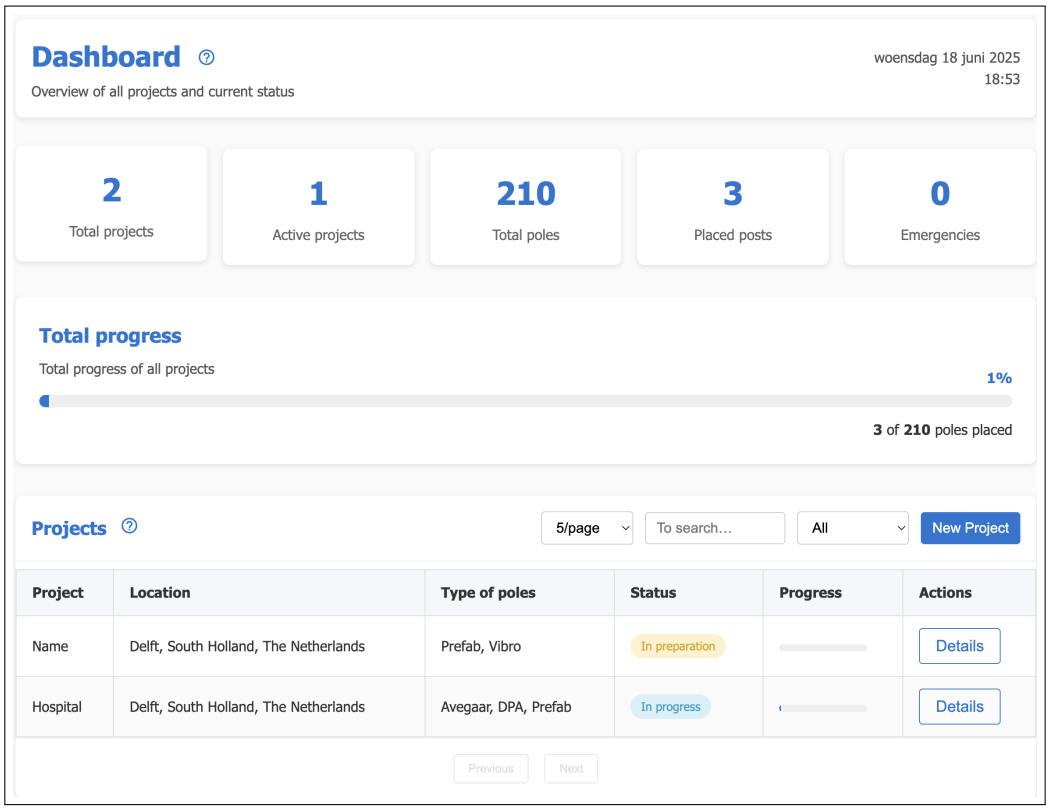


Figure 3: Projects List overview.

Aggregated Visual Summaries

To help users quickly interpret trends and distributions, two visualizations are provided: a donut chart showing the distribution of pile types, and a bar chart highlighting the frequency of recent calamities. These charts are placed at the bottom of the Projects List view and update dynamically.

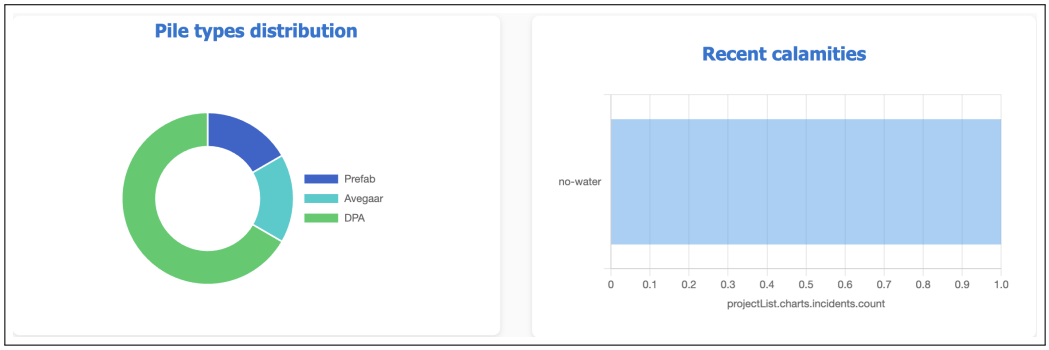


Figure 4: Visual aggregation charts for pile type distribution and incidents.

Activity Logs and Events

The *Recent Events* page presents detailed logs, beginning with an activity summary and followed by a table of recorded events. Each entry includes a timestamp, project ID, event type (e.g., "Pile added"), and optional notes. A secondary table aggregates this data per day, making it easy to compare site activity across projects.

Per-Project Dashboards

Each project has its own dashboard containing contextualized data, project metadata, and operational tools. The header section includes the project name, location, status, and shortcut buttons to key subpages such as `Incidents`, `Piles`, `Map`, and `Weather`.

Summary cards display project metadata (client, dates, contact) and progress statistics (piles placed, timeline percentage, delays). Two circular meters indicate completion and failure percentages, which update in real time.

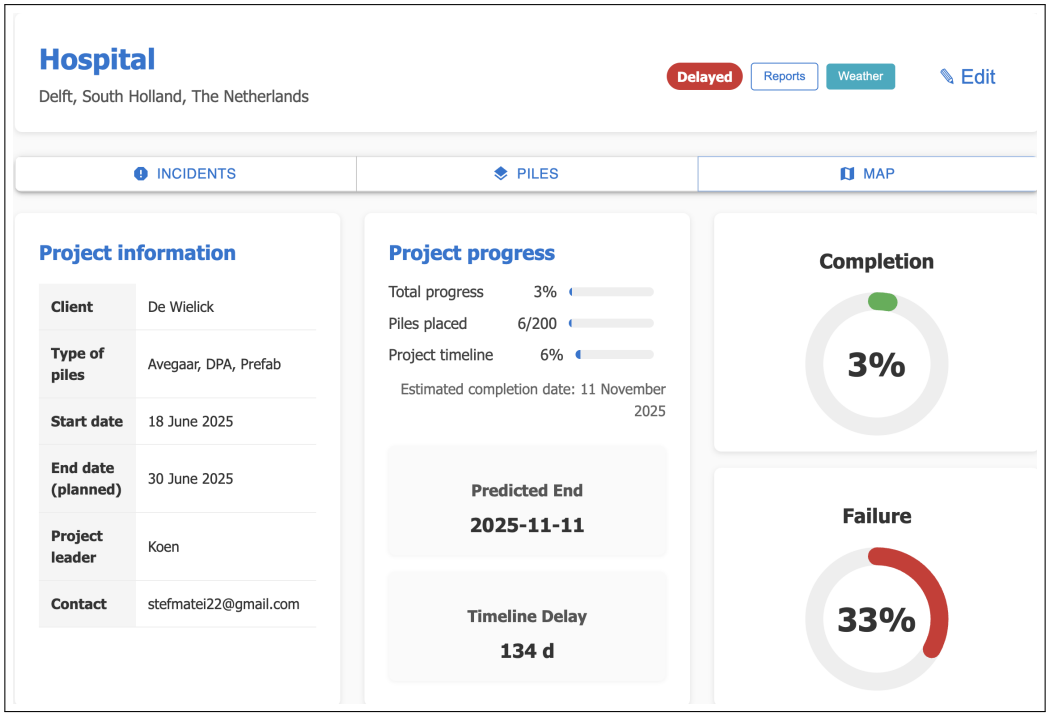


Figure 5: Individual project dashboard showing metadata and progress.

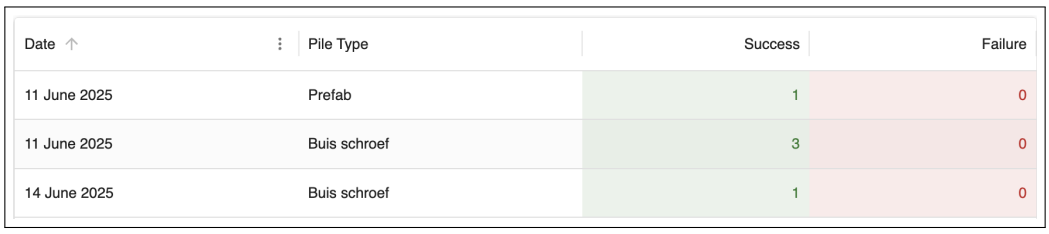
Daily Records and Charts

Below the summary, a daily log lists piles placed each day, grouped by type and success/-failure status (green for success, red for failure).

Interactive visualizations below this table include:

- **Daily Productivity:** bar chart of successful vs. failed piles per day.
- **Pile Status Distribution:** pie chart of current pile states.
- **Pile Type Breakdown:** pie chart of pile type usage.
- **Production Trend:** line graph showing success and failure rates over time.

All graphs are expandable to fullscreen, enabling better display on mobile devices or in meetings.



Date	Pile Type	Success	Failure
11 June 2025	Prefab	1	0
11 June 2025	Buis schroef	3	0
14 June 2025	Buis schroef	1	0

Figure 6: Daily records information

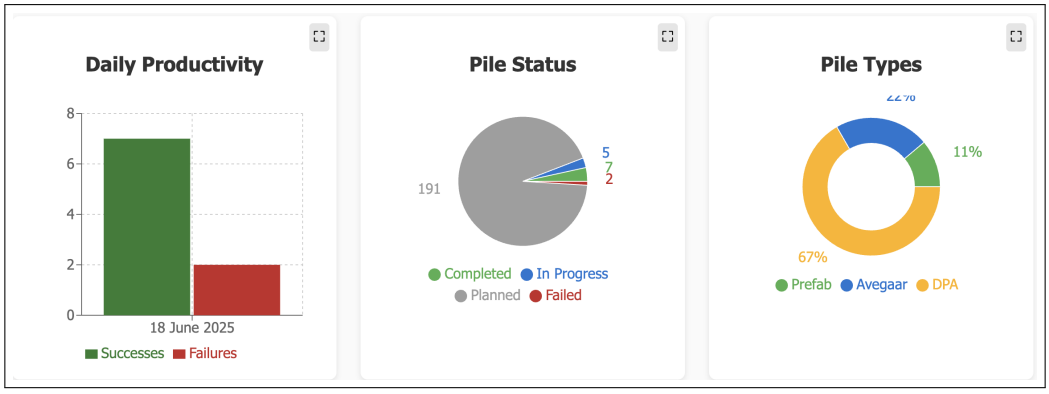


Figure 7: Project information charts

Incident and Pile Management
The `Incidents` page summarizes all reported calamities, showing totals and resolved percentages. Users can filter entries and log new incidents with details such as pile ID, severity, actions, and descriptions. Charts visualize incident counts by severity and trendlines over time.

On the `Piles` page, each pile is listed with details like ID, type, timestamps, and current status. This allows for granular pile-level tracking and historical inspection.

Overall, the dashboard module serves as the operational hub of the supervision platform. Through a combination of visual summaries, searchable tables, and interactive charts, it empowers users to monitor performance, detect issues, and make timely decisions across all projects.

3.2.3 Role-Based Access Control

Our application uses WebAuthn-based authentication[8] and a Role-Based Access Control (RBAC) [6] system within the service-login microservice to ensure that each user only sees and updates functionality that is relevant to their role. All permission checks take under 20 ms, and unauthorized requests are refused with 100% accuracy. This architecture isolates authentication and authorization functionality from the rest of the platform, making it easy to add new roles or change permissions without affecting essential business functions. WebAuthn uses public-key credentials to eliminate the need for traditional passwords. Users can register platform authenticators, sync credentials via Google, or use hardware keys like YubiKeys[9] to log in seamlessly across desktop and mobile devices.[7]

The definition of three separate roles: **Administrator**, **Supervisor**, and **Reporter** is precisely aligned with the system requirements acquired during the first elicitation phase. These roles were defined in response to the client's feedback and functional requirements, with the goal of clearly and securely managing data access and permissions across many user groups:

– **Administrator:** They have full system access, which means they can see every menu item and control, including the user and project management screens. Admin-

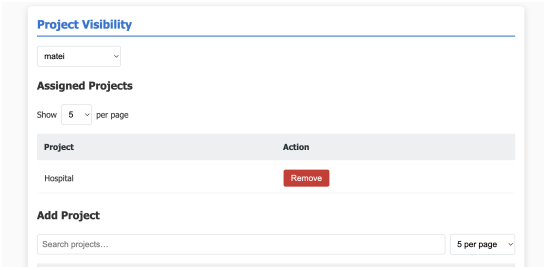
[8]WebAuthn is a W3C Recommendation for passwordless, phishing-resistant authentication based on public-key cryptography; see `https://www.w3.org/TR/webauthn/`.

[9]YubiKey is a hardware security key supporting FIDO2, U2F, OTP, and smart card protocols, providing phishing-resistant, passwordless authentication; see `https://www.yubico.com/products/yubikey-hardware/`.

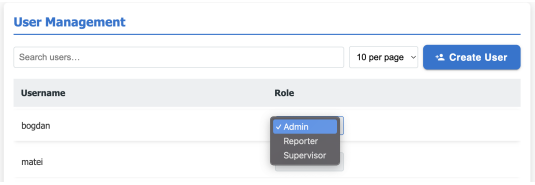
istrators can use the "Users Managament" page to assign or revoke responsibilities, limit individual users access to certain projects and create new user accounts with tailored permissions, sending invitation links via email, QR code, or on-screen display (see Appendix C.1). An audit-log viewer allows them to view every login event, authorization change, and data modification across all projects. They can also add or remove projects, modify global settings, and handle system-wide integrations.

– **Supervisor:** Scoped for one or more projects (assigned by an Administrator). Supervisors have complete read/write access to their projects, which allows them to start and stop live sensor streams, correct auto-extracted pile plans, register incidents, attach receipts or images, and update pile statuses. Administrative menus (user roles, global settings, and audit logs) are concealed so that they may concentrate on on-site data collecting and project execution.

– **Reporter:** Access to information is read-only. Reporters only see dashboards and export utilities like interactive charts, progress meters, and daily log tables. They can receive project performance summaries in PDF format, but all data entry and management features are disabled or hidden to prevent unintentional changes.



(a) Project visibility dashboard for each role.



(b) User management interface showing role assignments.

Figure 8: Role-based views in our service-login microservice.

By centralizing RBAC in a single, performant microservice, UI components and API endpoints automatically adapt to each user's role. This approach streamlines development and auditing while providing flexible, fine-grained control over who can view or modify any piece of project data.

3.2.4 Image Data Extraction and Analysis

Supervisors frequently take pictures of paper receipts, delivery notes, purchase invoices, and site permits, with their phones, but this vital information is lost in unorganized photos. Our OCR pipeline automates and standardizes image-based data extraction, including decoding, preprocessing, field-level parsing, and post-processing. This solution combines dual-OCR scans, dynamic regex-driven parsing, and operator-assisted overrides to handle diversity in receipt layouts, lighting conditions, and fonts without using fixed coordinate templates.[8]

Images are first uploaded to one of our internal routes as Base64-encoded texts. The server decodes each string into a binary buffer and simultaneously launches two OCR routines: 'ocrDutch' for full-text parsing and 'ocrDigitsBig' for high-accuracy digit extraction. We partition the task to accurately detect huge blocks of text, such as vendor

names, dates, and line-item descriptions, as well as six-digit order numbers (the "Bonnr"). If the raw text lacks an explicit "Bonnr:" line, we scan the aggregated results for a six-digit token and prepend it to ensure consistent field presence.

Our 'parseReceiptText' method extracts essential fields from raw OCR data using a customizable ruleset. A regular expression (with optional flags) is applied to non-empty trimmed lines for each defined field. Single-match fields, such as invoice numbers or delivery dates, use a capture group and optional post-processing code. Multi-match fields, like line-item tables, gather all regex matches into arrays of objects. This flexibility enables us to accept arbitrary vendor templates: new extraction rules can be implemented at runtime, without redeploying code, to handle new receipt formats or changing client requirements.

The 'ocrDutch' method automatically detects picture orientation using Tesseract's OSD mode and rotates the buffer accordingly. We use the Sharp[10] library to perform picture modifications, including grayscale conversion, normalization, denoising, sharpening, thresholding, and DPI-aware scaling.[11]. These techniques reduce the skewed scans, low contrast, and noise found in field images. The preprocessed image is passed into Tesseract with optimized parameters (OEM 2, proper PSM, and language set to Dutch), creating high-fidelity text in under ten seconds on standard server hardware.[9]

In parallel, 'ocrDigitsBig' recovers numeric codes by focusing on a cropped portion of the receipt (the upper half) where order numbers and totals are most commonly found. After grayscale normalization and thresholding, we use Tesseract in single-line mode (PSM 12) with a specific whitelist of digits. This specialized run identifies digit-only strings with minimal false positives, capturing six-digit order numbers even in busy layouts.

The combined OCR results are standardized into a structured schema. JSON objects are mapping field names to values or arrays of values within two seconds. Parsed data is given to the client alongside the entire raw text, allowing for immediate UI display of both the extracted fields and the underlying receipt image. Supervisors can then examine and, if necessary, manually edit any fields using an inline override interface that saves changes while maintaining audit trails for consistency.

Finally, all image data and extracted information are saved in our PostgreSQL database. Each OCR request, including the raw image, normalized text, and parsed data, is timestamped and assigned a user ID, allowing for full traceability and reversal. This approach not only automates ineffective manual entry, but it also works perfectly with downstream reporting, converting unstructured receipt photos into usable, dependable project data.[10] If you'd like to see more screenshots, please refer to Appendix C.2.

3.2.5 Automated Report Generation

One important feature of the application is the **automated report generation**, which compiles all project data into a structured PDF document. This report includes everything that happened during the project—from listing all the involved parties to daily

[10]Sharp is a high-performance Node.js image-processing library built on libvips, offering fast, efficient APIs for image resizing, format conversion, compositing and more; see `https://sharp.pixelplumbing.com/`.

[11]Tesseract OCR works best on images with at least 300 DPI and internally rescales source images to 300 DPI to maximize recognition accuracy; see `https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html`.

activity summaries and detailed tables for each pile type used. It is designed to be reviewed at the end of the project by managers, contractors, and all other key stakeholders.

This feature is a crucial part of the project as it provides a clear overview of the entire process. It serves as the final result of all the work done and makes it easy to verify and trace what occurred throughout the project.

The report is structured as follows:

- **Preface:** This section includes the project title, a photo of the finished structure, and a list of the key stakeholders. These include the client, constructor, main contractor, foundation company, geotechnics team, concrete supplier, pile supplier, and supervisor.
- **Daily Summaries:** Each project day has a dedicated summary. This includes the piles added on that day, any changes in the concrete mix, and incidents that occurred. At the end of each daily section, attachments such as concrete delivery slips and related photos are included.
- **Pile Data Tables:** For each type of pile used in the project, a table is generated with key properties such as diameter, concrete volume, number of hits, and other relevant data. The report currently supports three pile types:
  - **Prefab:** Includes an additional table showing the number of hits per 250mm.
  - **Avegaar:** Has unique properties specific to auger-based piles.
  - **DPA:** Includes extra data such as Pulldown Force (PD), Motor Force (BM), and an associated graph.
- **Sondering diagrams:** Sondering diagrams for DPA and prefab piles are graphical plots of the site's resistance parameters recorded during pile installation. For DPA piles, the diagram shows boormotor torque and pulldown force. For prefab piles, the diagram plots the number of hammer blows per 250 mm. These sondering diagrams serve to visualize the drive resistance profile, delineate subsurface soil layers, estimate pile bearing capacity, detect anomalies or obstructions during driving, and optimize pile design and quality control
- **Future Support:** We chose just 3 types of piles because they are the most used in the projects and they have lots of differences between them. The future work will include other types of piles also. Moreover, other data can be included into the daily logs and a more humanly language can be used.

  This report is done using React and jsPDF, html2canvas libraries and you can see images in AppendixC.9.

3.2.6 Pile Map

The interactive pile map module is implemented to visualize pile data extracted from construction plans and synchronize this information with the back-end pile record system. It operates as an important frontend interface that consumes parsed data from DXF files and displays a real-time overview of piling progress, offering both spatial accuracy and interactivity for project supervisors and engineers.

The current implementation relies on parsing DWG files, which are first converted to the DXF format using a local open-source conversion utility. This approach replaces

an earlier, image-based PDF parsing method, which only achieved 45–50% accuracy due to high visual density, inconsistent labeling, and schematic overlap. The DXF-based workflow provides a more structured and accessible format, enabling more consistent and reliable parsing. Across a variety of typical plan formats, this method achieves an accuracy of approximately 85%.

Following conversion, a custom DXF parser processes the file to extract both geometric and textual entities. The parser identifies and isolates relevant elements such as pile labels, coordinates, pile dimensions (width and height), and associated handlers. All non-essential drawing elements, including auxiliary construction lines, decorative elements, and unreferenced layers, are systematically excluded to reduce noise. The cleaned data is stored in an intermediate JSON representation, which is further refined to preserve only fields directly relevant to the frontend visualization and user interactions.

Using this refined JSON structure, the pile map is rendered in the frontend as a dynamic and scalable SVG interface. Each pile is drawn at its exact coordinate location with a visible label and standardized dimensions. The visual status of each pile is indicated by color-coding, updated in real time via the system's back-end database:

- **Gray** for planned,
- **Blue** for in progress,
- **Green** for completed,
- **Red** for failed, and
- **Black** for configured.

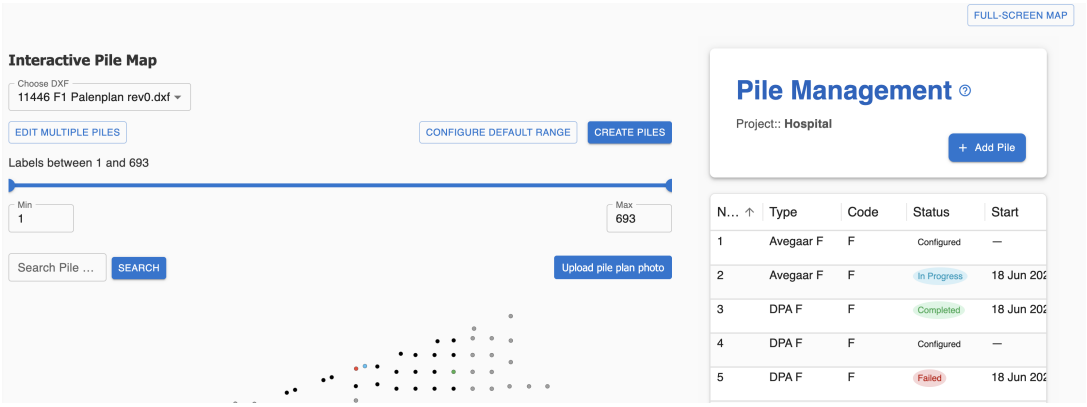


Figure 9: Rendered interactive pile map showing status colors, pile labels

These status values are dynamically retrieved from the project's database to ensure that visual feedback remains synchronized with field operations. This enables supervisors to assess pile execution status at a glance and monitor progress in near real-time.

To accommodate parsing inaccuracies or unconventional plan layouts, the frontend provides several editing tools. Users can interactively reposition piles via drag-and-drop, remove incorrectly placed items, or add new ones to empty areas on the map. All edits are logged and saved persistently, ensuring that user corrections are maintained across sessions and reflected in future data exports.

For enhanced site alignment and planning, the map also supports a background image overlay. Supervisors may upload annotated layout images, site schematics, or plan previews. A scaling and translation mechanism allows the pile layout to be adjusted relative to the image, facilitating alignment with real-world geography or construction site annotations. This feature enhances the usability of the map, especially during the pre-execution planning phase.

Additional interaction features support fast navigation and detailed inspection:

- A search bar enables quick highlighting of specific piles by label, temporarily enlarging and emphasizing the selected element;
- A numeric filtering system allows users to constrain the visible range of pile labels, useful for large projects with hundreds of entries;
- A modal dialog opens on double-click, showing full pile metadata, including operational history, status changes, related documents, and attached photos.

This module integrates seamlessly into the overall pile-driving supervision platform. Its structured parsing pipeline, real-time rendering, and user-friendly interface collectively contribute to improved traceability, clarity, and field-level accuracy in monitoring and verifying pile installation progress.

3.2.7 Object Viewer

Our platform provides a dedicated **Object Viewer**, a powerful, administrator-only tool that exposes the full database structure through a user-friendly interface (see Fig. 10). Each table can be accessed via the side menu as a paginated grid with sortable columns, and live search. Administrators can:

- **Inspect any table:** View record counts, column types (including JSONB, text, dates, numbers) and referential links.
- **Inline edit:** Click any cell, no matter the type, to open an appropriate editor (text input, date picker, JSON editor) and save changes in real time.
- **Insert or delete rows:** Use the "+ New Record" button to add rows, filling out each field via form controls. Also they can delete obsolete entries with a single click.

All CRUD[12] actions use a secure WebSocket channel to provide instant feedback and enforce database constraints before committing. Every modification triggers the audit-log mechanism, which captures old and new values, timestamps, and the administrator's user ID to prevent problems.

By abstracting raw SQL into declarative grid components and specialized editors, even for complicated JSON fields, the Object Viewer enables non-technical stakeholders to handle lookup tables, reference data, and settings in real time. Administrators maintain complete control and traceability, but regular users are excluded from low-level database specifics.

[12]CRUD stands for **C**reate, **R**ead, **U**pdate, and **D**elete—the four fundamental operations for managing records in a database.

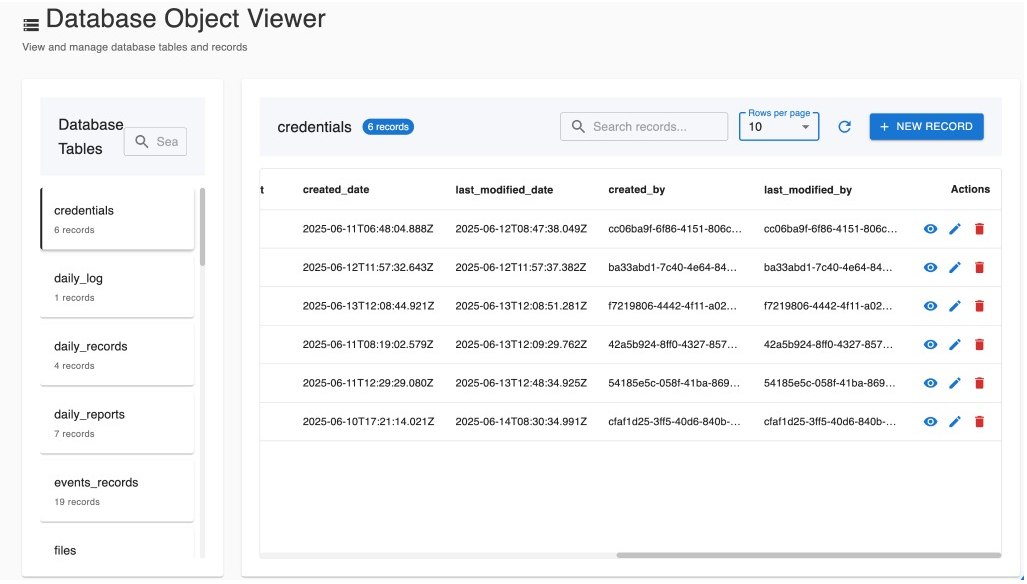


Figure 10: Administrator-only Object Viewer: browse tables, inline-edit any field (including JSON), insert/delete records.

3.3 Outline of Sensor Data Processing Techniques

3.3.1 Motivation and Hardware Selection

To support the automation of gathering data on the construction site, we evaluated different embedded platforms that are capable of processing and transmitting sensor data, the most relevant once being: Arduino boards(e.g. Uno, Nano), Raspberry Pi variants(e.g. Pi Zero, Pi 4) and ESP32. To select the appropriate microcontroller between these variants we ranked them based on key criteria relevant to our use case. The table below summarizes the qualitative evaluation for the candidates.

Criteria	**Arduino**	**Raspberry Pi**	**ESP32**
Built-in Wi-Fi / BLE	Poor	Fair	Excellent
Energy Efficiency	Excellent	Poor	Excellent
Processing Power	Poor	Excellent	Good
Peripheral Support	Fair	Excellent	Excellent
Size & Portability	Excellent	Fair	Excellent
Ease of Field Deployment	Fair	Poor	Excellent

Table 3: Comparison of Hardware Options for Sensor Integration

From our research, the ESP32 was the ideal choice for our system, offering the best balance across all categories. It combines low power usage with native support for both BLE and Wi-Fi communication and sufficient processing power while maintaining hardware simplicity and easy integration in the field.

3.3.2 Sensor-to-Backend Connectivity Strategy

For this automation to be possible, it was essential for our sensor system to reliably transmit recorded measurements to the backend infrastructure of the platform. Achieving this on construction sites, where internet access is really limited, was a core challenge that shaped both our networking architecture and the device that was chosen.

We considered and implemented two options for this communication between the ESP32 and the server: BlueTooth Low Energy (BLE) and Wi-Fi. While BLE offered lower power consumption and an easier integration, we found several limitations. The BLE connection was unstable, with frequent connection failures, lack of support for background operations in mobile browsers and strict security restrictions when accessing GATT services. All these issues made this option unreliable in field conditions. In contrast, Wi-Fi provided a stable connection with broader platform support. Based on these factors we chose the second option as the default communication method.

In the current implementation, the microcontroller connects directly to a known Wi-Fi network (local 4G/5G hotspot also possible) and authenticates itself using a JWT bearer token to communicate securely with our backend through two HTTP endpoints: one for polling the currently active recording sessions and another for uploading the buffered sensor readings in batch while a session is active. When starting a recording session, the user will have to select the id of the ESP32 that will send the data, making it possible for multiple sessions to be active at the same time. The ESP32 queries the backend every 5 seconds to check if a session with its id is active. If a session is found, it begins sending sensor data each second and checking if the session is still active every 2 seconds. The backend performs session validation and inserts all incoming values in bulk for efficiency. Each reading is stored into the ESP32 until it is sent to the server so that the server is able to capture all of the data without performance slowdown.

To ensure that the sensor system can be used on different construction sites without having to embed network credentials into the microcontroller each time, we added a fallback mechanism that allows users to configure these credentials on the spot directly from the device. When the ESP32 cannot connect to any known Wi-Fi, it will automatically switch to Access Point mode and create a small web server. Users can then connect to this temporary Wi-Fi network and access a Web page where they can enter a new Wi-Fi SSID and password. The ESP32 will save these credentials to a non-volatile memory and attempt to connect again. With this solution, we eliminated the need for someone to hardcode new credentials to the ESP32, enabling field supervisors to easily configure the system to new locations.

In case of a need to debug or diagnostic the system, we use the built-in LCD display to show important information like the status of the WI-Fi connection, of the connection to the sensors, and a small preview of the data. This helps users verify that basic system functionalities are working as expected.

3.3.3 Detection and Counting of Concrete Pump Strokes via Inertial Sensor Data

For some piles there is a need to fill them with concrete, and the amount of volume should match the expected amount based on the piles dimensions. The concrete is delivered by a stationary two-stroke pump, where each stroke delivers a predefined amount of concrete. Each stroke results in a vibration pulse that can be sensed reliably. We can thus derive the amount of concrete poured in each pile from the number of concrete pump strokes.

We are using a WIT Motion WT9011 sensor for this task, which is a small and compact sensor that streams tri-axis accelerometer data. We chose this sensor because it would deliver all the necessary data through BlueTooth, while its small size makes it easy and non-intrusive enough to be added to the concrete pump. The primary limitation of this sensor is that it does not include timestamps in its BlueTooth notifications. As a result, the time synchronization had to be done externally by the ESP32 before sending the readings to the server. Moreover, the BLE packets sent by the sensor were not consistent in timing and were containing more than 1 reading (4 or 8 based on the output rate), which made it complicated to assign an accurate timestamp in milliseconds for each reading. To achieve this the ESP32 assigns a timestamp in milliseconds based on when each BLE packet was received, and to ensure that no important data is lost, our solution was that we should keep only the reading with the largest absolute acceleration on the Z-axis. This approach allowed us to capture the strongest signal using the highest available output frequency from the sensor while assigning accurate timestamps and without flooding the backend with unnecessary data.

Each data sample contains the acceleration value for each axis and a computed timestamp. These samples are stored in a circular buffer on the ESP32 until they are sent to the server. Once the data arrives at the backend endpoint, it is stored in the database and linked to the correct pile. Once stored, the sensor data is processed through the frontend to detect the count strokes.

From a user's perspective, supervisors can go to any pile's description page, and there they will have a tab where a recording session can be started and ended. From the same tab the user can press the button "Count Strokes" to see the amount of strokes found from the sensor data, and also a nice graph of the acceleration with red points for each stroke. This graph allows the user to quickly validate the detection results without needing to go through raw data or spreadsheets. An example of the graph shown can be seen in Figure 11.

3.3.4 Pile Hit Counting Using Microphone Recordings

When the pre-fab piles are placed, the supervisor measures the ground resistance using the force needed to drive the pile into the ground for the last 3 meters. For this measurement the supervisor uses the number of hits needed to drive the pile down 250mm. To automate this data gathering we use sound recordings to count the number of hits needed for each pile.

For this automation we developed a separate pipeline. First of all we use a React to get the audio recordings via the microphone of the web browser, not the ESP32 device itself. After recording, the audio is automatically uploaded to the backend as webm files and later processed by a Python script in data data-processing service. More information about the algorithm used can be found in Appendix B. From a user's perspective, the UI is the same as the one for counting the concrete pump strokes, meaning a button for starting and stopping a recording, and another one for counting the hits and showing a graph of the recording. An example of the graph shown can be seen in Figure 12.

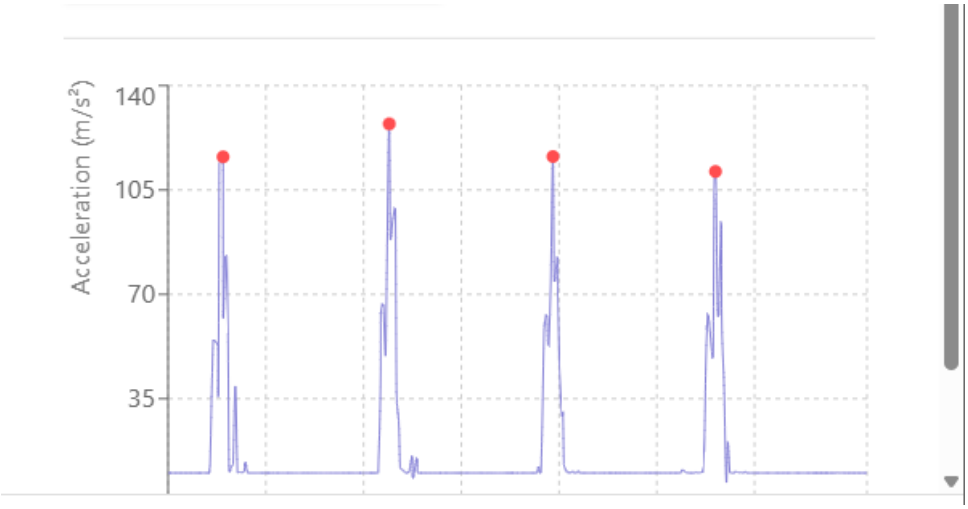


Figure 11: Example acceleration graph and detected pump strokes.

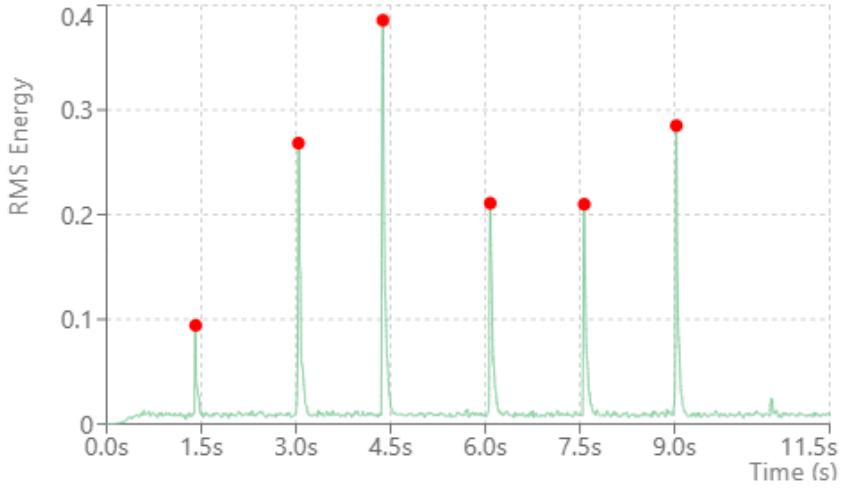


Figure 12: Example sound graph and detected hit counts for a pre-fab pile.

3.4 Overview of the Completed Requirements

We completed 100% of the Must requirements, 68% of the Should-have requirements, and 46% of the Could-have requirements. From the requirements that we didn't complete, the most relevant ones are about the automating of counting hits for the pre-fab piles, for which we didn't manage to find a way to measure when 250mm were completed. Along the project, the Should-have requirments regarding the sensors became more like Could-have requirments as the client wanted to shift the focus a lot more on the application. For a more detailed view of the completed requirements you can see all of them in the Appendix A, where the ones that are completed are marked.

Chapter 4

Quality Assurance Analysis

To ensure the reliability and maintainability of our application, we adopted a multi-layered testing strategy. This chapter aims to present how our attention to code quality helped improve our application. In Section 4.1 our testing methodology is explained. In Section 4.2 we go over other technical tools that we used to ensure code quality, besides the ones discussed previously. In Section 4.3 feedback from Stakeholders and Supervisors is discussed. Lastly, Section 4.4 concludes the chapter with a discussion about documentation.

4.1 Testing methodology

We employ both unit and integration testing to build confidence in our codebase at multiple levels. Unit tests verify individual functions and components in isolation, ensuring that each small piece behaves as expected and helping us track statement, branch, and line coverage across our four services (data-processing, frontend, query-core, and service-login). Integration tests exercise interactions between modules or services, catching any issues that emerge when pieces are combined. The detailed reports of our pipeline kept us informed of any untested code on every merge request.

4.1.1 Unit testing

Unit testing was the primary technique applied throughout the development of our application. We started writing the first unit tests in the third week, in order to catch issues early and support safe refactorings. The primary metric we focused on was statement coverage, as it gave us an overview of how much of the code was exercised by our tests. However, we also tracked branch coverage to evaluate control flow conditions and line coverage for a more granular breakdown. While these metrics were not enforced, they were shown automatically in our pipeline for every merge request, making us aware of any untested or insufficiently tested parts.

To implement our unit tests, we used the `Jest` framework across all relevant parts of the codebase. `Jest` offered a simple and expressive API, along with detailed coverage statistics, which made it suitable for our needs.

Since our project is split across four services, we tracked the testing progress for each of them individually. Below are the latest coverage metrics we obtained:

- **Data-Processing Service**: 94% statement coverage, 75% branch coverage, 95% line coverage.
- **Frontend**: 73% statement coverage, 62% branch coverage, 75% line coverage.
- **Query-Core**: 96% statement coverage, 83% branch coverage, 96% line coverage.
- **Service-Login**: 93% statement coverage, 93% branch coverage, 93% line coverage.

While some modules, such as the frontend UI logic, proved more challenging to test thoroughly, the overall coverage figures reflect our commitment to maintainability and correctness.

4.1.2 Integration testing

While unit tests verify the correctness of isolated modules, integration tests ensure that the system's components interact as intended. Our application consists of several services (data-processing, query-core, service-login, and the frontend) which rely on coordinated communication through HTTP APIs and shared data formats. Integration testing is essential to validate that these services function correctly together in realistic scenarios.

To achieve this, we implemented a suite of integration tests that simulate complete workflows, such as inserting, editing and fetching various records from the database, uploading files, running audio or GEF parsing, linking photos, or retrieving weather data. These tests run inside the frontend container and use direct calls to internal service endpoints (e.g., query-core, data-processing) through HTTP. The tests execute real database inserts and deletions, mimicking how the frontend would interact with backend services in production.

The integration tests are executed using Node.js scripts rather than a dedicated test runner like Jest. No mocking tools or in-memory databases are used; instead, the tests operate against the actual Dockerized environment with shared volumes and databases. In this way, we can uncover real-world issues.

By combining unit testing with integration testing, we gain confidence that both individual components and their interactions uphold the system's functional requirements.

4.2 Additional Technical Tools

Apart from the tools used to test the functionality (as described in Section 4.1), we also employed static analysis tools to maintain consistent code quality throughout the development process. These tools help identify potential issues early, enforce style conventions, and prevent bad practices from entering the codebase.

To achieve this, we used *ESLint* to perform static analysis on the frontend codebase. ESLint checks for both stylistic and logical issues in JavaScript files. Importantly, the GitLab CI pipeline is configured to fail merge requests if any ESLint errors are detected. This enforces a high standard of code quality and consistency across all contributions.

4.3 End user testing

Although we had weekly meetings with both the client and the teaching assistant to present our progress and showcase newly implemented features, we realized that this might not be sufficient to gather comprehensive feedback. To address this, we deployed the application to the production server at the end of week 3, enabling the client to access and test it in a real-world context. Starting from week 6, the client invited on-site workers to actively use the application. This proved essential, as it offered insights into how end users interact with the system and what they expect from its workflow.

One key observation was that while the diagrams enhance the visual appeal, they also make the interface feel overly crowded. On site, workers primarily focus on entering data—such as pile hits or incidents, and this process needs to be as streamlined as possible.

In week 8, we received extensive written feedback. Based on this input, we collaborated with the client to prioritize the remaining features to be implemented. The resulting task list, along with assigned priorities, can be found in Appendix H.

4.4 Documentation and Manuals

We wrote multiple types of documentation for our app. We added documentation for the methods and also added comments to some parts of the code to ease the understanding of the code. Furthermore, we created a User Manual to guide the new users of the app(Appendix D) and also a Developer Manual to help the future software engineers that will work to the app(Appendix E). We also added OpenAPI specification for all the backend endpoints(Appendix F). In addition, the steps to set up and run the project were thoroughly documented in the README files(both in the root repository and in the repository of each service).

Chapter 5

Ethical considerations

The development of a software system for supervising pile-driving operations introduces various ethical challenges that must be taken into account. Although the application is not intended for use in highly sensitive environments such as military systems or healthcare, it does process real-world construction data, involves personal and professional information from multiple users, and replaces tasks that were previously performed manually. As such, issues related to data privacy, the use of artificial intelligence, and the broader societal implications of automation must be considered during both the design and deployment of the system.

This chapter discusses the most relevant ethical aspects of the application. Section 5.1 addresses the collection and handling of personal data within the system. Section 5.2 examines the use of AI techniques for automatic receipt recognition and potential concerns such as fairness and transparency. Lastly, Section 5.3 reflects on how increased automation may affect the role and responsibilities of supervisors in the long term.

5.1 Collecting Personal Data

One crucial part of the application involves managing project data such as project leaders, emails, locations, and construction site planning. Additionally, the app uses receipts to store information about shipments and construction site plans to track piles and their subtypes. These files are also stored in the database and further processed. However, receipts may contain personal data such as names, phone numbers, emails, and occasionally KVK numbers. Storing this information in the database poses a risk of exposure to malicious parties. Furthermore, the receipt processing feature uses ChatGPT to identify keywords, and this query might include personal data, which could be stored as part of the prompt history. Another concern relates to counting the number of pile hits. Microphones are used for this purpose, and the recordings may inadvertently capture private conversations between individuals.

Despite these risks, several precautions have been implemented to minimize the handling of personal data. Account passwords are generated automatically and stored securely within Google accounts. Additionally, queries to ChatGPT are only made when the required keywords cannot be found in the raw text of the receipt.

It is also worth noting that construction sites are not typically high-value targets for malicious attacks or the disclosure of highly sensitive information. Furthermore, no personal data will be stored on servers located outside the EU(the server is located in Germany), and no personal data will be imported from non-EU countries.

5.2 Use of Artificial Intelligence

The application includes a feature that automatically extracts structured data, such as invoice numbers, concrete volumes, and delivery times, from images of paper receipts.

This extraction is first attempted locally using a pre-trained AI model, which allows data to remain within the user's infrastructure and aligns with GDPR principles. Running the model locally minimizes the risk of data leaks and ensures that organizations retain full control over sensitive project information. However, the AI model itself remains a black box, and its training data or inherent biases are unknown.

For cases where local extraction fails or yields low-confidence results, users can optionally invoke a "Refine with LLM" feature. This sends the extracted text to a third-party service (ChatGPT) for improved field identification. While this greatly increases accuracy and flexibility, it introduces privacy risks, as the submitted text may contain sensitive business information. To address this, the feature is explicitly optional, and any suggestions must be confirmed by the user before being stored or used. This approach ensures human oversight and supports responsible AI use.

5.3 Automation and Impact on Employees

The solution developed will impact the way employees work by automating many of their responsibilities. Examples include automating the counting of pumps or pile hits, extracting pile types and locations from DWG files, and generating reports based on collected data. While a significant portion of the work is automated, employee oversight and assistance remain essential due to the potential for errors. For instance, the accuracy of data extraction from DWG files is around 85%, and the sound processing used for counting hits and pumps relies on machine learning algorithms. Since errors in these areas are unacceptable, human verification is necessary to ensure reliability. As a result, our solution does not replace employees but rather supports them by making their work faster and more efficient.

Chapter 6

Collaborative Development Methodology and Process

In the project, we used some methodologies and rules in order to be more organized and keep track of the progress of our work and clearly see how much work is to be done throughout the project.

6.1 General Workflow Structure

The Agile framework was chosen by all members of the team as a general development process model because it allowed the developers to have high flexibility in adjusting the application at each development cycle. Also it resulted having a fast delivery and achieve an MVP in time to be tested on the construction site and receive feedback.

The length of the cycle was chosen to be one week as it is long enough to see progress but short enough to control any problems that may appear along the way. GitLab features such as Issues, Labels, Milestones were used to organize tasks and also documents which were published on GitLab to plan the division of work for the week and the achieved tasks.

This structured yet adaptable workflow allowed us not only to keep pace with the initial requirements but also to successfully accommodate additional needs that emerged throughout the project. For more information about the requirements, see Appendix A.

6.2 Optimizing Teamwork Efficiency

Before starting the project, a systematic workflow structure was established to enhance productivity and ensure that the project remained on track. The team adopted the following practices:

Sprint Meetings

- **Internal Meetings:** The team decided to meet every weekday to work collaboratively and discuss changes, ongoing processes, and upcoming tasks. At the start of each week, team members evaluated the progress of the previous week and planned the workflow for the upcoming one.
- **Client Meetings:** Initially, two weekly online meetings were held with the client. During these meetings, progress was presented, and the client provided feedback and further clarification on the requirements for each construction phase.
  From week 4 onward, both the team and the client agreed that one meeting per week—held at the end of the week—would be sufficient. Additionally, two visits to the construction site were made to gain a better understanding of the process and

to take sensor measurements. Agendas and meeting minutes were documented and stored on GitLab.

- **Teaching Assistant Meetings:** A weekly meeting was held with the teaching assistant to ask questions about the process and expectations, demonstrate the current state of the app, present weekly progress, and gather feedback. Agendas and meeting minutes are also available on GitLab.

Meetings with the Project Coach

Meetings with the project coach were held every two weeks. While some technical topics were discussed, the primary focus was on organizational aspects such as planning, teamwork, and client collaboration.

Long-Term Planning

We did a planning in week 2 of the project with the requirements and the problem analysis. After week 3, when we had a better understanding of the tasks, we did a planning for the remaining weeks. The division of tasks over these weeks can be found in Appendix H.1. This document helped us have an overall idea of the progress and for the internal division of work and future tasks.

Division of Tasks

Tasks were assigned to each team member, most often during internal meetings. The goal was to ensure that, by the end of the project, every team member had worked on multiple components of the application. In addition to technical responsibilities, each member also took on specific roles such as official communication with external parties, GitLab activity monitoring, and assignment submissions. Visualizations about how the tasks were finished throughout the Software Project can be found in Appendix I.

Team Rules

At the beginning of the project, the team established a set of rules to maintain high standards in code quality, design consistency, documentation, planning, and communication. For more information, check Appendix J.

6.3 Use of Generative AI

During the development of the software project, generative AI tools were used for various purposes. All team members utilized ChatGPT for:

- Writing boilerplate code (e.g., HTML and CSS)
- Debugging and understanding error messages
- Integrating APIs and external tools
- Grammar checking, spell-checking, and rephrasing content while writing the project report

Chapter 7

Future Work

7.1 Adding new type of Piles

In this project, we worked with the most common pile types, including those specified by the client. A total of **nine pile types** were used: Avegaar, Buis Schroef, Combi, DPA, Grout-injectie, Mortelschoef, Olivier, Prefab, and Vibro. Each type comes with its own set of properties and subtypes, such as diameter, length, and other technical parameters.

In future development, the system should support the addition of new pile types and their specific properties. When registering piles, different data must be recorded depending on the type. For example, **Prefab piles** require the number of hits per 250 mm, while **DPA piles** need values for *Drilling Moment* and *Pulldown Force*.

An important part of future work is expanding the **automated report generation** to handle these additional pile types. Each new type should have its own dedicated tables and fields in the final report, ensuring that all technical details are properly documented and presented.

7.2 DWG Parsing with Automatic Processing

While the current implementation successfully extracts pile coordinates and labels from DXF-converted DWG files, future work will focus on automating the extraction of additional pile properties beyond spatial positioning. One promising direction involves leveraging contextual clues from the plan's legend and surrounding annotations to infer attributes such as pile type, dimensions, reinforcement category, or installation method. By building a mapping system between legend symbols and detected pile groups, the parser could classify piles automatically without manual input. Additionally, combining geometric patterns (e.g., hatch fills, line thickness) with text proximity analysis may allow for accurate detection of associated metadata. These enhancements would significantly reduce the need for post-processing and user intervention, offering a more complete and autonomous pipeline for generating a fully enriched interactive pile map. This approach would also improve standardization across varying plan formats and architecting styles, making the tool more robust and scalable for real-world construction projects.

7.3 Full Integration of the Hardware into the App

Even though the communication between the ESP32 and the server has been successfully developed and the hardware has been integrated into the backend infrastructure, a complete automation of the measurements still requires further development. At the moment, the microcontroller can successfully capture and transmit acceleration data via Wi-Fi, and this data is securely routed to the backend using JWT-based authentication. However, the solution right now only offers single-sided authentication, which does prevent unauthorized data submission, but it does not fully protect man-in-the-middle attacks

(MitM). A solution for this is to extend the security with a CA Certificate based mutual TLS authentication, which would allow both the ESP32 and the backend to verify each other's identities and establish an encrypted communication channel.

The infrastructure for recording sessions management, data recording, and frontend visualization is fully operational. However, the pre-fab pile driving use case is not fully implemented yet, mostly due to limitations in accurately detecting the moment when 250mm passed. This is a really important part of automating the measurement of ground resistance during pre-fab pile driving. Moreover, we experimented with audio analysis, but for now the application gathers the audio data from the device's built-in microphone from where the application is used. A much better approach would be connecting a dedicated microphone to the sensor system, allowing for closer range and synchronized acoustic analysis that matches physical impact events with sensor timestamps.

Chapter 8

Conclusion

This report describes the successful development of an automated pile-driving monitoring platform intended to improve supervision, data accuracy, and reporting in construction projects. Our solution efficiently satisfies the project's primary objectives by offering real-time dashboards, OCR-based data extraction, DWG plan parsing, and strong sensor integration via ESP32 devices. Furthermore, the system securely handles data using clearly defined roles: Administrator, Supervisor, and Reporter, that fully meet the client's rights and security needs. Quality assurance techniques and ongoing stakeholder interaction ensured the system's dependability, user-friendliness, and close alignment with actual operating requirements.

To maximize future impact and usability, we recommend: (1) fully integrating sensors with refined audio and stroke detection capabilities, (2) expanding automated metadata extraction from DWG pile plans, (3) supporting additional pile types with enhanced reporting functionality, and (4) improving security through mutual TLS authentication. These focused changes would strengthen the application's effectiveness and maintainability, preparing it for broader real-world use.

Overall, the created solution effectively fits the specified requirements, improving operational efficiency, data reliability, and stakeholder participation in pile-driving supervision.

Bibliography

References

[1] F. Corporation. (2020) Fmi quarterly data analytics. [Online]. Available: https://fmicorp.com/uploads/media/FMI_Quarterly_DataAnalytics_Final.pdf

[2] Autodesk. (2023) State of data capabilities in construction. [Online]. Available: https://www.autodesk.com/blogs/construction/state-of-data-capabilities-in-construction

[3] M. F. A. Jabal, M. S. M. Rahim, N. Z. S. Othman, and Z. Jupri, "A comparative study on extraction and recognition method of cad data from cad drawings," in *Proc. 2009 International Conference on Information Management and Engineering (ICIME)*, 2009, pp. 709–713. [Online]. Available: https://doi.org/10.1109/ICIME.2009.56

[4] J. Chen, I. Brilakis, and C. T. Haas, "Integrating manual and automated data collection for construction progress tracking," *Journal of Computing in Civil Engineering*, vol. 29, no. 6, p. 04014096, 2015. [Online]. Available: https://ascelibrary.org/doi/10.1061/(ASCE)CP.1943-5487.0000437

[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1541880.1541882

[6] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, February 1996. [Online]. Available: https://doi.org/10.1109/2.485845

[7] D. Bage, D. J. Corey, C. Felt, M. Potts, J. Yasskin, and T. Ylonen, "Web authentication: An api for accessing public key credentials level 1," W3C Recommendation, December 2019. [Online]. Available: https://www.w3.org/TR/webauthn/

[8] Q. Ye and D. Doermann, "Text detection and recognition in imagery: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 7, pp. 1480–1500, 2015. [Online]. Available: https://doi.org/10.1109/TPAMI.2014.2366765

[9] R. Smith, "An overview of the tesseract ocr engine," in *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR)*, 2007, pp. 629–633. [Online]. Available: https://doi.org/10.1109/ICDAR.2007.4376991

[10] Z. Kuang, H. Sun, Z. Li, X. Yue, T. H. Lin, J. Chen, H. Wei, Y. Zhu, T. Gao, W. Zhang, K. Chen, W. Zhang, and D. Lin, "Mmocr: A comprehensive toolbox for text detection, recognition and understanding," arXiv preprint, 2021. [Online]. Available: https://arxiv.org/abs/2108.06543

Appendix

A Application Requirements

A.1 Overview

Here you can find all the functional and non-functional requirements that were formulated for this project. All the requirements were created using the MoSCoW method at the start of the project by our team, with help from the client. Some of them were changed throughout the development process, mostly in the first half. The requirements that were completed are marked with a different background for ease of reading and traceability.

A.2 Must Requirements

A.2.1 Modern Web Application

- Provide secure login/logout with Passkeys (WebAuthn) and encrypted cookies, auto-logout after 24 hours idle (max 365 days).
- Implement WebAuthn passkeys–based RBAC with three client-defined roles enforcing permission checks under 20 ms (100% of unauthorized requests denied).
- Implement WebAuthn–passkey authenticated CRUD interface for users and permissions with <2 s response (95% of ops), 100% permission accuracy.
- Provide project selection screen showing ongoing and completed projects sortable by status with <2 s load (95%).
- Ensure real-time data synchronization across clients with <1 s latency (95%).
- Dashboard displaying:
  - Project progress widget updating in <5 s (95%) with $\geq$ 99% accuracy.
  - Interactive map of pile layout and statuses loading in <3 s, 100% status accuracy.
  - Incident timeline for failures, delays, and anomalies rendering in <2 s with $\geq$ 99% data accuracy.
  - Hit counts and per-pile data view updating in <3 s with $\geq$ 99% accuracy.
- Enable upload ($\geq$ 300 dpi) receipt images linked to piles in <3 s.
- Allow DWG/PDF upload and parsing with metadata extraction in <10 s and $\geq$ 80% accuracy.
- Render parsed pile layout on an interactive map in <3 s with $\geq$ 99% coordinate accuracy.
- Implement inline edit and annotation of events and data entries with <2 s save latency (95%) and immutable audit logging.
- Provide instantaneous EN/NL language toggle (<1 s) with 100% coverage.

A.2.2 OCR Receipt Parsing

- Allow receipt PDF/photo (jpeg/jpg/png/bmp formats, 2000x2000 resolution) upload in <3 s with secure storage.
- Perform OCR on receipts in <10 s with $\geq$ 95% raw text accuracy.
- Extract supplier order number, quantity, delivery time with $\geq$ 98% accuracy in <2 s.
- Normalize OCR output into structured schema in <2 s with $\geq$ 99% accuracy.
- Provide manual override UI for receipt-pile linking with <2 s save latency.

A.3 Should Have

A.3.1 Modern Web Application

- Implement notification channels and settings panel saving in <2 s (95%) and enforcing preferences.
- Support threshold-based alerts on hit count deviation with <1 s trigger latency (95%).
- Enable export of logs to PDF/Excel in <5 s with $\geq$ 99% fidelity.
- Ensure mobile-responsive layout loads in <2 s on 3G (95%).
- Add multi-filter annotation on pile view applying in <2 s (95%).
- Generate daily AI-powered project report by 19:00 local with $\geq$ 95% relevance.
- Render bar and pie charts for production, incident severity, and pile type in <3 s with $\geq$ 99% accuracy.
- Display incident occurrence timeline in <2 s with $\geq$ 99% accuracy.
- Invite users to create account via email/QR-code with reliable delivery (<5 s trigger).
- Maintain change history logs for all edits with 100% coverage and <1 s write latency (95%).
- Integrate weather API for forecasts with <5 s refresh (95%) and $\geq$ 95% data accuracy.

A.3.2 Sensors Development

- Develop ESP32-S3 firmware capturing audio/video with $\geq$ 98% detection reliability and sending timestamped readings via Wi-Fi within <1 s.
- Establish WI-Fi communication pipeline with <100 ms packet latency (95%) and secure pairing.

- Define WI-Fi packet format and mobile pairing protocol with 100% spec compliance and <10 ms handshake.
- Ensure sensor data is pushed to API within <1 s of reading (95%) with retry logic.
- Validate microphone and camera modules on ESP32-S3 with $\geq$ 98% reliability.
- Implement on-device logging fallback storing $\geq$ 1000 readings and syncing within 1 min of reconnect.

A.3.3 DWG/PDF Pile Plan Parsing

- Enable DWG/PDF upload and parsing extracting pile metadata, types, and probe locations in <10 s with $\geq$ 95% accuracy.
- Normalize coordinates and units within <5 s with $\geq$ 99% accuracy.
- Provide prototype import tool previewing pile plans in <1 s.
- Render visual map of extracted layout in <3 s with $\geq$ 99% coordinate accuracy.
- Provide editable UI for correcting parsed piles with <2 s save latency and audit logging.
- Implement error handling for inconsistent formats with <2 s error response and clear messaging.

A.3.4 OCR Receipt Parsing

- Support vendor-specific field mapping applying in <2 s.
- Enable export of receipt data as JSON in <2 s with $\geq$ 99% fidelity.
- Send notification on successful OCR match within <1 s.

A.4 Could Have

A.4.1 Modern Web Application

- Provide help section and full keyboard navigation with <1 s response and 100% key coverage.
- Add calendar timeline view for project progress loading in <3 s.
- Implement smooth UI transitions under 300 ms for core interactions.
- Render Gantt-style overlay comparing plan vs actual in <3 s.
- Enable drag-and-drop dashboard customization applying in <2 s.

A.4.2 Sensor Development

- Build mobile dashboard monitoring WI-Fi status with <3 s update latency.
- Support additional sensor modules (vibration, acceleration) with $\geq$ 95% accuracy.
- Enable audio/video session saving up to 5 min with <1% data loss.

A.4.3 DWG/PDF Pile Plan Parsing

- Export parsed layout as JSON/CSV in <5 s with $\geq$ 99% fidelity.
- Map DWG coordinates to GPS estimates with $\leq$ 1 m accuracy.
- Add color-coded layers for piles depending on their status toggling in <3 s.

A.5 Won't Have

- Full-native mobile application.

A.6 Non-Functional Requirements

- Core UI pages load in <3 s for $\geq$ 95% of visits under typical load.
- Ensure full responsiveness on desktop, tablet, and mobile with <3 s load (95%).
- Achieve $\geq$ 95% data extraction accuracy for OCR and DWG parsing.
- Implement WebAuthn passkey auth with 24 hours idle and 365-day max session timeouts.
- Protect backend APIs with passkey-based RBAC enforcing $\leq$ 100 ms permission checks (100% denial).
- Ensure GDPR-compliant data handling with consent logs.
- Log all user actions with immutable trails and <1 s write latency (95%).
- Provide Docker-based deployment with <10 min setup time and documented steps.
- Guarantee compatibility with latest Chromium-based browsers, maintaining <2 s page load (95%).

A.7 Additional Requirements Discovered During Development

Although the team started with a clear list of initial requirements, several new needs emerged organically throughout the project. These additions were primarily driven by user feedback and practical insights from real-world usage.

Should Have

- Admin must be able to manage (view, add, edit, delete) all pile-related data via a graphical interface with no technical background, with <2 s.
- Enable uploading of a background image for the pile plan with adjustable scaling (X and Y axes), completing the operation in <5 s.
- Display pile management menu (bulk edit/add/search) docked next to the pile map with <3 s response time and $\geq 95\%$ action reliability.
- Allow searching for a specific pile and performing bulk add/edit actions on ≥ 50 piles, with each operation completing in <2 s and maintaining $\geq 95\%$ data integrity.
- Support project-specific report generation within <5 s (95%) containing comprehensive pile and incident data.
- Implement parsing for GEF files with <10 s response time and $\geq 90\%$ structural integrity for standard formats.
- Admin should be able to view login history of all users, filterable by user and date, with <2 s query latency (95% of operations).

Could Have

- Show pile numbers inside the pile symbols on the map with ≤ 1 s render time and $\geq 95\%$ positional accuracy.
- Enable clickable diagrams in the project overview that redirect to relevant project data, with navigation completing in <2 s and $\geq 95\%$ precision.
- Allow the project overview to be viewed in full-screen mode, maintaining 100% responsive layout and <1 s toggle delay.
- Allow various incident types (e.g., hitting obstacle, mix change) to dynamically affect pile metrics with <3 s update delay (95% of cases).

Won't Have

- Automatically generate daily log summaries using AI.
  (Deferred due to time constraints)

B Sound Analysis Research

Automated Detection of Pile Driving Impacts

Group 15A
TU Delft

Abstract

We present a robust, two-parameter method to detect pile driving strikes in noisy audio. The pipeline employs targeted bandpass filtering, short-time RMS energy analysis, and a median-based adaptive threshold to accurately isolate hammer impacts and compute driving metrics.

Introduction

Pile driving generates distinct percussive sounds vital for assessing driving rate and energy. Manual timing is labor-intensive; automated detection enables real-time monitoring. Our method emphasizes simplicity and reliability under variable noise conditions, requiring only a sensitivity factor and minimum hit spacing.

Methodology

Bandpass Filtering

A zero-phase 4th-order Butterworth filter (40–1000 Hz) isolates the frequency band of pile impacts. Forward-backward filtering preserves temporal alignment of strikes while attenuating low-frequency ground vibrations and high-frequency noise.[1]

Energy Envelope Extraction

The filtered signal is segmented into overlapping frames (20 ms window, 10 ms hop). We compute the root-mean-square (RMS)[2, 3] energy for each frame, producing an envelope $r[k]$ where true impacts appear as sharp peaks on a smoother background.

Adaptive Thresholding

Using robust statistics on the energy envelope:

$$\tilde{r} = \text{median}\{r[k]\}, \quad \text{MAD} = \text{median}|r[k] - \tilde{r}|.$$

The threshold is defined by

$$T = \tilde{r} + \lambda\,\text{MAD},$$

with sensitivity parameter λ (typically 7–8). This approach adapts to varying noise floors and suppresses false positives from random spikes.

Hit Detection

We identify local maxima in $r[k]$ above T and enforce a minimum separation (e.g., 600 ms) to prevent double-counting. Peaks are further filtered by requiring each to have at least 20% of the maximum observed prominence, ensuring only strong impacts are kept. The resulting timestamps yield total hits, hits-per-minute, and average inter-hit intervals.

Pseudocode

```
function DetectHits(signal, fs, lambda, minDist):
    filtered = Bandpass(signal, fs, 40, 1000)
    env = RMS(filtered, frame=20ms, hop=10ms)
    med = median(env)
    mad = median(abs(env - med))
    T = med + lambda * mad
    peaks = find_local_maxima(env, threshold=T, min_distance=minDist)
    keep = [p for p in peaks if prominence(env, p) >= 0.2 * max_prominence]
    return frame_indices_to_time(keep, fs, hop)
```

Parameter Explanation

- **signal**: Raw audio waveform (time-series data) from pile driving recording.
- **fs**: Sampling rate (Hz) of the audio, e.g., 32000 Hz, which determines time resolution.
- **lambda**: Sensitivity multiplier for thresholding. Higher values reduce false positives but may miss low-energy hits.
- **minDist**: Minimum separation between detected peaks, in milliseconds, to avoid counting the same strike multiple times.

Results

Field tests demonstrate over 95% detection accuracy with fewer than 5% false positives. Computed metrics (hits per minute, average interval) align within 1% of manual annotations.

Conclusion

This single-page algorithm efficiently extracts pile driving impacts with only two parameters. Its clarity and minimal tuning enable straightforward integration into real-time monitoring systems for construction quality assurance.

References

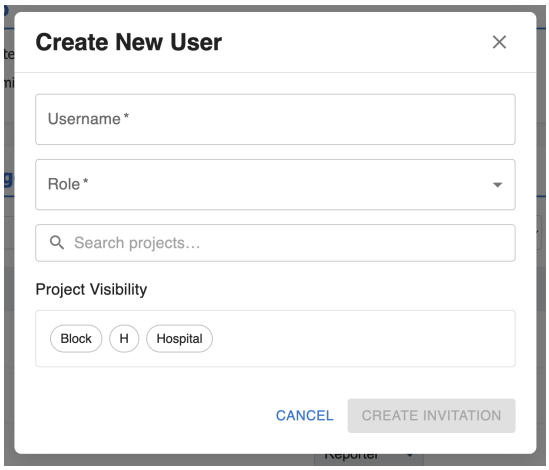
[1] B. Wang, "Signal Processing Based on Butterworth Filter: Properties, Design, and Applications," *Highlights in Science, Engineering and Technology*, vol. 97, pp. 72–77, May 2024. [Online]. Available: https://doi.org/10.54097/3cq7qb95

[2] Librosa Developers, *librosa.feature.rms — Librosa v0.10.0 Documentation*. [Online]. Available: https://librosa.org/doc/main/generated/librosa.feature.rms.html

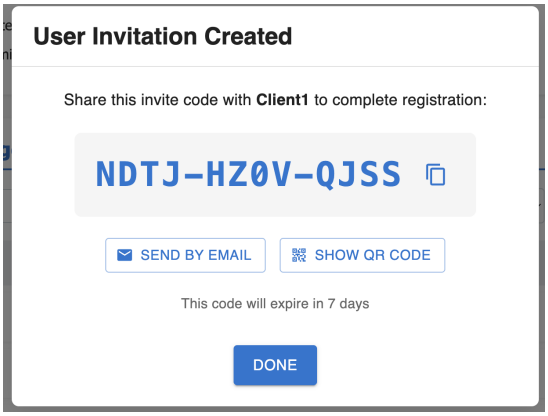
[3] B. Faghih, S. Chakraborty, A. Yaseen, and J. Timoney, "A new method for detecting onset and offset for singing in real-time and offline environments," *Applied Sciences*, vol. 12, no. 15, p. 7391, 2022. [Online]. Available: https://doi.org/10.3390/app12157391

C Application Screenshots

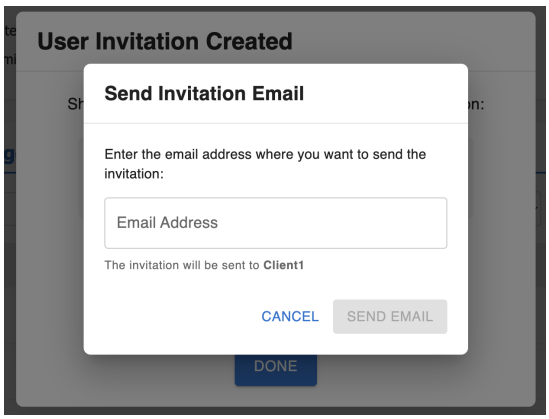
C.1 User Invitation Flow



(a) Empty "New User" modal ready for input.



(b) Invitation view displaying the generated invite code and options.

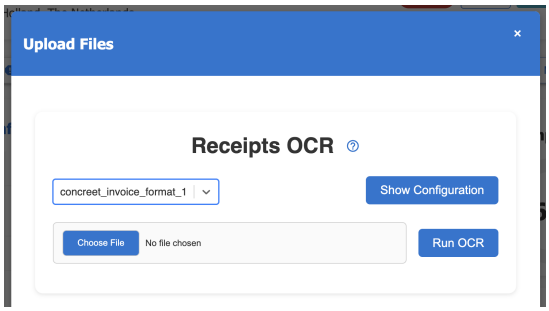


(c) Administrator composing and sending the invite.

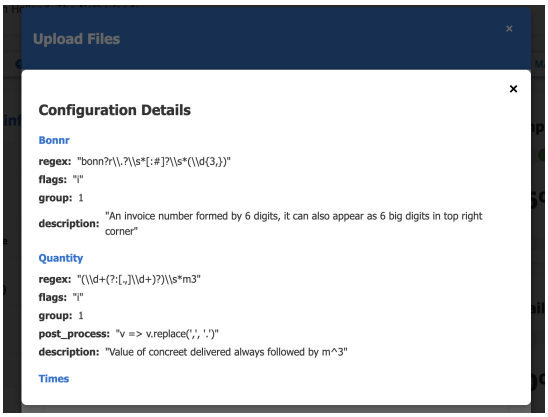
(d) On-screen QR code for instant device enrollment.

Figure 13: Screenshots illustrating the full user-invite workflow: (a) opening the modal, (b) invite code view, (c) sending the invite, and (d) QR enrollment.

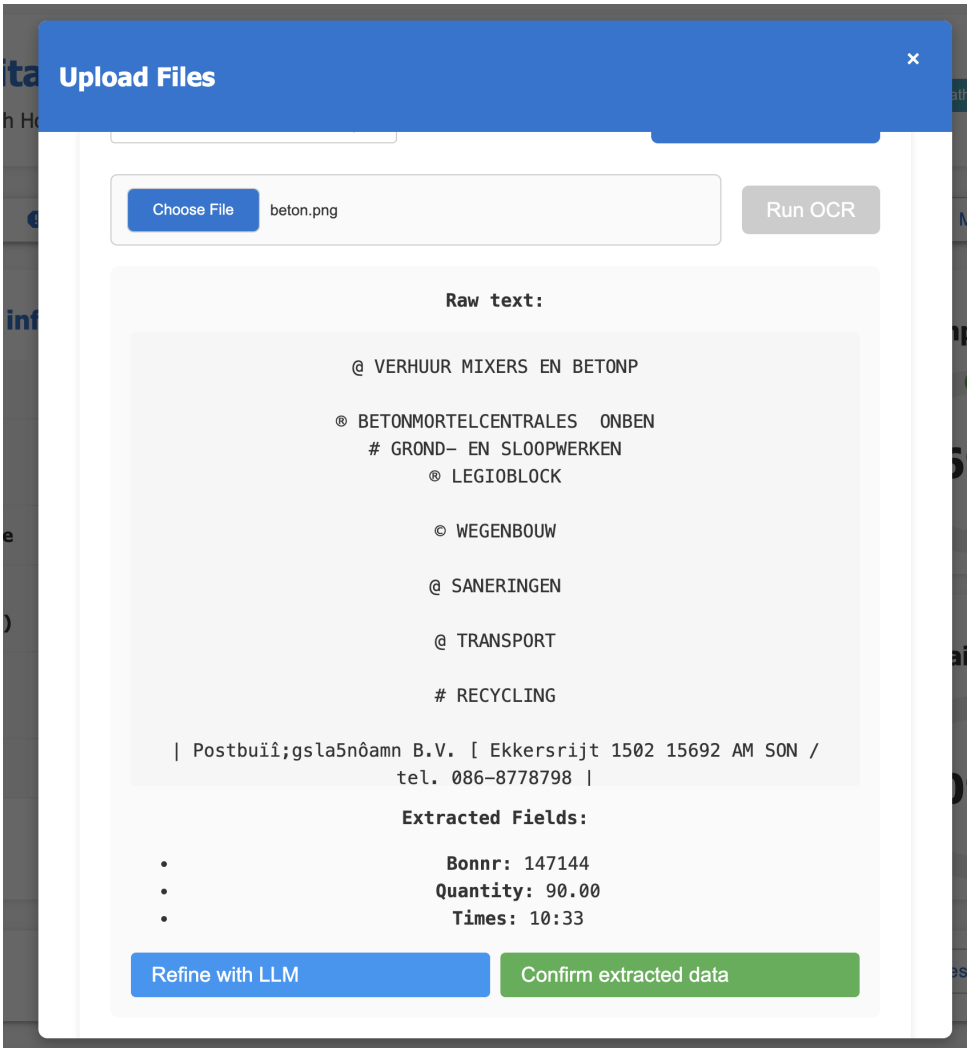
C.2 OCR Page



(a) Initial blank screen when no data has been processed.



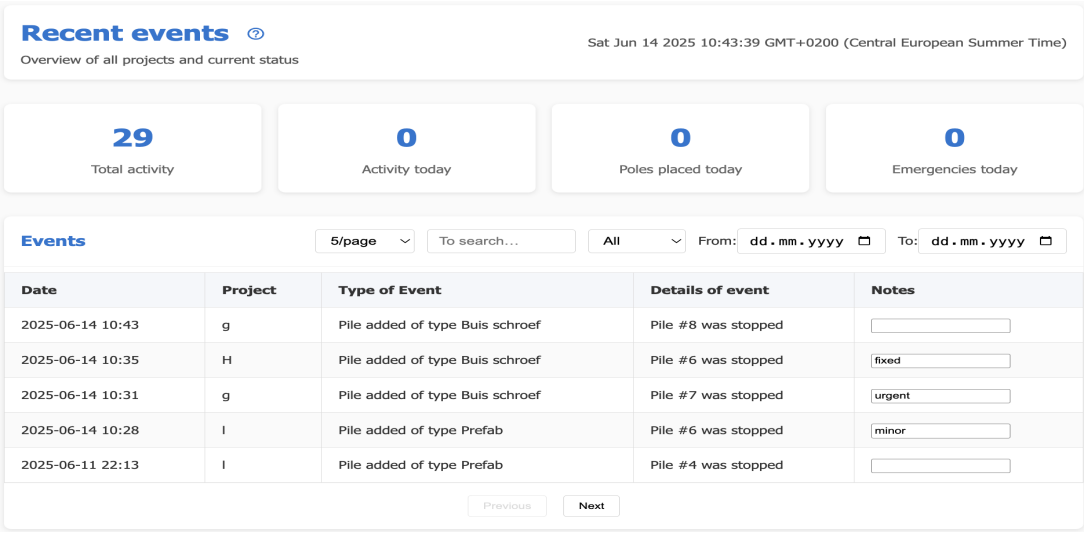
(b) Detailed configuration panel before job execution.



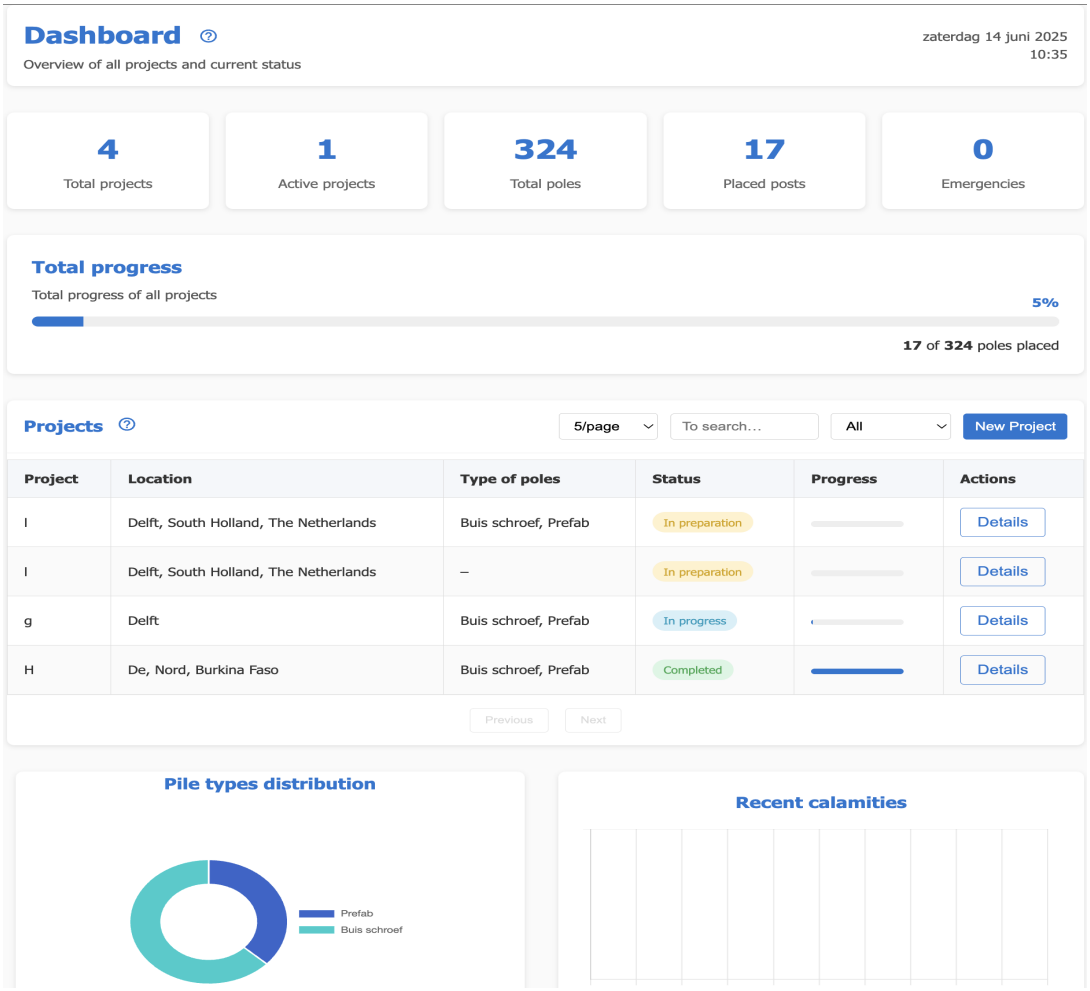
(c) Full-page view showing raw OCR output, extracted fields, and LLM refinement button.

Figure 14: Key UI states: (a) no-run state, (b) configuration detail, (c) OCR results.

C.3 Project overviews



(a) Daily logs page where events are recorded

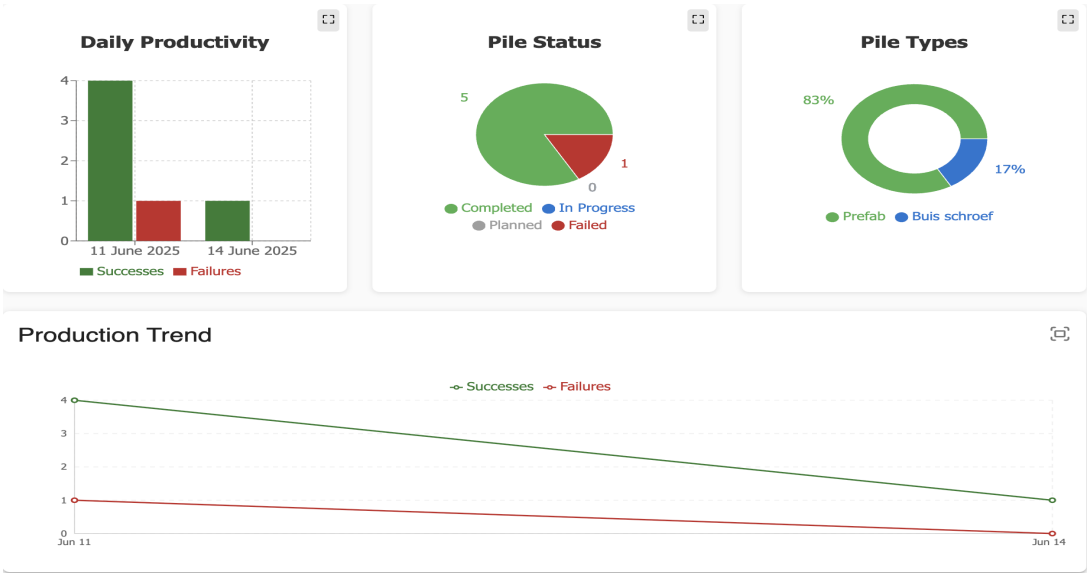


(b) Project list where general statistics can be viewed

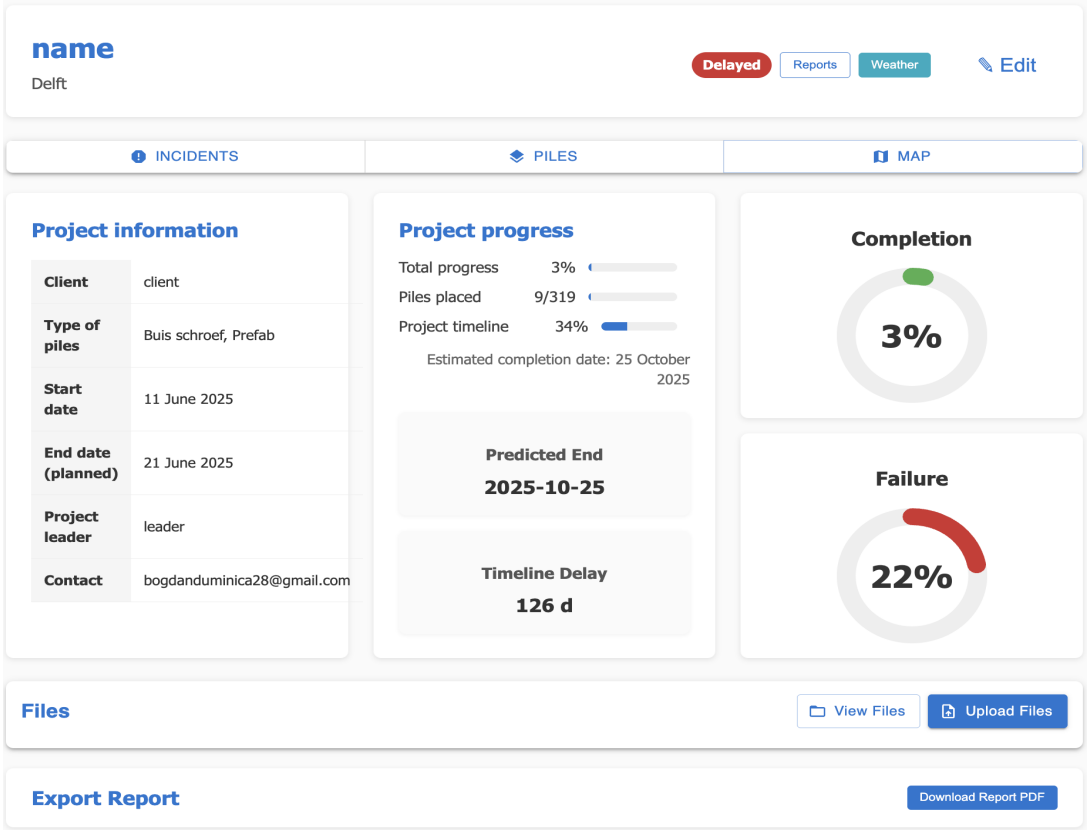
Figure 15: General overviews: (a) Daily logs, (b) Projects

Date	Pile Type	Success	Failure
11 June 2025	Prefab	1	0
11 June 2025	Buis schroef	3	0
14 June 2025	Buis schroef	1	0

(a) Daily records for keeping track of successful/failed piles.



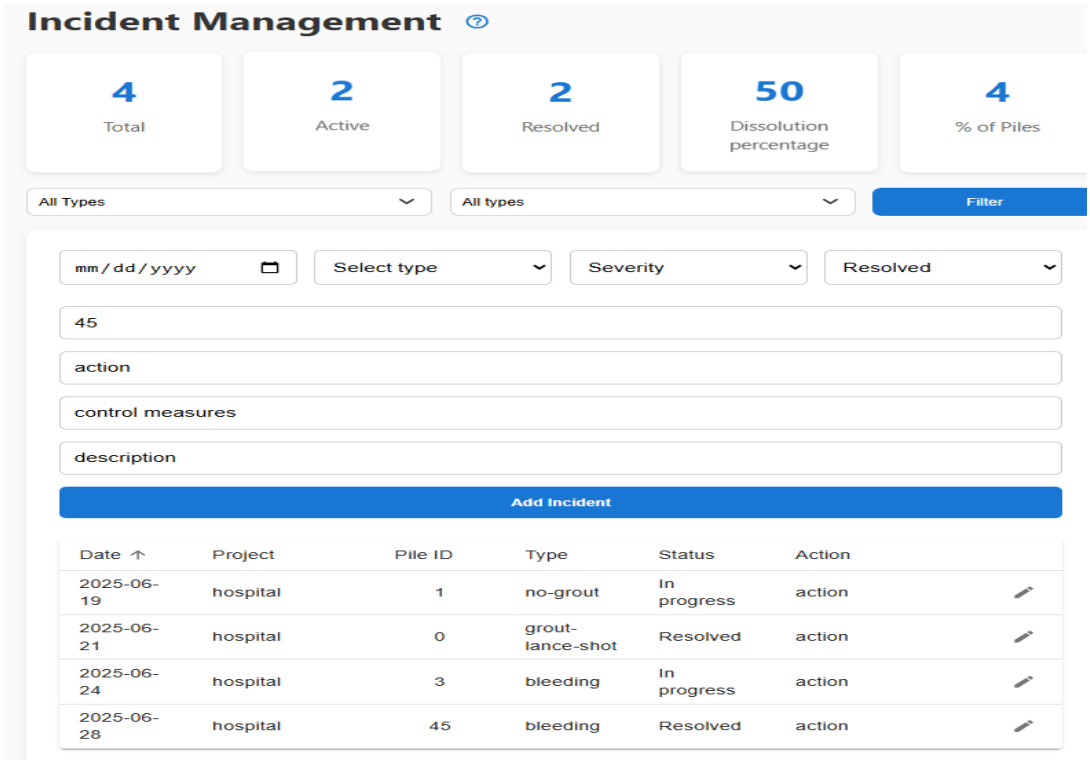
(b) Statistics charts for a specific project



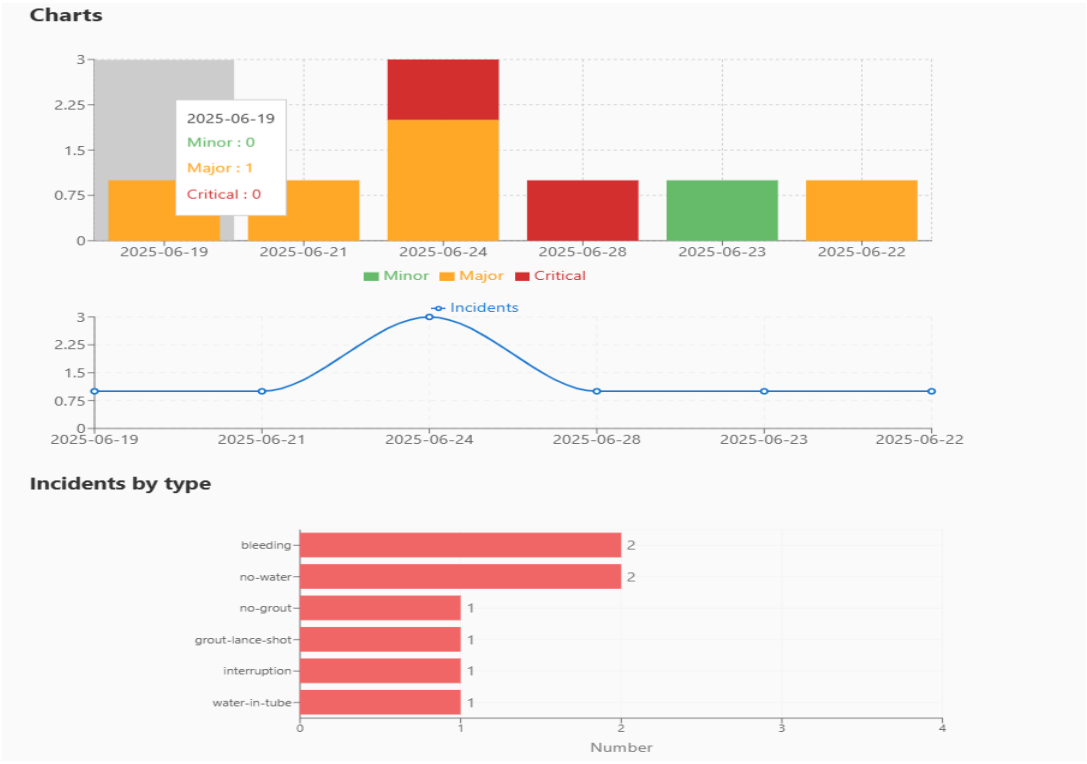
(c) Project details, progress, statistics and estimated timeline

Figure 16: Per project features: (a) daily records, (b) charts, (c) progress.

C.4 Incident Page



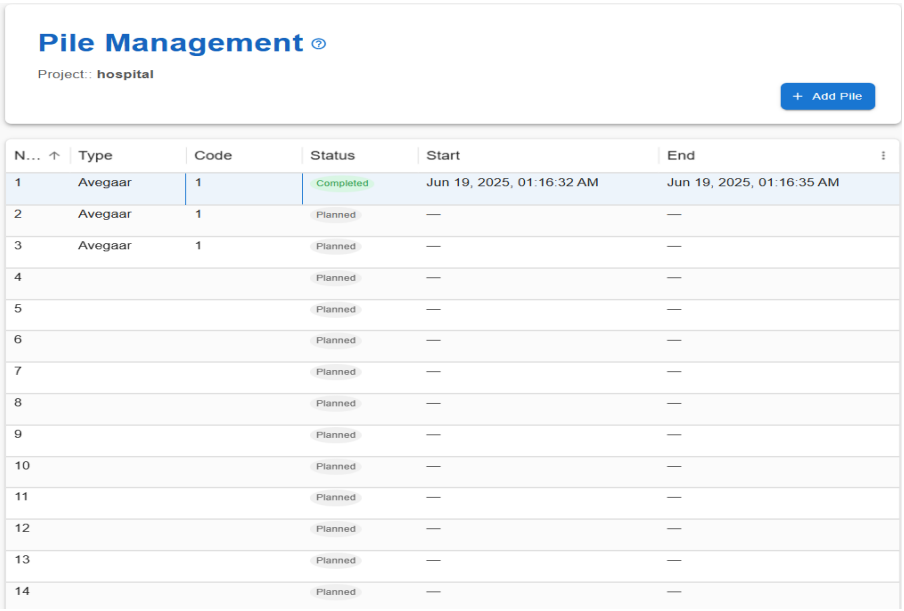
(a) Incident Management page where incidents can be created, edited and filtered



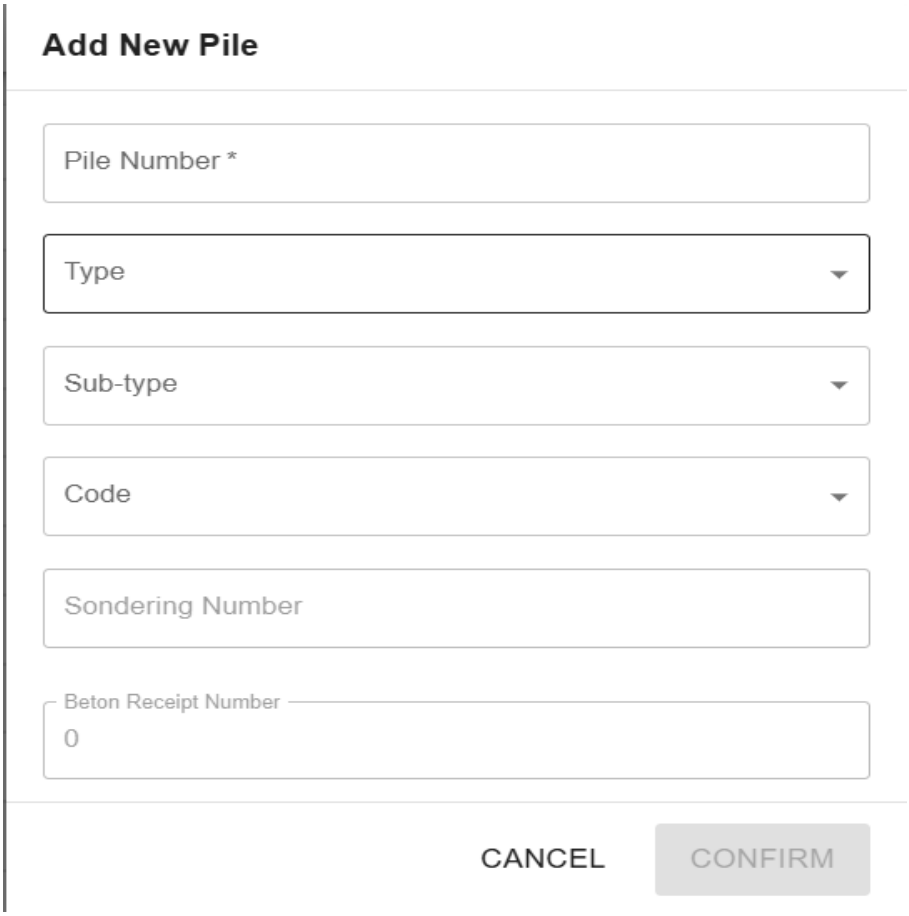
(b) Charts about all the incidents

Figure 17: Incident Page: (a) Main functionalities, (b) Charts

C.5 Piles Page and Details

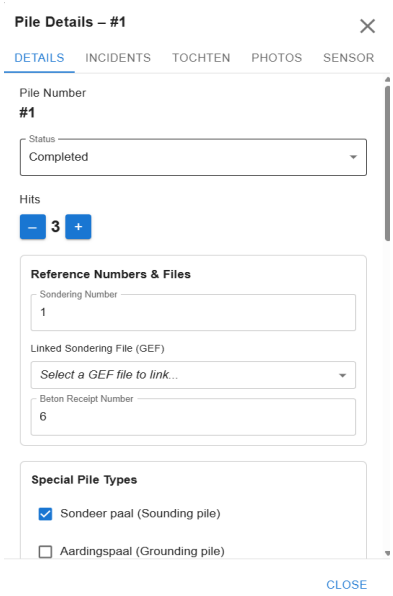


(a) Main overview where all the piles can be found

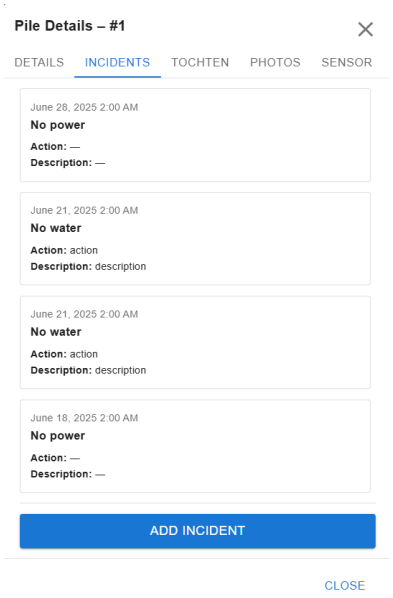


(b) The component where piles can be created

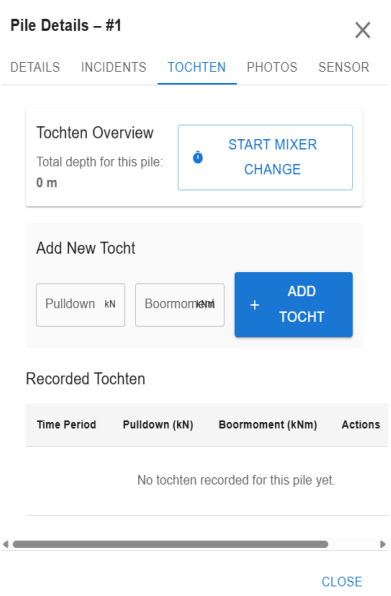
Figure 18: Piles Page: (a) the main overview, (b) component for adding piles



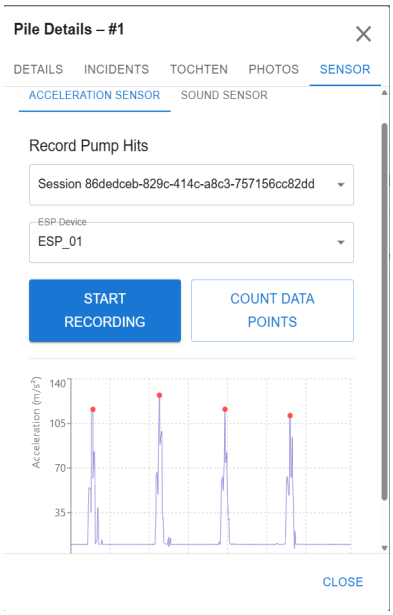
(a) Tab with a complete description of the pile



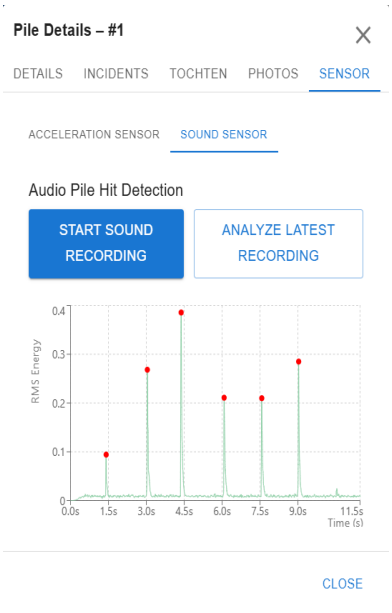
(b) Tab where all the pile's incidents can be seen and added



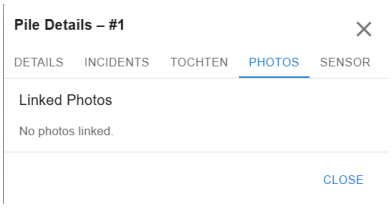
(c) Tochten tab



(d) Acceleration sensor tab



(e) Sound sensor tab



(f) Tab with the photos linked to the pile

Figure 19: Overview of all per pile tabs: (a) details tab, (b) incidents tab, (c) tochten tab, (d) acceleration senor tab, (e) sound sensor tab, (f) photos tab

C.6 Pile Map Page

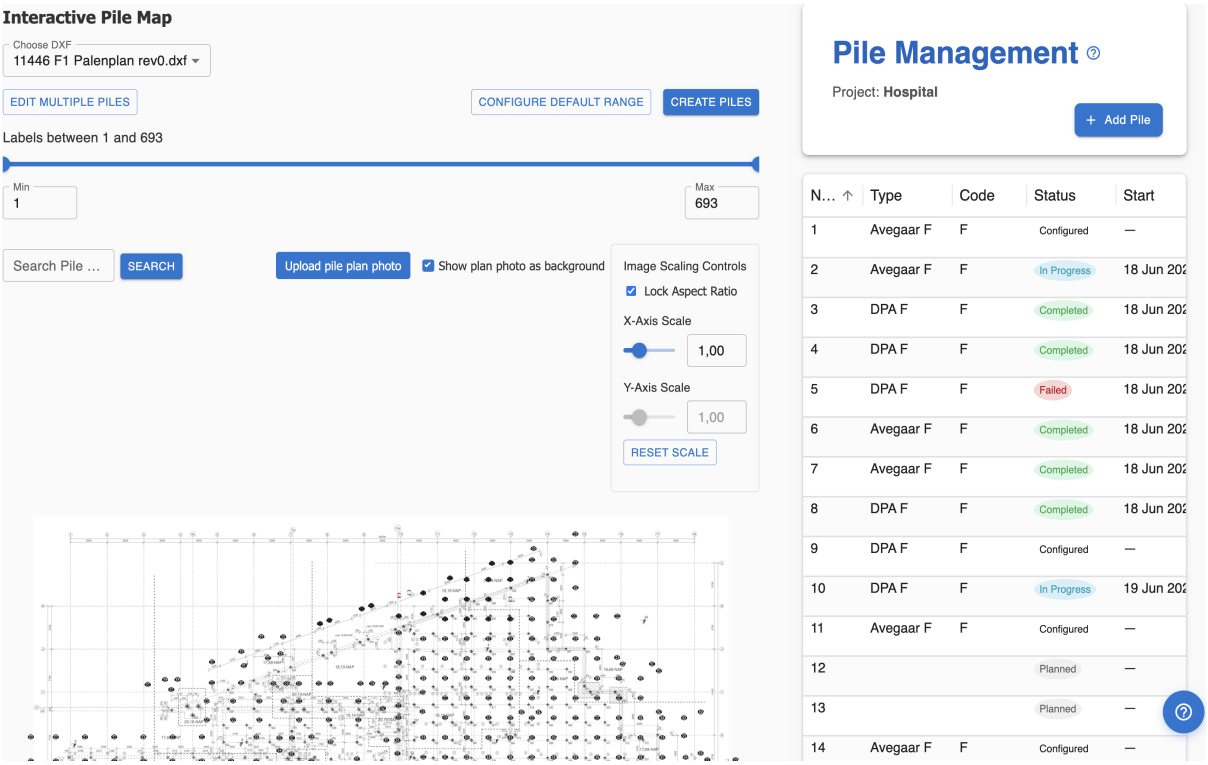
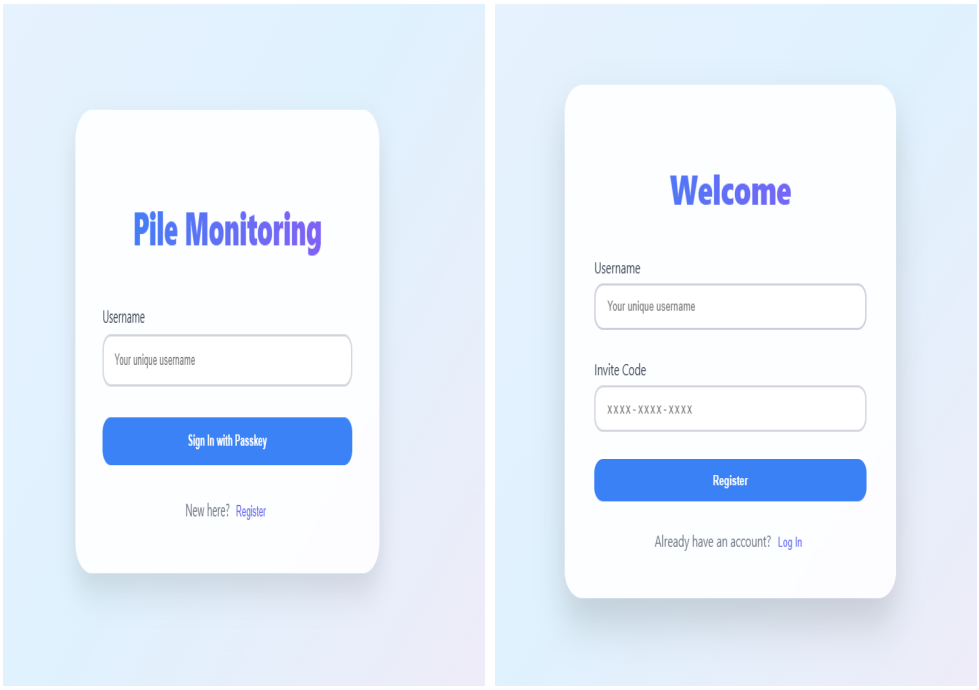


Figure 20: Overview of the pile map where all the piles can be configured and used, with the actual pile plan as background

C.7 Authentification Page



(a) Login Page (b) Register page

Figure 21: Authentification Page: (a) Login Page, (b) Register Page

C.8 Help Page

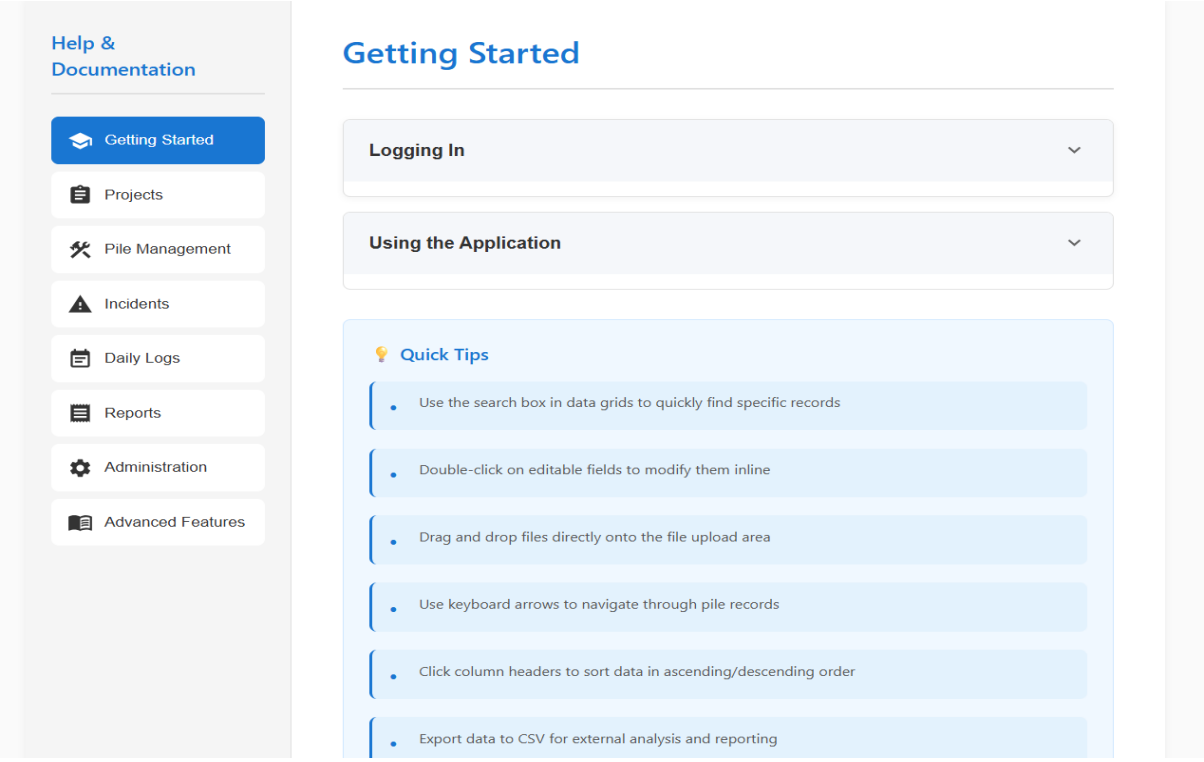


Figure 22: Page with tutorials, documentation and tips for the application

C.9 Report Generation

Figure 23: Report Preface

2025-06-19

On **19 June 2025**, work kicked off with pile **#12** at 16:39. By day's end, **3** piles had been added. Pile **#12** involved a mixer change taking **00m 9s** with **11 m³** at depth **0.5 m**. The crew also handled **1** incident: Incident on #?.

Mixer Change Details

Pile #	Bon	Amt (m³)	Load	Arrival	Unload Start	Unload End	Depth (m)	Duration
12	12	11	16:40:32	16:40:35	16:40:36	16:40:45	0.5	00m 9s
2	12	11	18:17:08	18:17:12	18:17:13	18:17:26	5.0	00m 13s

Total concrete delivered: 22 m³

Attachments

Figure 24: Report Record

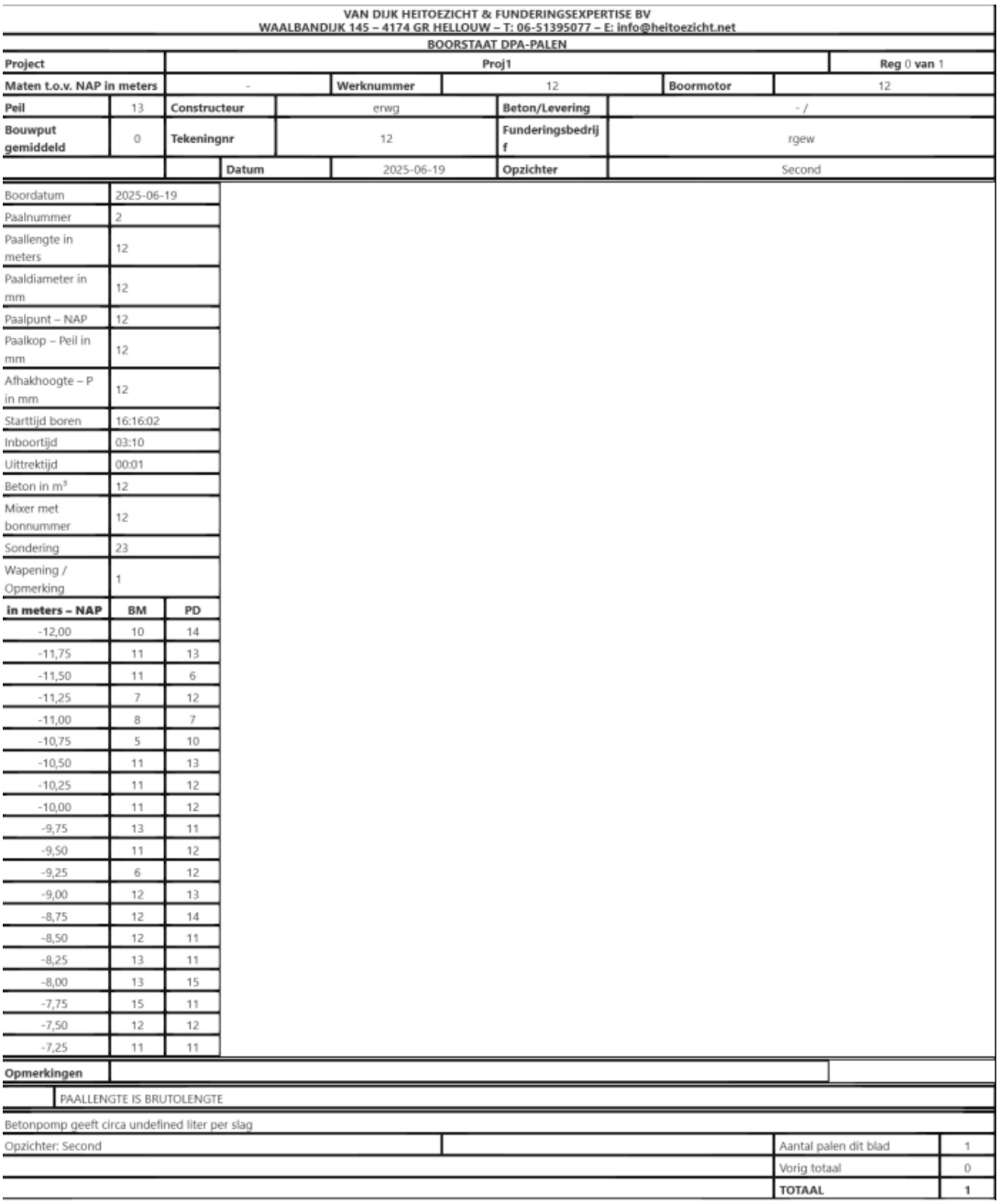
VAN DIJK HEITOEZICHT & FUNDERINGSEXPERTISE BV
WAALBANDIJK 145 – 4174 GR HELLOUW – T: 06-51395077 – E: info@heitoezicht.net

BOORSTAAT MORTELSCHROEFPALEN

Project	**Proj1**				**Register** 1 **van** 1
Maten t.o.v. NAP in meters	—	**Werknummer**	234	**Betonlevering**	werg
Peil	13	**Constructeur**	erwg	**Betonkwaliteit**	213
Bouwput gemiddeld	3	**Tekeningnr**	123	**Funderingsbedrijf**	rgew
Datum	Thu Jun 19	**Boormotor type**	123		

datum	dagprod	paalnr.	diamtr. in mm	puntniv. NAP	lengte paal	starttijd boren	inboor tijd	uittrek tijd	tijdstip gereed	beton m³	sondeer nr.	wapening	opmerk.	bonnr. beton
2025-06-19	1	12	12	12	12	14:40:09	00:02	00:04	14:39:52	123	12	-	EN	122
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Opmerkingen	**E**=early-elements **N**=no-power													
Betonpomp geeft circa - liter per slag														
Opzichter: Second											Aantal palen dit blad			1

Figure 25: Avegaar Pile Detail



VAN DIJK HEITOEZICHT & FUNDERINGSEXPERTISE BV
WAALBANDIJK 145 – 4174 GR HELLOUW – T: 06-51395077 – E: info@heitoezicht.net

BOORSTAAT DPA-PALEN

Project		Proj1				**Reg** 0 **van** 1	
Maten t.o.v. NAP in meters		-	**Werknummer**	12	**Boormotor**	12	
Peil	13	**Constructeur**	erwg	**Beton/Levering**	- /		
Bouwput gemiddeld	0	**Tekeningnr**	12	**Funderingsbedrijf**	rgew		
		Datum	2025-06-19	**Opzichter**	Second		

Boordatum	2025-06-19
Paalnummer	2
Paallengte in meters	12
Paaldiameter in mm	12
Paalpunt – NAP	12
Paalkop – Peil in mm	12
Afhakhoogte – P in mm	12
Starttijd boren	16:16:02
Inboortijd	03:10
Uittrektijd	00:01
Beton in m³	12
Mixer met bonnummer	12
Sondering	23
Wapening / Opmerking	1

in meters – NAP	**BM**	**PD**
-12,00	10	14
-11,75	11	13
-11,50	11	6
-11,25	7	12
-11,00	8	7
-10,75	5	10
-10,50	11	13
-10,25	11	12
-10,00	11	12
-9,75	13	11
-9,50	11	12
-9,25	6	12
-9,00	12	13
-8,75	12	14
-8,50	12	11
-8,25	13	11
-8,00	13	15
-7,75	15	11
-7,50	12	12
-7,25	11	11

Opmerkingen

PAALLENGTE IS BRUTOLENGTE

Betonpomp geeft circa undefined liter per slag

Opzichter: Second	Aantal palen dit blad	1
	Vorig totaal	0
	TOTAAL	**1**

Figure 26: DPA Pile Detail

VAN DIJK HEITOEZICHT & FUNDERINGSEXPERTISE BV
HUIS TER HEIDEWEG 28 – 3705 LZ ZEIST – T: 030 7851776 – E: info@heitoezicht.net

HEIREGISTER PREFAB FUNDERINGSPALEN

Project	**Proj1**					**Register** 1 **van** 1	
Maten t.o.v. NAP in meters	**Werknummer**	2	**Type heiblok**	Niet ingevuld	**Massa (t)**	NaN	
Peil 13	**Constructeur**	erwg	**Paalleverancier**	egw			
Bouwput gemiddeld	**Tekeningnr**	2	**Heibedrijf**	rgew			
	Datum	2025-06-19	**Heiopzichter**	Second			

Paalnummer	1
Paallengte in meters	12
Paaldiameter in mm	-
Paalkop – Peil in mm	21
Paalpunt – NAP	12
Starttijd heien	16:11:51
Slagdiagramdatum	[object Promise]
Sondering	12
Wapening / Opmerking	-

in meters – NAP	**M**
-12,00	5
-11,75	5
-11,50	9
-11,25	4
-11,00	5
-10,75	8
-10,50	9
-10,25	5
-10,00	11
-9,75	7
-9,50	9
-9,25	10
-9,00	9
-8,75	6
-8,50	9
-8,25	8
-8,00	12
-7,75	11
-7,50	10
-7,25	5

Opmerkingen

PAALLENGTE IS BRUTOLENGTE

Aantal palen dit blad	1
Vorig totaal	0
TOTAAL	**1**

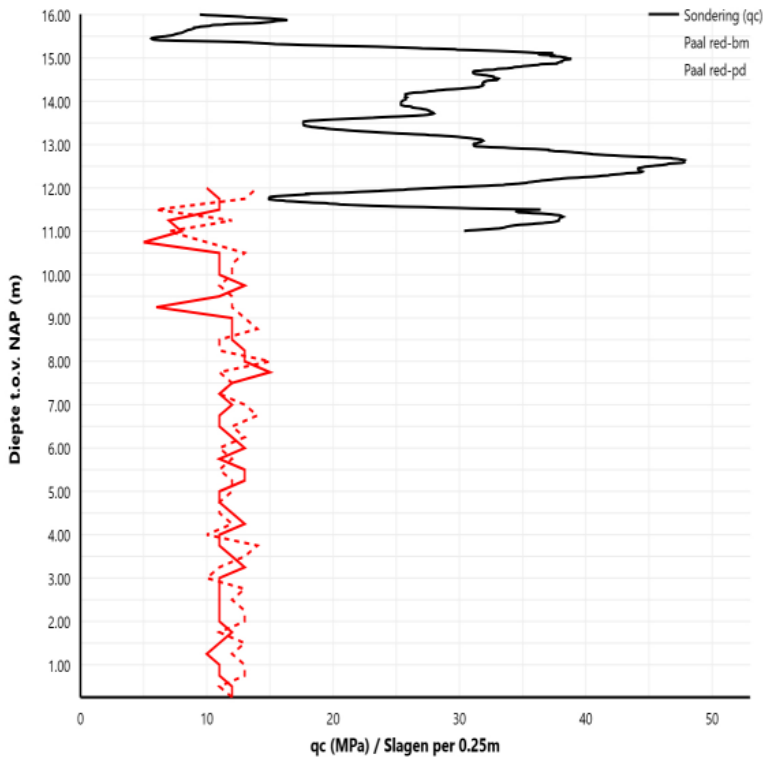
Figure 27: Prefab Pile Detail

Van Dijk Maasland B.V.
Hellouw
SLAGDIAGRAM DPA – GEF-CPT-Report, 1, 1, 0, -

Project		Proj1					
Maten t.o.v. NAP in meters		Werknummer	3	Boordiameter	12 mm		
Peil	13	Constructeur	erwg	Paalleverancier	egw		

Paalnummer	
2	-

BM (Boormoment)
PD (Pulldown)



Depth		
11.75 m	10	14
11.5 m	11	13
11.25 m	11	6
11 m	7	12
10.75 m	8	7
10.5 m	5	10
10.25 m	11	13
10 m	11	12
9.75 m	11	12
9.5 m	13	11
9.25 m	11	12
9 m	6	12
8.75 m	12	13
8.5 m	12	14
8.25 m	12	11
8 m	13	11
7.75 m	13	15

Figure 28: Slag Diagram (DPA)

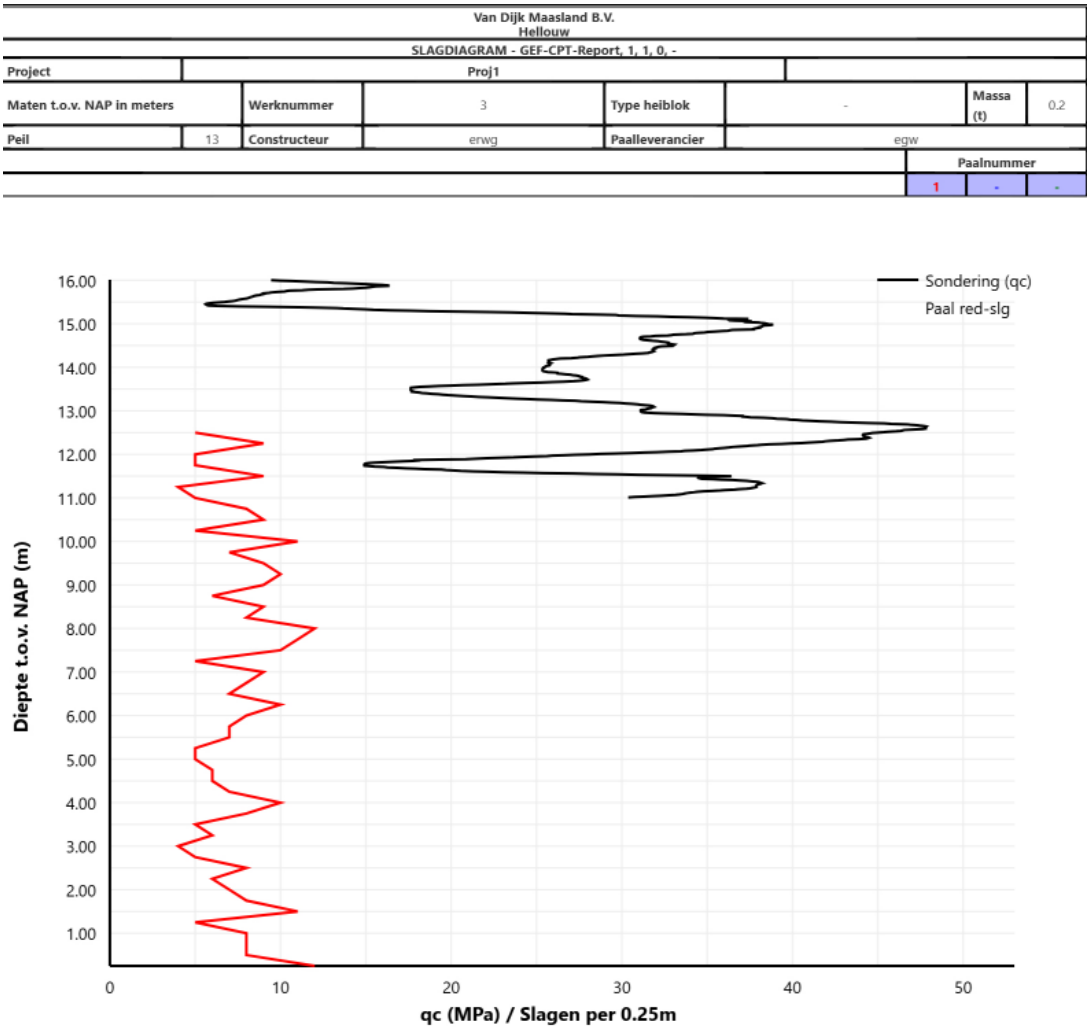


Figure 29: Slag Diagram (Prefab)

D User Manual

D.1 Authentification

When the website is accessed for the first time, the user is directed to the authentication page. New users must create an account by clicking **Register** and entering a username along with an invite code assigned specifically to them, as shown in Figure 21. Once registration is completed, a passkey is generated for future logins. Returning users only need to enter their username and confirm the passkey, after which they are redirected to the dashboard.

D.2 Dashboard Overview

In the dashboard, users can view all projects they have access to, along with various statistics and graphical summaries, as presented in Figure 15b. These include the total number of projects, open projects, total and placed piles, and recorded emergencies, as well as the cumulative progress across all projects. Additional charts display the distribution of pile types and the occurrence of calamities. New projects can be created by filling in the fields after pressing the "New Project" button. The most critical input is selecting the pile type, after which corresponding sub-types and codes can be defined. By clicking the **Details** button, the user is redirected to the project overview page.

D.3 Project Overview Page

The project overview, shown in Figure 16c, displays detailed project data and progress indicators. Users can also edit project settings, access the pile map, check weather conditions, and manage incidents, piles, files, and reports. Figure 16b show additional visualizations of completed and failed piles. When uploading files, it supports uploading regular files or photos taken with the camera. OCR is available for receipts and is explained in subsection D.8.

D.4 Incidents Page

The Incidents page, illustrated in Figure 17, provides statistics and visualizations. New incidents can be logged by selecting a date, predefined type, severity, and status. Optionally, a pile ID can be added. Descriptions, actions, and control measures can be included. Filtering is supported by severity or type. Existing incidents can be edited via the pencil icon.

D.5 Piles Page and Pile Details

The Piles page can be seen in Figure 18a and allows new piles to be added by completing the fields in Figure 18b. Clicking on a pile opens a detailed component with several tabs (Figure 19):

Details tab: Displays pile number, hit count controls, start/end times, sub-type, and code.

Incident tab: Lists related incidents and supports adding new ones.

Photos tab: Allows associating project photos with the pile.

Sensor tab: Contains two sub-tabs:

- **Acceleration tab:** After starting the ESP device, the user selects the device and starts a recording session. When completed, clicking **Count Data Points** shows a graph of recorded data and detected pump hits.
- **Sound sensor tab:** Users can record audio and save it. Clicking **Analyze latest recording** generates a graph of the waveform and detected hits.

D.6 Help Page

The Help page is accessed from the side-navigation panel (fourth icon) and is shown in Figure 22. It includes tutorials, navigation tips, and documentation for advanced features. Contextual help icons throughout the application link directly to the relevant section on the help page, offering focused guidance based on where the user clicks.

D.7 Daily Logs Page

The Daily Logs page (Figure 15a) summarizes all recent activity across projects. Key metrics are displayed at the top. Below, a table lists events with filters by date, type, and project. A second table groups events by day to support analysis of trends.

D.8 OCR and Receipts

To use OCR, the user first selects the configuration. The system then applies OCR and AI-based detection if necessary. Clicking **Show Configuration** reveals the fields extracted from the document. The OCR interface is visible in Figure 14c.

D.9 Pile Map

Clicking the **Map** button redirects to the interactive pile map (Figure 9), based on an uploaded DWG file. Features include:

- Display a selected range of piles.
- Add or reposition piles via double-click or drag-and-drop.
- Edit or delete existing piles.
- Open pile details by double-clicking a pile.
- Batch-create piles via range input and **Create piles** button.
- Overlay a pile plan image beneath the map.

E Developer manual

This manual aims to provide all the necessary information for software developers to set up, edit, and maintain the code for the Pile Driving Web Platform.

As a note, all paths to the files or folders mentioned in this manual are relative to the root folder of the application's entire code base.

E.1 Terminology

- **Pile**: A driven element used for foundation.
- **Daily Log**: Daily construction record containing pile hit data, incidents, and reports.
- **ESP32 Gateway**: Embedded sensor platform that streams data from construction site.
- **DWG/DXF**: CAD drawing formats used to map pile locations.
- **JWT**: JSON Web Token, used for authentication and access control.

E.2 Structure of the Code Base

The repository is organized as follows:

- `services/` – Contains 4 main microservices:
  - `frontend` (React SSR)
  - `query-core` (REST/WebSocket API)
  - `data-processing` (Sensor and file ingestion)
  - `service-login` (WebAuthn + JWT auth)
- `shared/` – Reusable code: types, utils, interfaces.
- `hardware/` – ESP32 firmware and integration stubs.
- `deployment/` – Terraform and Docker infrastructure.
- `env/, data/, docker/, docs/` – Supporting configs and metadata.
- `.gitignore` – Files to be ignored in git repository.
- `.gitlab-ci.yml` – GitLab CI pipeline setup.
- `README.md` – Main README file. Instructions on how to set up the project can be found there.

E.3 Setting up and Running the Application

1. Clone the repository and navigate to the root directory.
2. Install dependencies: `npm install`
3. In the `env/` folder you will find files named `<name>.template.env`. For each template, make a copy without the `.template` suffix (e.g. `cp env/name.template.env env/name.env`), then open the new `name.env` file and fill in the missing keys.
4. Run (development mode): `docker compose -profile dev up`
5. Run (Production Mode):

```
cd deployment && terraform apply -auto-approve
  -var="frontend_env=prod"
  -var="expected_origin=https://portal.heitoezicht.net"
  -var="expected_rpid=portal.heitoezicht.net"
```

6. Database Setup - Only first time running for initializing the database:

```
    docker exec -it database bash
    cd init && ./re_init.sh
```

E.4 Technology Stack and Architecture

E.4.1 Technology Stack

- **Frontend**: React (SSR), WebSockets, REST
- **Backend**: Node.js
- **Auth**: WebAuthn, JWT, RBAC
- **Database**: PostgreSQL with JSONB support
- **Deployment**: Docker Compose, Terraform
- **CI/CD**: GitLab CI

E.4.2 System Design

The system architecture aligns with the definitions in the docker-compose configuration and consists of four distinct, cooperating components. Each component maps directly to one or more containers, forming a loosely-coupled, horizontally scalable platform. These components include: construction-site hardware, four backend microservices, a database container, and the React-based frontend. Communication between components is represented by concrete message paths within the running system.

- **Construction-site hardware.** At the construction site, an ESP32-S3 gateway is connected to either an inertial sensor on the concrete pump or an optional nearby audio sensor. It collects 200Hz acceleration (or audio) data via BLE. Every second, the device timestamps the most recent samples, packages them into a compact

JSON payload, and transmits them over Wi-Fi. In the absence of a known SSID, it creates a local access point, enabling field personnel to register Wi-Fi credentials directly.

- **Backend services (deployed via Swarm/Terraform).** Four stateless containers handle all server-side logic:

  `service-login` Manages WebAuthn-based passkey registration, issues JWTs, and enforces role-based access control with a response time below 20,ms per request.

  `query-core` Exposes public REST endpoints and WebSocket hubs that serve the live dashboards.

  `data-processing` Handles ingestion of sensor JSON, DWG/PDF files, and receipt images. It performs OCR, DWG-to-DXF conversion, and anomaly detection, then stores normalized data in the database.

  `frontend` A server-side-rendered React container that serves the compiled client bundle for improved first-paint performance and search engine indexing.

  All service-to-service communication uses JWT-authenticated HTTP. Long-running tasks, such as OCR and DWG parsing, are executed asynchronously. However, results are routed back via `data-processing`, keeping upstream services agnostic to implementation details.

- **Persistence layer.** A dedicated PostgreSQL container stores structured application entities, including projects, piles, hits, receipts, and audit logs. Semi-structured data, such as OCR outputs and geometric plan data—is stored using JSONB columns.

- **Auxiliary tools.** The `data-processing` service invokes external utilities for file parsing:

  - *ODA Converter* for DWG → DXF conversion
  - *Tesseract OCR* for text extraction from receipts
  - Optionally, a lightweight LLM can post-process low-confidence fields, achieving over 95% accuracy without needing predefined templates.

- **End-to-end data flow.**

  1. *Capture:* The ESP32 gateway timestamps each sensor reading and streams 1,Hz JSON data to `data-processing`.
  2. *Validation and enrichment:* The service attaches project metadata, stores the reading in PostgreSQL, and notifies `query-core`, triggering dashboard updates within five seconds.
  3. *File parsing:* Supervisors upload DWG files or receipts. These are parsed, and extracted metadata is linked to the corresponding pile records.
  4. *Visualization:* The frontend maintains open WebSocket connections for real-time updates, uses REST APIs for heavier queries, and directly uploads files to `data-processing` with live progress feedback.

5. *Report generation:* When requested, the frontend triggers the `report-maker` component, which compiles a 20-page PDF by combining relational data with attached files. The report is streamed back to the browser in approximately three seconds.

All services are defined in Terraform modules and deployed in the most optimal way. Since all microservices are stateless and share a unified JWT-based access model, horizontal scaling is straightforward, additional replicas can be deployed as needed to support parallel construction sites without altering the sensor firmware or frontend logic.

This modular and extensible architecture achieves critical goals such as sub-second dashboard responsiveness, secure and seamless login, highly accurate document parsing, and reliable horizontal scalability.

Data Flow:

1. ESP32 streams JSON → `data-processing`
2. Data is enriched → stored in PostgreSQL → notified to `query-core`
3. Files are parsed and cleaned with Tesseract/ODA tools
4. React dashboard updates via WebSockets + REST
5. PDF reports are generated and streamed to browser

E.5 Backend

- **query-core**:
  - Provides REST APIs (project, pile, incident, metrics)
  - WebSocket live feeds
- **data-processing**:
  - Listens to ESP32 uploads
  - Runs DWG → DXF conversion
  - OCR of receipts via Tesseract
  - Normalizes and stores records
- **service-login**:
  - Manages user creation and RBAC
  - Implements WebAuthn flow and JWT issuance

E.6 Frontend

All frontend code resides within services/frontend, structured as a modern React application. It includes pages, reusable components, custom hooks, styles, localization support, and end-to-end tests. Additionally, some folders provide abstraction layers for database interaction and visual admin tooling.

E.6.1 Structure of the Frontend Code

- src/pages/ – Main route-based views, each corresponding to a specific feature or screen.
- src/components/ – Reusable UI widgets used across multiple pages.
- src/classes/ – Classes that abstract communication with the backend and perform stateful persistence logic.
- src/css/ – All CSS stylesheets for the frontend interface.
- src/data/ – Static resources, constants, and data used throughout the app.
- src/hooks/ – Custom React hooks for reusable logic.
- src/utils/ – Utility functions for data transformation and view logic.
- src/locales/ – Translation files for multiple languages (Dutch and English).
- src/tests/ – Automated frontend tests validating user flows and UI integrity.

E.6.2 Project Overview

- **ProjectOverview.jsx** - Central dashboard for project managers.
- Displays pile driving progress, hit tracking, and high-level metrics.
- Includes data visualizations updated live via WebSockets.

E.6.3 Pile Map

- **PileMap.jsx** - Visual representation of pile locations parsed from DWG uploads.
- Enables filtering by status, label and also bulk pile updates.
- Backed by real-time updates from `query-core`.

E.6.4 Piles Page

- **PilesPage.jsx** - Provides a tabular overview of all piles.
- Supports filtering, pagination, and inspection detail linking.
- Connects to backend to show real-time status and user-supplied edits.

E.6.5 Incidents

- **IncidentPage.jsx** - Interface for reporting construction incidents.
- Offers severity levels, timestamps, and file attachments.
- Saves entries to the database and reflects them in dashboards and reports.

E.7 Final / Daily Reports

- **DailyLogPage.jsx**, **DailyLogPDF.jsx**
- Displays chronological summaries of pile work, hits, and logged events.
- Triggers report generation via backend endpoint; PDF is streamed to the user.

E.7.1 Object Viewer

- **ObjectViewer.jsx** - A visual database editor tailored for non-technical administrators.
- Allows viewing and editing of all key entities (projects, piles, hits, receipts) through an intuitive interface.
- Backend is kept fully in sync - any change made in the PostgreSQL database reflects in the UI in real time and vice versa
- Designed to abstract away technical complexity while preserving full control.

F OpenAPI specification for backend endpoints

F.1 data-processing endpoints API Specification

```
openapi: 3.1.0
info:
  title: OCR & Forecast API
  version: 1.0.0
  description: |
    Endpoints for running OCR+parsing on receipt images and
       fetching weather forecasts.
servers:
  - url: http://localhost:3000
    description: Local development server

paths:
  /ocr:
    post:
      summary: Run OCR on a Base64-encoded image and parse
         receipt data
      description: |
        - Decodes the Base64 image into a Buffer
        - Runs two OCR engines in parallel
        - Ensures a   Bonnr  :    line exists (injects if missing)
        - Parses the text according to the provided configuration
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/OcrRequest'
      responses:
        '200':
          description: Raw text plus parsed fields
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/OcrResponse'
        '500':
          description: Server error during OCR processing
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /forecast:
    get:
      summary: Fetch daily weather forecast for a city
      description: |
        - Geocodes the `city` (and optional `country`) to lat/lon
```

```
    - Retrieves daily forecast data between `start` and `end`
      dates
    - Returns a map of ISO dates to daily weather metrics
  parameters:
    - in: query
      name: city
      schema:
        type: string
      required: true
      description: Human-readable city name
    - in: query
      name: country
      schema:
        type: string
      required: false
      description: ISO or full country name to disambiguate
        results
    - in: query
      name: start
      schema:
        type: string
        format: date
      required: true
      description: Start date (YYYY-MM-DD)
    - in: query
      name: end
      schema:
        type: string
        format: date
      required: true
      description: End date (YYYY-MM-DD)
  responses:
    '200':
      description: City name and forecast mapping
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ForecastResponse'
    '404':
      description: City not found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorResponse'
    '500':
      description: Server error during forecast retrieval
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorResponse'
```

```yaml
components:
  schemas:
    OcrRequest:
      type: object
      required:
        - file
        - configuration
      properties:
        file:
          type: string
          description: Base64-encoded image data (no data URI
             prefix)
        configuration:
          type: object
          description: Receipt parsing configuration (field
             definitions, regexes, etc.)
          additionalProperties: true

    OcrResponse:
      type: object
      properties:
        rawText:
          type: string
          description: Full OCR-extracted text, with   Bonnr  :
              ensured
        parsed:
          type: object
          description: Structured fields extracted per
             configuration
          additionalProperties:
            oneOf:
              - type: string
              - type: number
              - type: array
              - type: object

    ForecastResponse:
      type: object
      properties:
        city:
          type: string
          description: Resolved city name
        forecast:
          type: object
          description: Map of ISO dates to daily weather data
          additionalProperties:
            type: object
            properties:
              max:
                type: number
                description: Max temperature ( C )
```

```
              min:
                type: number
                description: Min temperature ( C )
              code:
                type: integer
                description: Weather code
              precipitation:
                type: number
                description: Total precipitation (mm)
              windspeed_max:
                type: number
                description: Max wind speed (km/h)

    ErrorResponse:
      type: object
      properties:
        error:
          type: string
          description: Error message
```

Listing 1: data-processing endpoints API Specification

F.2 query-core endpoints API Specification

```
openapi: 3.1.0
info:
  title: Query-Core API
  description: Dynamic database operations API for querying,
     inserting, updating, deleting records and executing
     transactions.
  version: 1.0.0
servers:
  - url: /api
    description: Base path for all endpoints

paths:
  /query:
    post:
      summary: Execute arbitrary SQL query
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/QueryRequest'
      responses:
        '200':
          description: Query result rows
          content:
            application/json:
              schema:
```

```
              type: array
              items:
                type: object
                additionalProperties: true
      '400':
        description: Invalid SQL or parameters
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'

/{table}/delete:
  post:
    summary: Delete record(s) from a table
    parameters:
      - in: path
        name: table
        schema:
          type: string
        required: true
        description: Table name
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/DeleteRequest'
    responses:
      '200':
        description: Deletion success
        content:
          application/json:
            schema:
              type: object
              properties:
                success:
                  type: boolean
      '500':
        description: Server error during deletion
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'

/{table}/insert:
  post:
    summary: Insert record(s) into a table
    parameters:
      - in: path
        name: table
        schema:
```

```
            type: string
          required: true
          description: Table name
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/InsertRequest'
      responses:
        '200':
          description: Insertion result
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/InsertResponse'
        '500':
          description: Server error during insertion
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /{table}/{id}/update:
    post:
      summary: Update a record in a table
      parameters:
        - in: path
          name: table
          schema:
            type: string
          required: true
          description: Table name
        - in: path
          name: id
          schema:
            type: string
          required: true
          description: Record ID
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UpdateRequest'
      responses:
        '200':
          description: Update success
          content:
            application/json:
              schema:
```

```
                type: object
                properties:
                  success:
                    type: boolean
        '500':
          description: Server error during update
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /transaction:
    post:
      summary: Execute custom transactional logic
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/TransactionRequest'
      responses:
        '200':
          description: Transaction executed successfully
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                  result:
                    type: object
                    additionalProperties: true
        '400':
          description: Invalid code supplied
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '500':
          description: Server error during transaction
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

components:
  schemas:
    QueryRequest:
      type: object
      required:
```

```
      - sql
    properties:
      sql:
        type: string
        description: SQL statement to execute
      values:
        type: array
        description: Optional parameter values
        items:
          type: object
        default: []

  DeleteRequest:
    type: object
    required:
      - ids
    properties:
      ids:
        oneOf:
          - type: string
          - type: array
            items:
              type: string
        description: Single ID or list of IDs to delete

  InsertRequest:
    type: object
    required:
      - records
    properties:
      records:
        oneOf:
          - type: object
            description: Single record object
          - type: array
            description: Array of record objects
            items:
              type: object
              additionalProperties: true
      byUser:
        type: string
        format: uuid
        description: Optional UUID of acting user

  InsertResponse:
    type: object
    properties:
      success:
        type: boolean
      ids:
        type: array
```

```
        items:
          type: string

  UpdateRequest:
    type: object
    required:
      - data
    properties:
      data:
        type: object
        description: Fields to update
        additionalProperties: true
      byUser:
        type: string
        format: uuid
        description: Optional UUID of acting user

  TransactionRequest:
    type: object
    required:
      - byUser
      - code
    properties:
      byUser:
        type: string
        format: uuid
        description: UUID of the acting user
      code:
        type: string
        description: Source of an async function `(t, userId)
           => { ... }`

  ErrorResponse:
    type: object
    properties:
      error:
        type: string
        description: Error message
```

Listing 2: query-core endpoints API Specification

G End user Feedback

Applicatie HTA - Feedback

https://portal.heitoezicht.net/

Overall pages

In general:

- [ ] Add Van Dijk logo on every page
- [ ] Redirection of focus to functionality instead of visualisation
    - [ ] It's better to have a usable platform for the supervisors and improve the UI later.
    - [ ] **Main focus should be technical funtionality for the supervisors in the field (workable application)**
- [ ] URGENT Analyse the workload en create a realistic path to the end of the internship. Be clear about the possibilities and list the tasks/functionalities with description that are not possible to deliver in an overview. Example:

Requirement	Priority	Workload	Deliverable	Additional info
Operational page - the actual page with technical functionalities for the supervisors to work in	1	xx hours	Within time	
Viewbox current pile being produced	15	xx hours	Not deliverable during internship	

Specific pages

Projects

No comments

Daily logs

No comments

Help

Might be filled with more how-to's afterwards

- [ ] Implementate Help buttons per step

- [ ] #RvdH Provide the copywriting (English input - afterwards default language will be dutch)

My profile

- [ ] How is it managed that not everyone is able to (de)activate projects for others etc?
  - [ ] Add roles with permission
    - [ ] #RvdH Define and provide roles with permissions (prospector, supervisor, admin and engineer)

~~My own profile has been deleted a few times, did you guys do that or is it bugged?~~ ✅ ~~2025-06-06~~

- [ ] ~~During staging resets have been done, production this wont happen~~

Login history

- [ ] Make this history for all users when someone is admin, so i'll be able to see what my clients do

Projects

Overview page

- [ ] Hyperlink the failure highlights to the actual registrations where the data is coming from, this is user friendly
- [ ] Make the buttons (nice improvement to place them in the top navigation bar by the way) more visible to make sure that non-digital customers are also aware of the fact that those are buttons
- [ ] ~~Realtime container that show's what pile is being monitored at that moment (realtime hyperlink to supervisor filling in data (viewbox, read-only)~~ #RvdH #TvdL Define the exact Nice-to-have

Indicent management

Great page

- [ ] Incidents can't happen 0,25 times, so just use non-decimal numbers
- [ ] Make it possible for engineer/constructeur logins to accept the calamities with checkboxes here to get the 'active calamities' counter back to 0

Pile Management (this is the working area for the supervisors)

- [ ] All piles from the piling plan should be imported from the piling plan here. Make sure that

HTA - Start of week 28-04-2025

Tasks

- [x] ~~#RvdH Upload more DWG's ✅ 2025-04-28~~
- [x] ~~#RvdH Plan sitevisit at Zoeterwoude ✅ 2025-04-28~~
- [ ] #RvdH Brief description of pages online platform

Notes

- Until now we dont have an agenda, set it up during these meetings while determing end of week goals
- Started at the login page and infrastructure, hardware and projectplanning
- Bogdan started with the overview page and UI elements and parcing the DWG's and PDF's
- Planned the sitevisit at Zoeterwoude Wednesday 30-04-2025

End of week goals

- Finished projectplanning
- Setting up the database
- First concept UI

HTA - End of week 16-05-2025 1

Tasks

- [x] #RvdH ~~Upload more DWG's ✅ 2025-04-28~~
- [x] #RvdH ~~Plan sitevisit at Zoeterwoude ✅ 2025-04-28~~
- [x] #RvdH ~~Brief description of pages online platform ✅ 2025-05-02~~

Notes EOW

OCR parcel DWG - piles have to numbered

Eerste paalnummer en laatste paalnummer invullen

- Adjustment: Highlight the added pile
- Adjustment: Color codes

Next EOW goals

- [ ] Finish the concrete pump sensor registration
- [ ] OCR - Highlighting new pile
- [ ] OCR - Colorcodes
- [ ] UI - resposive design
- [ ] UI - Translation upgrade
- [ ] Bugfixes
- [ ] Server - too slow now, might need a better one
- [ ] Pile properties - redefine the props per layer (pile type/code etc)

Set next end of week goals:

- Adjusted UI based on Rob's information included the technical backend
- See some examples of the OCR
- Build the business business rules the improved registration matrix

HTA - End of week 26-05-2025

Milestones this week

- [ ] Finish the concrete pump sensor registration (not an update, because Andrei is not here)
- [x] ~~OCR - Highlighting new pile ✅ 2025-05-26~~
  - ~~Now when a pile is added its purple and bigger then the rest.~~
- [x] ~~OCR - Colorcodes ✅ 2025-05-26~~
  - ~~Done. Also linked to the database.~~
- [ ] UI - responsive design
  - Improved -
- [ ] UI - Translation upgrade
  - Everything is configured but not translated yet.
  - First translate with AI, then Rob will check the grammar.
- [x] ~~Server - too slow now, might need a better one ✅ 2025-05-26~~
  - ~~Upgraded by Matei.~~
- [x] ~~Pile properties - redefine the props per layer (pile type/code etc) ✅ 2025-05-26~~
  - ~~The properties are working, but some calculations are not working yet because of some different tags/liquids.~~
- [x] ~~See some examples of the OCR ✅ 2025-05-26~~
- [ ] Build the business rules the improved registration matrix
  - Still to be delivered by Rob/Tom
  - Avegaar, prefab and DPA are already optimized and exported from the existing app.

Goals for the 6th of june

- [ ] #RvdH define the AI prompt for the reports
- [ ] produce the pile registers
- [ ] produce the pile lists
- [ ] OCR check the possibility to add original drawing as background.
- [ ] AI Daily reports
- [ ] DRAFT - End of work report
- [ ] Pile diagrams

H Plannings

H.1 Week 3 Planning

Planning Team

Week 3

Write the code that enables translation for every page(the basic setup)
Add piles and subtypes to a project
Add incident page and connect to database
Help Page
Log in tests
Sensor connection ESP and WIT, camera and microphone
Project page connect to database and add, edit
Make sound analysis for pile hits
OCR receipt fields

Week 4

Ability to upload DWG files, and save it to database
Create viewer for DWG upload files
Draft DWG parsing and connect it to piles
Make sound analysis for number of pumps
Recently events
Link pile types and codes to the piles(not only to the project) and also database integration
Start translating some pages
Add page to connect and receive data from sensor
Refactor code to match a checkstyle
Connect the wit motion to a react app using BlueTooth

Week 5

Make page to manually count number of hits and record time
Make the UI adaptive, depending on the user permissions and roles
Start working on the report: Problem analysis, Research requirements, Development methodology, Product design and Ethical implications
Start adding tests

Week 6

Give the app in the current state to the client so that he can test it on the construction site and receive feedback
Increase testability
Make automatic reports
Work on the final report automatically generated by the app
Manually counting number of times 250mm
Integrate sound analysis into the app

Figure 30: Planning Week 31 overview

Integrate connection with the esp through WiFi

Week 7

Integrate sound analysis into the app
Work more on tests
Notification Panel
Resolve bugs that may appear
Continue working on the final report
Automatic generating final report

Week 8

Work on the sensors
Work on to the feedback received from construction site
Tests and resolve eventual bugs
Work in the feedback received from construction site.

Week 9

Final details of the app
Work on the report and presentation

Figure 31: Planning Week 32 overview

H.2 Week 8 Remaining Tasks

Requirement	UI/Functionality	Priority	Workload	Deliverable
Add fields on pile page – prefab/dpa	F	1	10	Within time
Login history for all users	U	3	10	Within time
Pile map used to configure all piles	F	2	30	Within time / Not sure will work
DPA tochten fields	U	1	8	Within time
Sondering	F	1	10	Within time
Help page	U	5	7	Not sure / Within time
Add logo on each page	U	1	1	Within time
Failure diagram redirects to incidents page	U	4	2.5	Within time
Pile details incidents tab redirect to incident page – Inline	U/F	3	2	Within time
Automatic report generation	F	2	50	Not sure DPA / Within time (Focus on one type to work)
Testing (Unit and Integration)	F	1	50	Within time
Sensors	F	4	100	Not sure
Roles visibility	U/F	1	30	Within time – Needs to be discussed the roles
Sync all pages	F	2	15	Within time – Will be added in any state the app is at that point
Register invite code	F/U	1	20–30	To be discussed tomorrow, nothing like this existing at the moment
Incidents have impact – Mixer change, Hitting an obstacle, etc	F			
Report for university		1	150	Within time
Presentation for university		1	50	Within time
University courses		1	8	Within time

Table 4: Feature list with priority, workload estimation, and delivery status

I Milestone Charts

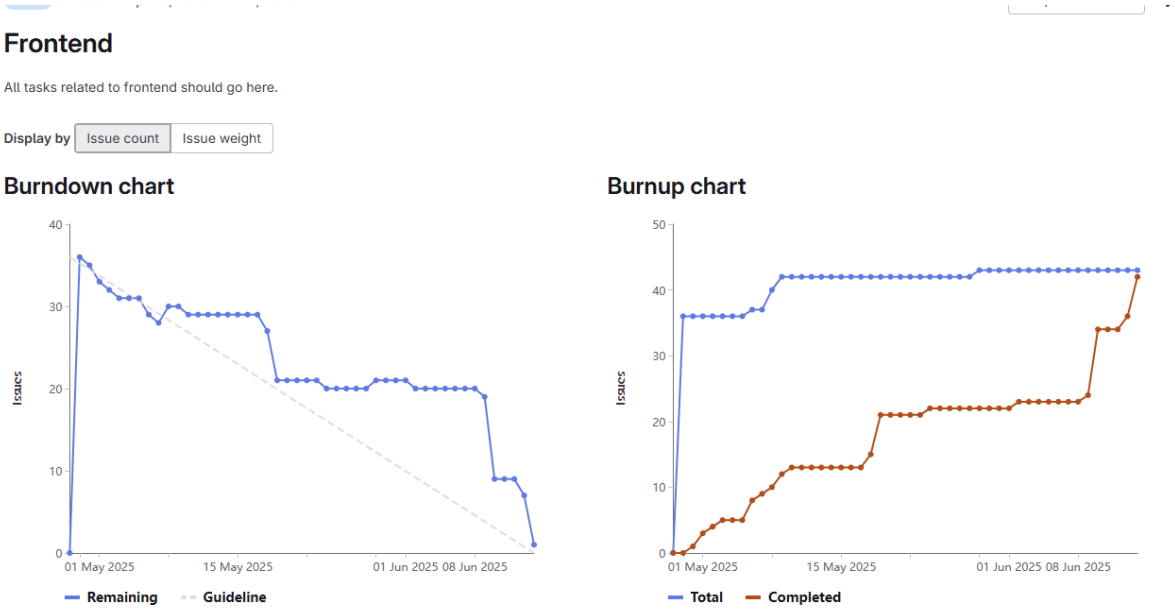


Figure 32: Burndown and Burnup Charts for Frontend Milestone

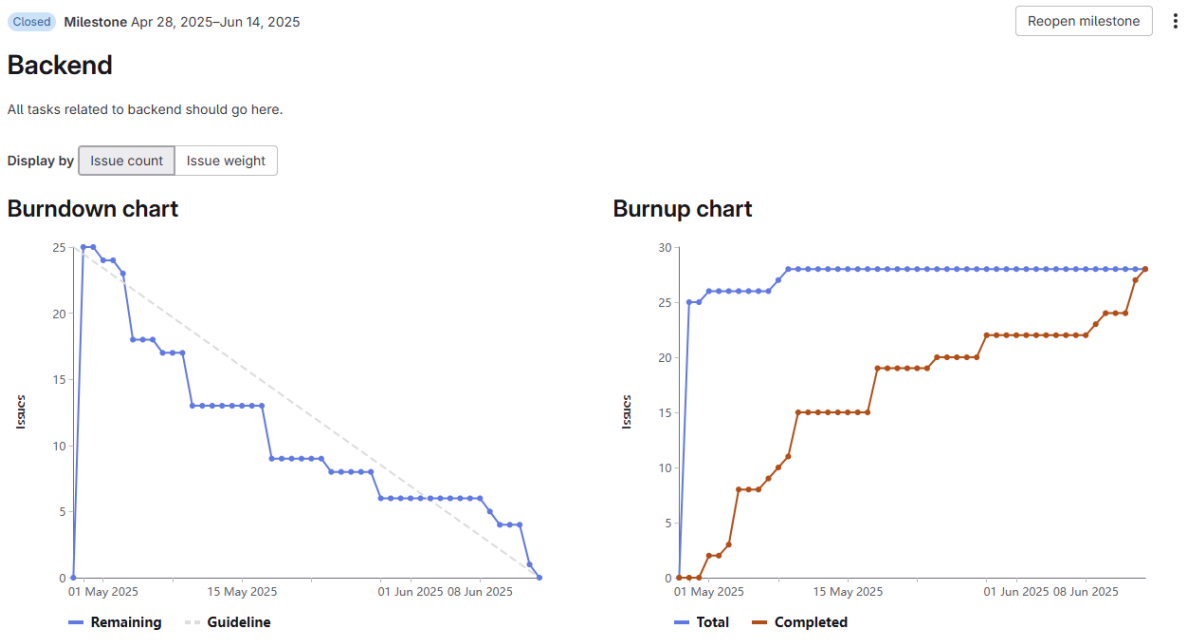


Figure 33: Burndown and Burnup Charts for Backend Milestone

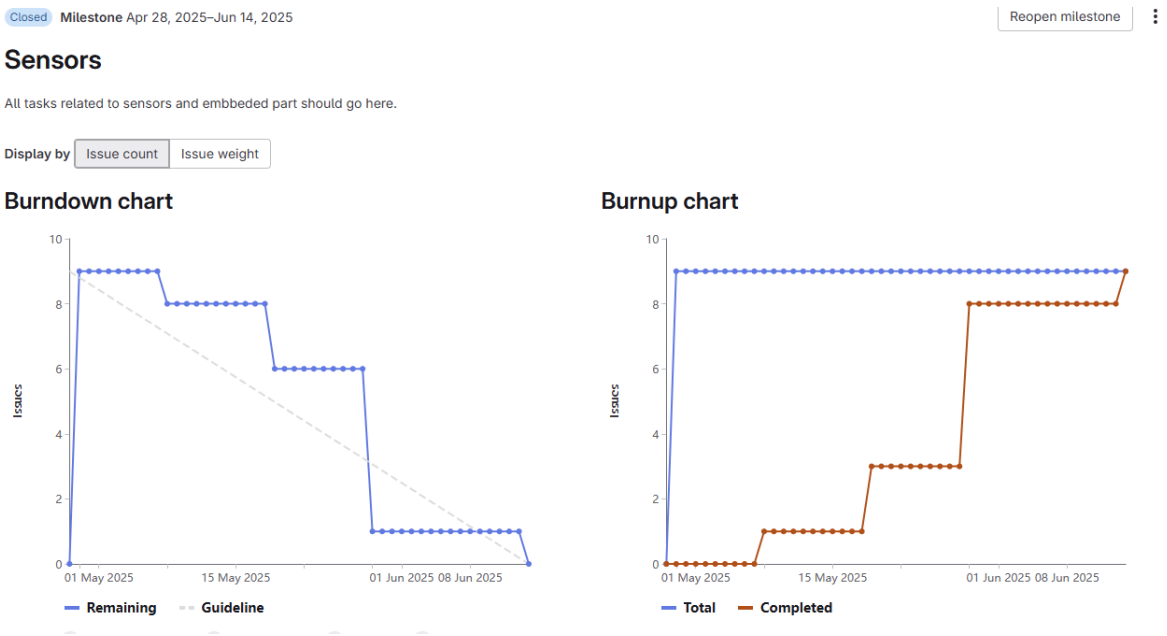


Figure 34: Burndown and Burnup Charts for Hardware Milestone

J Internal rules

J.1 Exploration and Research

J.1.1 Requirements

- All requirements must be written as *SMART* goals (e.g. specify the target time for a feature, required accuracy, etc.).
- Requirements are reviewed continuously with the client; the team seeks clarification on any ambiguous points.

J.1.2 Research and Design Methodology

- Any major change to the application (frontend or backend) must be proposed to the team in advance.
- Implementation may only begin once all relevant stakeholders have agreed on the proposed change.

J.2 Application

J.2.1 Software Quality

- Every merge request (MR) requires at least one approver and must pass the CI pipeline (including style checks, tests, and coverage reports) before merging.
- Integration tests must be introduced by the end of week 8.
- UI tests must be introduced by the end of week 6.

J.2.2 Documentation Quality

- All methods (frontend and backend) must include inline documentation.
- Significant design decisions must be discussed and recorded with the team.
- A final architectural diagram of the application must be produced at project end.
- OpenAPI documentation must be prepared for all service endpoints.
- Each MR must include a concise, descriptive summary of changes; issues and MRs should be cross-referenced where appropriate.
- README files must be kept up to date with any changes to build or run instructions.

J.3 Process

J.3.1 Planning

- Every change must correspond to an issue and an MR.
- Any quick fixes not following the MR/issue template must first be merged into the source branch.
- At the start of each week, tasks for the upcoming week are assigned to each team member.

J.3.2 Team Communication with the TA

- The agenda for each TA meeting must be uploaded to GitLab before the meeting begins.
- A designated minute-taker records notes and uploads them to GitLab after each meeting.
- Urgent questions between meetings should be addressed via Mattermost.

K Division of work

From the beginning of the project, the team aimed to have all the team members involved in working on both frontend and backend. However, over the 10 weeks, some people worked more on some features/parts of the app than others. For instance:

- Mihai focused on developing features of the app such as the report;
- Andrei worked more on the hardware part;
- Matei worked a lot on the backend and infrastructure but contributed significantly to improving the UI of several pages like Pile Details;
- Bogdan worked on various app features including Project Overview and Pile Map;
- Stefan contributed to the application as well as on testing.