



BSc Computer Science
Design Project

Detox@Home

Dmitry Goryachkin
Gustavo Silva Prado
Muhammad Reynard Enriza
Robert Pit
Sam Mulder

Supervisors: Randy Klaassen, Huub Lievestro

November, 2025

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

Abstract

This project presents the design and implementation of a prototype in support of the Detox@Home initiative. The resulting prototype consists of two main components: a **client-facing chat interface** and a **nurse dashboard**. The Chat Agent serves as a guided conversational assistant for the client, capable of asking and responding to preset questions based on the nurse's configuration. It can collect data in the form of questionnaires, remind clients about scheduled tasks, and flag concerning responses for the nurse to review. The Nurse Dashboard, in turn, provides an overview of each client's progress, displays submitted data and chat history, and allows nurses to create or edit questionnaires and manage client schedules.

The system was implemented as a web-based application following a three-tier architecture, consisting of a front-end interface, a Java-based back-end, and a local database for data storage. While the current version serves as an early prototype, it demonstrates the feasibility of an interactive support tool that could, in future iterations, integrate securely with remote databases and healthcare systems.

Contents

1	Introduction	3
2	Requirements	4
2.1	Chat Agent	4
2.2	Data and Storage	4
2.3	Client Requirements	5
2.4	Nurse Requirements	5
2.5	System Requirements	5
2.6	Summary	6
3	System Design	7
3.1	Architecture	7
3.2	Front-end	8
3.2.1	Client Chat Interface	8
3.2.2	Nurse Dashboard	9
3.3	Chat Agent	9
3.4	Data Storage	10
3.5	Summary	10
4	Implementation	11
4.1	Architecture	11
4.2	Front-end	12
4.2.1	Login	12
4.2.2	Client Chat	14
4.2.3	Nurse Dashboard	17
4.3	Back-end	20
4.3.1	Data storage	20
4.3.2	Process handling	21
4.4	Summary	21
5	Testing	23
5.1	Front-end	23
5.1.1	Client Chat	23
5.1.2	Nurse Dashboard	23
5.2	Back-end	24
5.3	Summary	24

6	Project Management	25
6.1	Methodology	25
6.2	Timeline	25
6.3	Roles	26
6.4	Communication	26
6.5	Risk management	27
6.6	Summary	28
7	General Discussion	29
7.1	Limitations	29
7.1.1	Testing	29
7.1.2	Nurse Interface	30
7.1.3	Client Interface	31
7.2	Reflections	32
7.2.1	Team Inner Communication	32
7.2.2	Project Deadlines	33
7.2.3	Client Requirement Communication	33
7.3	Future Work	34
7.3.1	Client Interface	35
7.3.2	Nurse Interface	35
7.3.3	Coding Standards	35
7.4	Summary	36
8	Conclusion	37
A	Project Proposal	39
A	Context	39
B	Problem	39
C	Goals	39
D	Plan	40
E	Functional Requirements	40
B	Chat Mock-Up	42
C	Dashboard Mock-Up	43
D	Initial Database Schema	44
E	Additional Dashboard Screenshots	45
F	Final Database Schema	48

Chapter 1

Introduction

This report presents the design and implementation of a prototype system developed in support of the *Detox@Home*[\[2\]](#) initiative. Detox@Home aims to assist individuals undergoing detoxification treatment from home, providing a safe and structured environment supported by professional supervision. Traditionally, such detox programmes require clients to visit specialised facilities or receive regular home visits from nurses to monitor their progress and well-being. While effective, this process is resource-intensive, demanding substantial time and attention from medical staff for each client. There has been continued research towards addressing this problem, like this thesis[\[3\]](#) and also interest from news-sites like this article[\[4\]](#), for example, showing the increasing recognition of the demand for improving home-based detoxification programmes. Building on these developments, this project explores how a web-based support system can facilitate communication and monitoring between clients and nurses in the context of the Detox@Home initiative.

To address this challenge, three goals were set:

- Design and develop a nurse interface to manage and monitor clients.
- Design and develop an accessible and conversational client interface to assist their routine and submit data.
- Design and develop a system which can store all relevant information and manage communication between all parts of the system.

These goals support the development of a digital system that could support the nurse's day-to-day responsibilities, improve communication with clients, support these clients in their detox and automate routine aspects of data collection and monitoring. By doing so, the system seeks to reduce the workload of healthcare professionals while maintaining the quality of client care and adherence to detox programmes.

The remainder of this report is structured as follows. Chapter 2 presents the initial and refined requirements that guided the system's development. Chapter 3 details the system design, including the architecture, user interfaces, and data handling. Chapter 4 discusses the implementation, describing how the design was translated into code and the technical challenges encountered. Chapter 5 outlines the testing procedures and their outcomes. Chapter 6 covers the project management aspects, including methodology, team roles, communication, and risk management. Chapter 7 provides a discussion of limitations, reflections, and lessons learned. Finally, Chapter 8 concludes the report and summarises future directions for the project.

In addition to describing the technical implementation, this report also reflects on the development process, including requirement refinement, project management, and lessons learned.

Chapter 2

Requirements

This chapter describes the requirements derived from the project description, website[2], expert feedback, official documents[5, 6] and meetings with the project supervisors. The requirements follow the MoSCoW protocol, with especially must and should being keywords. The MoSCoW prioritisation method was used to distinguish between essential requirements ('must'), important but not critical ('should'), desirable ('could'), or optional ('won't'). The different levels of prioritisation per requirement have been set by the team in consultation with the project supervisors. After which, the requirements have been divided into different categories, together describing the desired results. The initial project proposal can be found in Appendix A, as the version below is the revised and updated version of the requirements as they developed throughout the project.

2.1 Chat Agent

- The Chat Agent must respond in a natural, human-like way.
- The Chat Agent must be able to ask a predefined list of questions.
- The Chat Agent must initiate periodic check-ins with clients depending on the detox plan.
- The Chat Agent must be able to ask predefined questions in the form of questionnaires specified by the nurse.
- The Chat Agent must be able to deliver detox program steps configured by nurses.
- The Chat Agent must be able to notify the nurse based on the completion of steps in the Detox.
- The Chat Agent could answer questions based on questions and responses customizable by the nurse.

2.2 Data and Storage

- The system must store all chat transcripts.
- The system must store client medical data.
- The system must store questionnaires to be used by the Chat Agent.
- The system must flag anomalous medical data.

2.3 Client Requirements

- Clients must be able to communicate with the Chat Agent.
- Clients must be able to manually input data.
- Clients must be able to answer questionnaires in chat for short questionnaires and in an overlay for longer questionnaires.
- Clients must be able to ask questions related to their Detox.
- Clients must only be able to access the system via a unique access code generated by the nurse.
- Clients must be able to use the application on both mobile devices and desktop computers.
- Clients must be able to navigate the app with only a basic knowledge of Dutch.
- Clients must be able to use the app with only a basic understanding of mobile devices.
- Clients could have access to accessibility settings and adjust these to their needs.

2.4 Nurse Requirements

- Nurses must be able to log in with a username and password.
- Nurses must be able to view all their clients in one overview.
- Nurses must be able to monitor each client individually, including progress, health data and treatment plan.
- Nurses must be able to see anomalous answers or actions detected by the bot.
- Nurses must be able to configure detox programs for clients, to be executed by the Chat Agent.
- Nurses must be able to view the full chat history of each client.
- Nurses should be able to see the full history of questionnaires answered.
- Nurses could be able to access the systems on phones and tablets (in addition to desktop computers).

2.5 System Requirements

- The system must be implemented as a web application.
- The system must provide a Dutch language interface.
- The system must be stored locally and not rely on other (web)services.

2.6 Summary

Having established the functional and non-functional requirements that define the scope of the Detox@Home prototype, the project could now progress toward determining how these would be realised in practice. The next chapter, therefore, presents the System Design (3), describing the overall architecture, chosen technologies, and design decisions made to fulfil the requirements outlined here.

Chapter 3

System Design

This chapter describes the initial system design in its four different aspects, starting with the overall architecture of the system and the front-end user interfaces. After which, an overview of the back-end system is given, and the Chat Agent design is described. In this chapter, motivation is given for the design decisions made, to sketch a good idea of the initial project design and the philosophy behind it.

3.1 Architecture

The back-end would ideally be designed as a monolithic server program in which each distinct service runs as a dedicated thread within the same process space. Inter-thread communication is achieved through a shared-memory message-passing mechanism. In this mechanism, each thread owns a message queue, functioning as a "mailbox," that it polls periodically for incoming tasks. These queues are not directly accessed by other threads but are registered in a central data structure known as the queue map. This queue map links thread identifiers to their respective message queues. Messages sent between threads are non-blocking and non-compulsory. In other words, a message can request that a thread act, but it cannot force execution, preserving thread autonomy. Thread autonomy, in turn, ensures fair work distribution and execution. Message formats include structured data packets necessary for the requested operation, and strict rules govern which threads are permitted to send specific message types to others, maintaining controlled interaction patterns.

The system includes several service threads. These are the client interface, nurse interface, database interface, chat API interface, webpage repository, and authentication service. Each of these threads is responsible for its own domain logic while collaborating with others via the message-passing system. The main program serves solely as the orchestrator. Which would be responsible for initialising and managing thread life cycles, populating the queue map, and ensuring all services are registered and able to communicate appropriately.

This architecture ensures that system components remain modular, maintainable, and capable of concurrent operation without direct dependencies. Making it perfect for larger use cases and systems with many moving parts like ours. An overview of this system is shown in Figure 3.1.

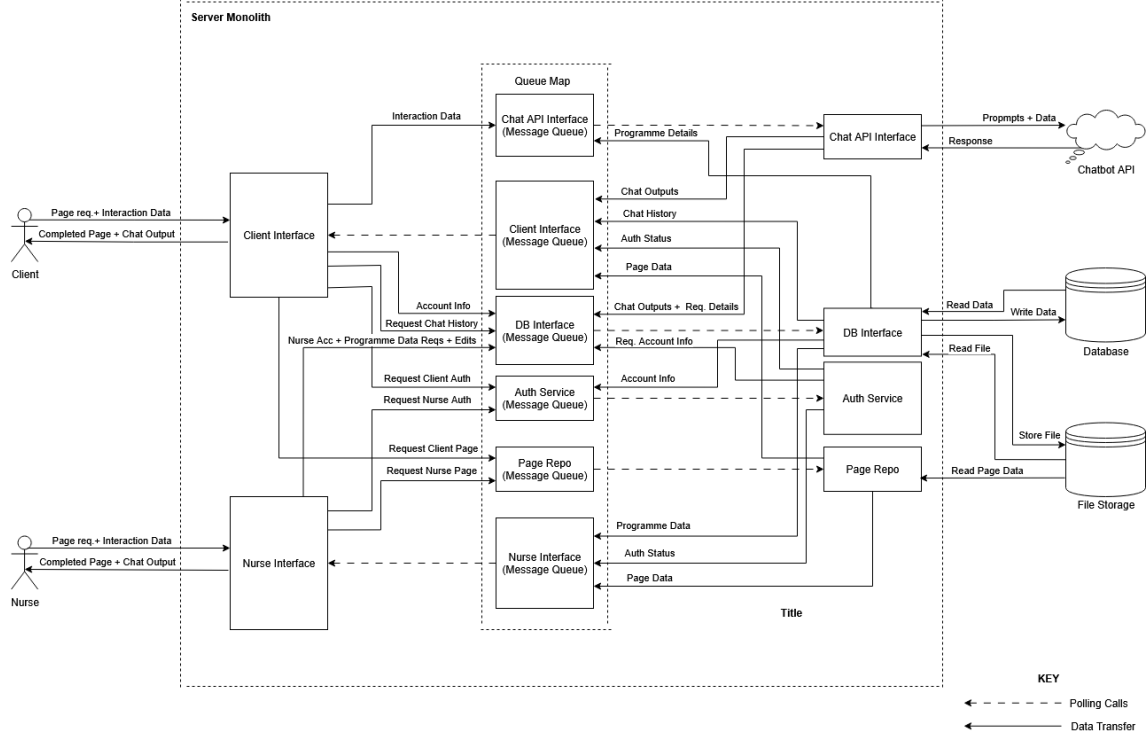


Figure 3.1: Monolithic architecture of the system

3.2 Front-end

3.2.1 Client Chat Interface

Due to how broad the clients are as a user base, the design rationale needed to focus more on approachability and accessibility. It can be assumed that, within a healthcare context, clients are varied in their level of technological literacy. Hence, it is important that the structure and styling of the webpages can at least accommodate smaller screens, as it is likely that smartphones are the most familiar form of interaction with the system. Furthermore, as the clients are detoxing from a substance, it is also under the assumption that clients are varied in their cognitive abilities. As such, the user experience must be familiar enough that a client would not need to relearn the user interface, and instructions are clearly given for their next steps or actions.

To reflect this motivation, the chat page should ideally follow conventions that are common among all chatting apps. The main conventions include:

- Outgoing chat bubbles on the right side.
- Incoming chat bubbles on the left side.
- Timestamps shown for each bubble.
- An input area at the bottom of the screen.

The main goal of this side of the system is to facilitate the collection of health data via the use of questionnaires and to communicate appointments with the client through the use of the Chat Agent. It is thus vital to facilitate this communication through the chat via incoming bubbles, but not overwhelm the client with too much information, such

as sending bubbles too frequently or sending large text bubbles. This design ensures that the Client Requirements (2.3) are met by providing a recognisable environment for their communication.

A mock-up of this chat UI is given in Appendix B, which was made within the Figma platform. In essence, it should facilitate a way to input health measurements and communicate with the Chat Agent, while also being usable on both mobile and computer platforms.

Overall, this design ensures that clients can engage with the chat system intuitively and reliably, regardless of their device or technical proficiency.

3.2.2 Nurse Dashboard

Based on the Nurse Requirements (2.4), it is apparent that the main objective of the dashboard is to display as much information about the client as possible, while also providing functionalities based on the data given. Therefore, the Nurse Dashboard’s design philosophy is more utility-centric; information should ideally be available the moment a nurse logs into the page. They should immediately be able to see the clients registered under them, their respective details, and perform actions that assist in monitoring their detoxification progress.

Hence, the dashboard should ideally support:

- Registering and deregistering of clients
- Viewing client details
- Viewing and configuring detox programmes
- Monitoring the chat history of the client
- Displaying what flags were triggered per client
- Notifying the nurse based on triggered flags
- Creating and configuring questionnaires

A Figma mock-up of this dashboard is given in Appendix C. Besides directly addressing the Nurse Requirements, this design would also partially address how questionnaires could be stored within the system, as outlined in the System Requirements (2.5). Ideally, the questionnaire creation should be scalable in how it handles question types and flexible in how question deletions and question creation are handled.

Furthermore, the security of the Nurse Dashboard was considered from an early stage, given the sensitivity of the data handled. Each nurse account is therefore protected by encrypted passwords, reducing the risk of credential leaks or unauthorised access.

With the front-end interfaces established, attention was turned to the Chat Agent that powers client interaction.

3.3 Chat Agent

For the Chat Agent, we considered 2 different approaches. One of which would be to use an LLM to respond to (most of) the questions and messages sent by the client. The other approach would be to keep the system contained by writing all responses ourselves and only use some LLM (Large Language Model) recognition features to make sure the conversation flows well.

To best adhere to the requirements stated at the start of this report, we chose to go with the contained version of the system. This makes sure we are in control at all stages of the conversation, which would not be the case with LLMs because LLMs can be very unpredictable. Besides that, we are not able to test or train an LLM to its fullest form. In our short time span, this is not feasible to then trust its capabilities of conversing with such a sensitive group of people.

After this decision was made, we came to Rasa. It is a tool for designing conversations with the use of an LLM. This might sound controversial with the decision made before, but the use of LLM is totally controlled. With Rasa, you can let the LLM respond, but you can also choose to write responses yourself. Additionally, the recognition feature is based on LLM, making it particularly useful for conversation flow and answering appropriately. And because it is using our own written responses at all times, there is no risk of harmful or inappropriate messages coming from the bot.

In conclusion, Rasa, which supports integration of controlled LLM recognition features, was therefore chosen. This approach balances safety and flexibility, enabling natural interaction without compromising control or predictability.

3.4 Data Storage

The system was designed using two main mechanisms that handle the different types of data required by the system. Structured data, such as user credentials and general client information, is stored in a standard SQL database using defined schemas. In parallel, more complex or frequently modified data—such as protocol schedules, questionnaires, and their corresponding answers—is stored in JSON files. The file paths to these JSON documents are referenced within the SQL database, allowing the system to retrieve and manage them when needed efficiently. An initial design can be found in Appendix D.

This hybrid approach combines the reliability and integrity of relational storage with the flexibility of JSON files, ensuring that both static and dynamic data can be handled effectively and scaled as the system evolves.

To manage and store this data, a separate server from the one running the main system was used, as illustrated in the original monolithic architecture diagram. The database was hosted on the University of Twente’s *Bronto* server, which runs on PostgreSQL. For the JSON files, Amazon’s S3 storage service was selected due to its reliability, scalability, and widespread use in data-driven applications.

3.5 Summary

This chapter presented the preliminary system design for the Detox@Home prototype, outlining the intended architecture, front-end interfaces, Chat Agent structure, and data storage approach. These designs provided a conceptual foundation and guided the overall development direction.

However, as with most software projects, certain aspects of this design evolved during implementation. Practical considerations, technical limitations, and integration challenges led to several adjustments and refinements to better align the system with its intended functionality.

The following chapter, therefore, describes the Implementation, detailing how the system was realised in practice and highlighting the differences between the initial design and the final prototype.

Chapter 4

Implementation

In this chapter, the implementation is described in detail, together with the challenges that had to be overcome along the way. It explains how this differs from the initial design and why, while also going into the specifics of the system and the way it was implemented.

4.1 Architecture

In large contrast to the planned design of the overall system architecture, the final implementation of the system instead relied on a simpler Three-Tier Architecture rather than the monolithic server. While theoretically ideal for the system's scalability to more users and security via thread safety, in practice, the monolith design was an excessive solution for an otherwise small-scale proof-of-concept.

Generally, the user interface of the Detox@Home web application would involve 3 major web pages:

- A login page
- A dashboard page intended for nurses
- A chatting page intended for clients

Extending their structure, each web page has its own HTML, CSS and JavaScript components, denoting their structure, styling and functionality respectively. The latter two files are linked within the HTML rather than the HTML containing these functionalities/styles, as a measure to avoid clutter between the components via a separation of concerns.

For both clients and nurses, their front-end functionalities involve frequent API calls to the back-end to fetch their individual data from the database and display it within the UI itself. This flow of data works in the opposite direction, where users also make API calls to store or update data in the database. For instance, a nurse can register new clients, which involves the nurse filling in the details about the client. These details are then saved within the database and are fetched when rendering the nurse's dashboard. Each client's details, detox programme data, message history, contact messages, questionnaire answers, flags, and questionnaires (with questions) are stored and updated within this database. Figure 4.1 illustrates this interaction between components, which is further applied to any web pages present in the system that contain a JS file. In essence, the system follows a Three-Tier Architecture by definition.

It is worth noting that these API calls follow RESTful principles, with communication to the back-end Java methods utilising explicit GET, POST, PUT and DELETE requests within the JS. Furthermore, the scalability of the front-end's functionalities depends on the

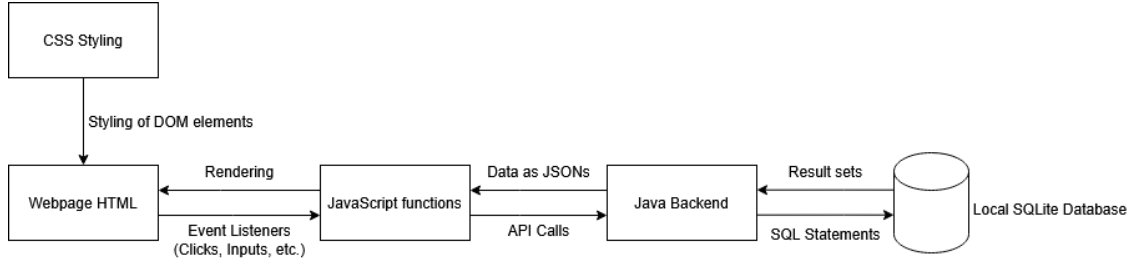


Figure 4.1: Data flow between front-end, back-end and database

implemented API calls that the back-end recognises. If new functionalities are needed that require an entirely different/new set of data, the back-end would need to be updated to accommodate this new need. As is, efforts were taken to keep this data pipeline relatively simple, for future iterations of the Detox@Home project to be able to maintain and create more functionalities.

Due to the nature of this project being an early prototype of the overall Detox@Home system, it was an essential design decision to store the database locally within the system to demonstrate its functionalities. As a result, the performance of the system is unaffected by an external internet connection to a remote server, as all data is stored within the system that has deployed the project.

4.2 Front-end

It is worth highlighting that considerations were made to use React rather than vanilla JavaScript, but this was quickly dismissed, as the risks of learning an unfamiliar library proved unviable for a 10-week design project. Earlier implementations also attempted to use an open-source website builder called WebStudio[7], but this approach was also discontinued due to issues arising from the export of these pages and the difficult navigation of the structure created by the app.

4.2.1 Login

Evidently, both user types (Client and Nurse) will initially land on the login page, as visible in Figure 4.2, to access their respective web canes can register themselves within the system and log in using an email and password, as visible in Figure 4.3, while clients can log in via the token given to them by a nurse. To expand the need for approachability within the system for clients, this login web page also defaults to the "token login" upon landing, so that clients have fewer clicks needed to log in, maintaining a more straightforward user flow.

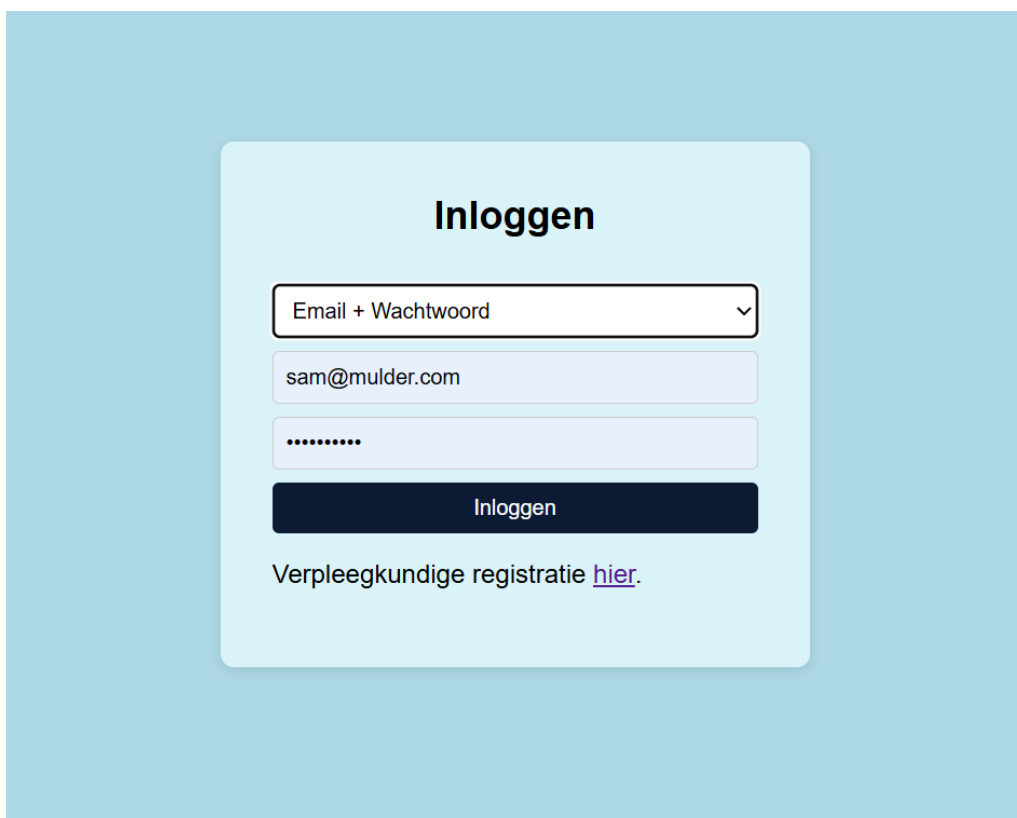
Following the need for security, nurse account passwords are now hashed with SHA-512 and a salt, which minimises the effect of intercepting or unauthorised access to the password of an account. Moreover, account passwords are enforced to have a length of 6 characters to increase this complexity, and account creation is restricted to only nurses who were given an organisational password. This is primarily a measure to prevent clients from registering themselves as nurses within the system, but also a measure to prevent non-users from registering themselves in the system.

However, the current build of the system hardcodes the organisational password within the JS file linked to the Sign Up HTML page (a "subpage" of the login HTML page). This is ideal for demonstrative purposes, but not ideal in real-world deployment.



The login landing page features a light blue background. In the center is a white rounded rectangle with a dark blue header 'Inloggen'. Below the header are two white input fields: the first is labeled 'Inlogcode' and has a dropdown arrow, the second is labeled 'Token'. At the bottom of the white rectangle is a dark blue button with the text 'Inloggen' in white.

Figure 4.2: Login landing page



The nurse login page features a light blue background. In the center is a white rounded rectangle with a dark blue header 'Inloggen'. Below the header is a white input field labeled 'Email + Wachtwoord' with a dropdown arrow. Below this are two light blue input fields: the first contains the email 'sam@mulder.com' and the second contains masked characters '.....'. At the bottom of the white rectangle is a dark blue button with the text 'Inloggen' in white. Below the button, the text 'Verpleegkundige registratie [hier](#).' is displayed.

Figure 4.3: Nurse login page

4.2.2 Client Chat

The chat development started with a separate Chat Agent and chat frontend. The Chat Agent would be built with Rasa, the contextual AI tool, which would then communicate with the front-end chat interface to offer replies and timed events. After 2 weeks of development by one of the team members, it appeared infeasible for us to use this in the actual product. This was caused by ways to get around the preset messages, enabling the default bot behaviour, where it would suddenly start speaking English or reply in unpredictable ways. Also, making sure the bot would follow certain expected behaviour and it keeping track of time were starting to become an issue, which we were not able to solve in the timespan of the project. With the project needing to advance towards a minimum viable product, it became clear that some steps had to be taken, so Rasa was unfortunately dismissed as an option, and the focus was directed towards manually scripted input and output. Making sure essential questions were available to the user to ask and to be answered, and leaving other questions for the nurse to handle. This also allowed us to have more control over timed events and custom, reliable behaviour.

Chat Page

The first page the client would land on is a login page; the functionalities and implementation of this are explained in the previous section. After logging in using the token acquired through the nurse, the client will see the chatting interface as shown in Figure 4.4. Unlike traditional chatting interfaces, however, "chatting" is facilitated using action buttons that work as menu items for asking different questions, similar to dialogue options for online assistants. These menu options are nested within each other, usually with a return button in case the client wishes to ask a different question. For instance, the client may ask a question about their next appointment by going down the menu path of: "I have a question about my schedule" > "When is my next appointment?". The reason for this is due to the chosen definition of "conversational" in the context of the project. In Detox@Home's case, a "conversational Chat Agent" refers to a Chat Agent that responds to preset questions with relevant information, serving as an assistant to the nurse rather than a replacement. In the case that a client wishes to send a message to their nurse, there is a menu option for them to send urgent or carry-over messages through the system to the nurse. These messages can then be discussed using an external communication platform, as the web application does not support live communication between the client and the nurse. All of these options are accessible through a bubble click-through system, as shown in Figure 4.5.

In regard to collecting client data for their detox, they are expected to answer questionnaires truthfully over the course of their programme, which are scheduled by the nurse. These questionnaires display questions differently depending on the question type, namely:

- Open-ended, presented as text input fields
- Measurement inputs, presented as numeric input fields
- Multiple choice, presented as select fields
- Numeric inputs, presented as numeric input fields
- Analogue sliders

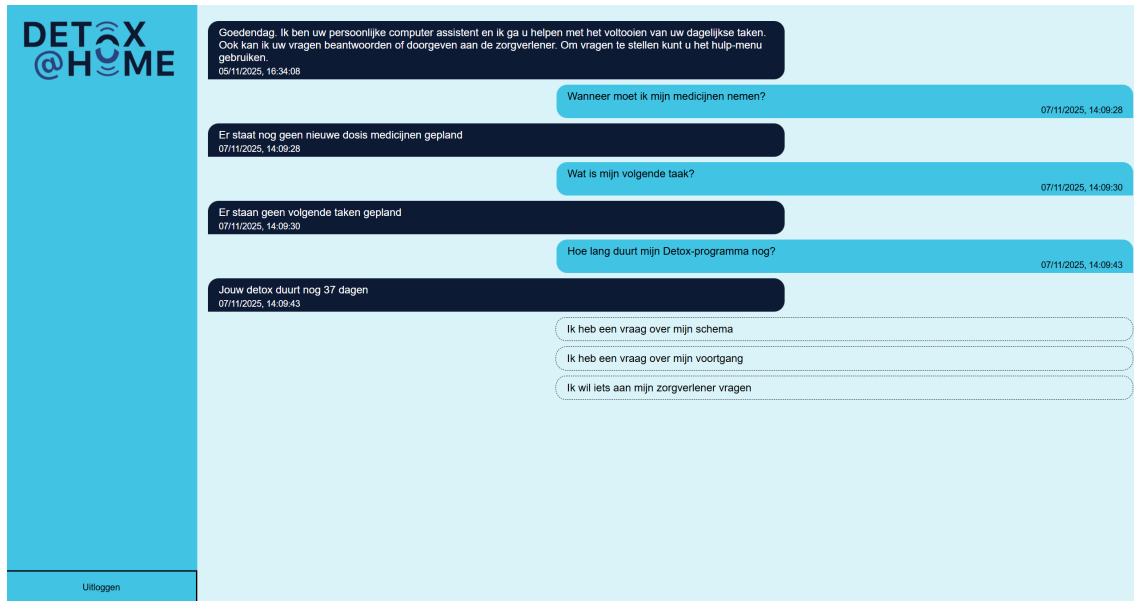


Figure 4.4: Chat page



Figure 4.5: Bubble Question Selector

Chat Agent

The Chat agent always initiates a conversation for the answering of these questions within questionnaires, done at scheduled times set by the nurse. How the client responds to the questionnaires depends on the number of questions. If the questionnaire has fewer than or equal to 3 questions, then answers are provided within the chat itself. Questions are asked as message bubbles by the Chat Agent, with the input area at the bottom of the screen adapting itself to the question type as shown in Figure 4.6. If the questionnaire has more than 3 questions, the Chat Agent will prompt the client to press an action button under its own message to display the questionnaire as a modal, allowing the client to answer in a traditional answer form as shown in Figure 4.7. This change was made because we received feedback on the in-chat system, which was our initial design. In the feedback, it was mentioned that this system may be tiring because of its endless feel of it. So with the form overlay, there is a clear end and a better overview.

After answering a questionnaire, the answers are evaluated by the front-end before being sent to the back-end so that they can be stored correctly. It evaluates all the answers based on flags which are set by the nurse in the dashboard. If the answer is listed as a flag, it will mark this so it can be shown to the nurse. But this is not the only flag showing up to the nurse, the system also checks if there has been an uncompleted task which has been on for half an hour, after which it will remind the client to complete their task. If this is still not done, it will send a flag to the nurse after another half an hour (1 hour total) so that the nurse can take action if they deem this necessary.

One more vital functionality is the timed events, as promptly mentioned above. The

Tijd voor de vragenlijst: Korte vragenlijst. Probeer deze zo eerlijk mogelijk in te vullen zodat u zo goed mogelijk geholpen kan worden. Om de vragenlijst te openen klikt u op de onderstaande knop.
07/11/2025, 14:20:13

Waar heeft u het meeste last van?
07/11/2025, 14:20:13

Hoofdpijn 07/11/2025, 14:20:19

Hoe voel je je? Waar links heel slecht en rechts heel goed is.
07/11/2025, 14:20:19

Verstuur

Figure 4.6: In-chat questionnaire

Hoe voel je je zonder koffie?
Antwoord hier

Hoeveel zin heb je koffie op een schaal van 0-10
0

Hoeveel zin heb in koffie?
[Slider bar with a blue circle in the middle]

Waar heb je het meest zin in?
Selecteer een antwoord ▼

Wat is je bloeddruk?
Bovendruk Onderdruk

Inleveren

Figure 4.7: Questionnaire overlay

system checks every minute for timed events; this includes the uncompleted task reminding and flagging, but also checks the schedule for tasks which need to be completed right now. It refreshes the schedule and checks if there are any tasks due this minute; if so, it executes this task and sends a message to the user to start their task. After which, it expects the questionnaire answers or a confirmation of the task being completed.

Data Flow

The data used and generated by the chat interface is managed through JavaScript fetch requests. When a client logs in, the system loads their chat history from the corresponding JSON file on the server. Subsequent messages are sent immediately to the appropriate storage: dynamic data such as chat messages, schedules, and questionnaire answers are communicated via JSON, while simpler requests, such as retrieving detox program durations or submitting individual health measurements (e.g., heart rate), are handled directly through API calls to the SQL database.

4.2.3 Nurse Dashboard

User Interface

Following the mentioned utility-centric philosophy mentioned in the Nurse Dashboard of System Design (3.2.2), large changes were made to the Nurse Dashboard's layout. As shown in Figure 4.8, a notable difference from the original mock-up is that the main content area is less rounded and has been converted into a scrollable section. The reason for this is due to the original design excessively wasting space, which did not translate properly to non-widescreen displays (e.g. 4:3 displays).

DETOX@HOME

Vragen Inbox

P

Vragenlijsten Beheren

Uitloggen

CLIENT REGISTREREN

Jane Doe

J.Doe

Foo Bar

Jacob Doe

J.Doe

George Doe

Clïëntgegevens

Bewerken

Clïënt Afmelden

Inlogcode cliënt: jd@7SfsBAQH

Naam: Jane Doe

Clïëntnummer: 1

Detox Details

Programma: Alcohol

Duur (Dagen): 30

Huidige dag: 7

Tijd	Schema type	Beschrijving	Getriggerde dagen	
08:00	Medicatie	Ochtendmedicijn	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30	<div></div>
10:00	Consult	Raadpleeg een arts	1,2,3,4,5,6,7,8,9,10	<div></div>
18:00	Vragenlijst	Dagelijkse vragenlijst	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30	<div></div>

Nieuwe taak toevoegen

Antwoordschiedenis

Figure 4.8: Layout of the implemented Nurse Dashboard

Moreover, the cell-by-cell design of the div elements in the original design was not scalable for the possibility of additional details being displayed. An example of this can be observed between the mock-up and the final implementation, where the mock-up originally did not display "contact questions" from the client, as that requirement was only formalised at a later stage of the project with the supervisors. Having each display be rendered as separate sections of the main content area minimised the challenge posed by changing requirements, as each section is isolated and movable within the HTML structure.

It is worth mentioning that the dashboard does not function with narrower screens and is essentially unusable on small mobile devices. It is assumed that the nurse will enter this page solely on workstations such as computers and laptops, with wide displays. As such, there is a lack of responsive web design in the dashboard, as this assumption rendered the need for small-screen flexibility redundant. The dashboard does function properly on tablet devices, ideally on a horizontal layout, as the screen space is often sufficient to display all DOM elements properly. However, the current build of the system does not account for swiping for navigation, making the dashboard prone to visual bugs.

Rendering

Mentioned in the Front-End section of System Design (3.2), the webpage designs were initially prototyped through the Figma platform, which served as a blueprint for the implementation of the website structure and the general navigation flow between pages. Also noted within this same section was the main principle of the dashboard, being the ability to provide immediate information to the nurse using it. How this goal was achieved was by rendering at the initial landing on the web page, by using a large DOMContentLoaded listener event.

The API call within this event makes use of the current nurse's email stored within the cookies to fetch all their registered clients and their respective data as a large JSON object. Each client is then rendered as a sidebar button, with each button running more rendering functions to reset and replace DOM elements within the webpage and listeners, to display the currently selected client. These rendering functions extend to the rendering of each client's questionnaires, the questions within the questionnaires, their triggered flags, chat history, answer history, and contact questions.

Based on this data, the dashboard contains the following functionalities at its core:

- A questionnaire manager to create, delete, and edit questionnaires
- A questionnaire editor to create, delete, and edit questions, their question type, and their flagged answers
- A schedule table to create new tasks for the client and to set the days of the programme which trigger these tasks
- A pane displaying the previous questionnaire answers submitted by the client, sorted by programme date and time
- A pane displaying the chat history of the client
- A flag inbox that shows flagged client answers and inactivity flags, which are timestamped and dismissible
- A contact messages pane containing all the messages sent by the client, divided into Urgent and Carry-over, which are all dismissible

These functionalities serve as the main workflow for a nurse using the Detox@Home system, and screenshots of their presence in the user interface can be found in Appendix E. The latter two features in particular display unique notifications on the sidebar, respective to the client who has either sent a contact message or has triggered a flag. These notifications extend to the main content area and the flag inbox button on the action bar to improve readability.

A major weakness of the current system is the lack of live updating. The system does not update its chat panes, flag inbox, contact message panes or answer history live, and requires either a re-render of the current client or a refresh of the page to display new messages/entries. As such, manual periodic refreshes are needed from the nurse in order to check whether any of the mentioned items have been updated. For instance, if a client triggered a flag by their questionnaire answer, then notification is delayed until the nurse either re-renders the client or refreshes the page.

Questionnaire Editor

The questionnaire editor is the feature with the most functionalities within the dashboard, and its layout can be seen in Figure 4.9. In essence, when the editor is opened, all questions of that questionnaire are rendered, with their question's name, type, range/choices (depending on the type) and flags spawning within the editor overlay. The nurse could then make changes to the question name, the type, the ranges, the choices of an MCQ, and the flags with the editor.

Currently, the system supports five question types:

- Open-ended questions
- Measurement inputs: either Blood Pressure or Heart Rate
- Multiple choice
- Numeric inputs
- Analogue sliders

The screenshot shows the 'DETOX@HOME' interface. The top navigation bar includes 'Vlaggen Inbox', 'Vragenlijsten Beheren', and 'Uitloggen'. The left sidebar lists clients: Jane Doe, Foo Bar, Jacob Doe, and George Doe. The main content area is titled 'Bewerken: Dagelijkse vragenlijst' and contains a list of questions with their respective types and flags. The questions are: 'Hoe voel je je zonder alcohol?' (Open tekst), 'Op een schaal van 0 tot 5,' (Numerieke invoer), 'Geef op de analoge schaal' (Analoog), 'Voer uw hartslag in.' (Meting), 'Voer uw bloeddruk in.' (Meting), and 'Heb je zin om alcohol te drinken?' (Meerkeuzevragen). Each question has a 'Vlaggen' button.

Figure 4.9: Questionnaire Editor layout with sample questions

The latter 3 question types support the creation of flagged answers, while the Measurement type creates flags by design via the lower and upper bounds on the health data expected. The system currently supports only Blood Pressure or Heart Rate as the expected health data, but is scalable to add more health data types. However, the functionality of this is not properly refactored within the JavaScript such that it is easily scalable, meaning that any new question types require the hardcoding of additional conditional statements

(else-ifs) and select options. In contrast, open-ended questions do not provide the option to configure their flag settings, as these questions are flagged by default for the nurse to assess within the inbox.

As the purpose of the dashboard is simply to display the data, exceptions are often thrown whenever the API call is either invalid, uses the wrong RESTful method, or if there is a back-end error (error 500). All of which are currently handled by throwing an exception into the console that lists the problem caused by the HTTP request.

An edge case does exist for which a user without an email stored in the cookies is logged in as a nurse. However, the system does detect and handle this by alerting the user that their session has expired, before redirecting them back to the login page.

The dashboard's functionality relies heavily on a set of RESTful API calls handled by the back-end, described in the following section.

4.3 Back-end

The back-end implementation is split up into 2 sections describing the full implementation and connection between all parts of the system. This includes the data storage consisting of an SQL server and file storage. It also includes the process handling, describing the API as well as the methods used to extract data from afore mentioned data storage.

4.3.1 Data storage

The first step towards building the back-end of the application was the development of the SQL server. There are 6 tables in the SQL database, reflecting 6 different types of data that the system needs to operate. These are:

1. Nurse information
2. Patient information
3. Patient health data (blood pressure and heart rate measurements)
4. Patient tasks
5. Patient flags
6. Contact questions sent by patients to the nurse

A corresponding schema of this database can be seen in Appendix F, displaying the connections between the tables. This schema roughly follows the original designed schema in Appendix D, with the main difference being the additional tables to account for the different functionalities.

This database was built, as originally planned, on a remote server using PostgreSQL. However, contrary to what was originally planned, the project's stakeholders expressed their wish that all the system's data be saved locally on the server that is running the application. In order to achieve this, the database needed to be migrated to a local one using SQLite. The team also considered other options for hosting the local database, such as H2, but SQLite was the simplest to work with, so that is the reason why it was chosen. It was not possible to directly import the schema from PostgreSQL to SQLite, since SQLite is not compatible with PostgreSQL's dialect. To facilitate this migration, the team used the 'SQLiteStudio' tool, which provides a GUI for editing SQLite databases.

SQLite Studio offered a more effective method to manipulate the database and perform this manual migration.

For the JSON files, a directory needed to be created on the computer which is running the server. The team's Amazon S3 buckets did not contain a lot of files, so this migration was significantly simpler than the SQL migration. The back-end calls expect this directory to exist to operate properly. In this directory, 3 things are saved in different JSON files per patient. These are:

1. Patient answers to questionnaires
2. Patient chat history
3. Patient protocol file

4.3.2 Process handling

With the data storage systems now complete, the next step was to create API methods that can be called by the front-end to manipulate the data.

As mentioned before, these methods follow the RESTful API standards. In order for them to be executed, an HTTP request has to be sent, starting with the `'/api/'` path segment, followed by the corresponding address and a request method. These methods can be:

- POST - Used for saving new data
- GET - Used for fetching data
- PUT - Used for updating data
- DELETE - Used for deleting data

Other request methods exist, such as PATCH, but they were not used in this project.

Each API request was designed to correspond to a specific entity in the database, ensuring a clear separation of concerns. For example, patient-related data could be accessed via `/api/patient`, while nurse-related data is available at `/api/nurse`. This structure makes the API both scalable and easy to navigate.

All requests and responses were formatted in JSON, allowing for consistent communication between the front-end and back-end components. Error handling was also implemented in some methods to make the system more robust and prevent system-breaking bugs. In addition, HTTP status codes (e.g., 200, 201, 400, 404, 500) were returned to clearly indicate the outcome of each request.

Finally, cookies were used to call these methods from the front-end. This ensures that patients cannot send requests which should only be accessible to nurses, and nurses cannot send requests intended for patients. However, there is a security limitation here, since if an attacker edits their cookies, the system's security is compromised.

4.4 Summary

The implementation chapter has detailed how the system was brought from the preliminary design into a working prototype. While the initial design suggested a monolithic architecture with controlled threading, the final implementation opted for a simpler Three-Tier Architecture, balancing complexity with the need to demonstrate a functional system

within the project’s limited time frame. The front-end interfaces, client chat agent, nurse dashboard, and back-end API were all implemented according to these design adjustments, with additional considerations made for usability, data handling, and security.

Despite achieving a working prototype, several aspects of the system remain sensitive to real-world constraints and user behaviour. For instance, the chat agent’s conversation flow is manually controlled, the nurse dashboard lacks live updating, and the database is stored locally rather than remotely. These factors highlight potential points of failure or unexpected behaviour during use.

Given these limitations, testing becomes an essential next step. It will not only verify that the system functions as intended but also uncover areas where performance, usability, and reliability may need improvement. The following chapter (5) focuses on evaluating the implemented system, examining whether the requirements defined earlier are met, and identifying possible issues before further iterations or deployment.

Chapter 5

Testing

In this chapter, the testing process is described, together with its results. Also, the design decisions and changes based on the testing are explained.

5.1 Front-end

5.1.1 Client Chat

Testing was done primarily through exploratory methods. Rather than following a fixed set of test cases, different user inputs and conversation paths were tried to observe the agent's responses and identify the weaknesses in the dialogue handling or logic. This hands-on, trial-and-error process allowed for rapid feedback and incremental improvements in usability and robustness.

5.1.2 Nurse Dashboard

Integration and Exploratory tests were performed during the development of the Nurse Dashboard. After each functionality listed in the Nurse Dashboard ([4.2.3](#)) was implemented, immediate testing would occur by redeploying the project within Tomcat and verifying the correctness of the component and its display within the webpage. These tests were then followed by an extensive End-to-End (E2E) test near the end of the system's implementation, where we attempted to test the dashboard by assuming the role of a nurse who manages and monitors their clients. This test in particular involved using all the dashboard functionalities as intended, while also testing for edge cases within input and form fields.

These tests often yielded visual bugs, HTTP request errors, database errors, or backend errors. The latter two failures were often caused by outdated local databases, either missing columns or tables, or oversights within the Java back-end. Each error found, however, assisted development and contributed to making a more reliable system.

Regardless, a limitation of this ad hoc form of testing during development involved the steps taken to test each feature. Each time a feature is implemented, the project and its WAR files would need to be rebuilt and redeployed on Tomcat. This became especially tedious for visual problems, where the focus of the testing was more on the positioning and styling of DOM elements rather than the fetching, rendering and updating of data.

Feedback sessions with the supervisors also assisted in the testing of the system by validating whether requirements are considered "fulfilled" directly through demos - adopting an agile-like form of feedback gathering. These session demos are also an extension of the

E2E testing, as oversights in how certain functionalities were expected to work from the perspective of an outside party were often pointed out.

5.2 Back-end

For testing the API methods in the back-end, a combination of unit tests and manual tests was performed. In order to speed up the development and testing process, the development cycle followed a test-driven approach. This means that every new functionality that was implemented was immediately tested manually via Postman and only committed if fully functioning. Postman is a tool designed for developing, testing, and documenting APIs. It provides an intuitive interface where requests can be constructed, parameters and headers can be adjusted, and responses can be inspected in detail. This allowed for quick verification of API endpoints, ensuring that inputs and outputs matched expectations before integrating them into the main application.

Unit tests were developed for some of the methods, but at the end of the development cycle, the number of different methods escalated quickly, and we did not have time to create unit tests for the new functionalities. Another problem faced by the unit tests is that some of the API methods take in `InputStreams` as a source of data, and it is not easy to emulate the behaviour of this class type in unit tests.

The system was also tested during the feedback sessions with the supervisors, and that was the closest the team got to User Testing. Unfortunately, the conclusion of the product's development came way too late, and we did not have time to perform user tests. The feedback from these sessions was taken into account and implemented into the system.

Overall, the system's back-end was mainly tested and validated manually, by testing the API methods in Postman.

5.3 Summary

The testing phase provided crucial insights into the system's functionality, usability, and reliability, revealing both successes and areas needing improvement. Challenges encountered during testing, such as visual inconsistencies, API errors, and the limitations of manual unit testing, informed decisions on development priorities and resource allocation. These insights not only shaped the refinement of the prototype but also highlighted the importance of structured planning, task distribution, and team coordination, all of which are addressed in the following Project Management chapter.

Chapter 6

Project Management

In this chapter, the management of the project is described. We describe the way our team handled the obstacles and challenges that came with the project. Consisting of the methodology, timeline, roles, communication and risk management.

6.1 Methodology

This project's time division was structured using a combination of the Click-up Kanban Board[1] and broader deadlines. At the start, we made a broad timeline detailing the steps in which we would like to complete the project, after which we would use Click-up to assign tasks and divide the larger steps into actionable parts and features. Every week, we would evaluate the progress together with our supervisor and receive feedback and/or ask questions about the project. Within the week, we would have regular meet-ups based on the demands and progress, which we would discuss and reflect upon before the supervisor meeting, so that the results could be shared and reviewed.

As for version control and file management we used Git. Git ensures that at any time files can be stored and kept up to date without preventing others to work on those same files. By using separate branches for the different parts of the project the team managed to seamlessly work together and later merge everything together into the final product.

6.2 Timeline

The timeline at the start of the project was very broad, with only bigger deadlines and general goals to work towards. It looked as follows:

- First Project Proposal - 12/09
- Finalising Project Proposal - 16/09
- Web-app Layout - 26/09
- Nurse Dashboard / Account linking - 10/10
- Chat System and agent - 17/10
- Minimum Viable Product - 24/10
- Deadline Project and Poster presentation - 7/11

In hindsight, it would have been a solid plan if we had managed to keep it, which unfortunately was not the case. It started strong, and we did manage to have a finished project proposal on 16/09, after which we started on the Web-app layout. But then it started to slack, since we split up the team into their own separate parts, some of us had 3 to 4 weeks before the first deadline/goal in the timeline. This led to some unproductive work and endless reiterations without actually getting real work done. This had to be compensated for towards the end, making the last 2 to 3 weeks very work-intensive and hasty. Every time we had an upcoming deadline or goal, we would make a smaller timeline and divide tasks as described in the methodology.

6.3 Roles

For the roles within our teams, we had some clear and some a little more vague ones. In the first week of the project, after getting to know each other a little better, we assigned a project leader who would keep track of deadlines and make sure we kept moving forward as a team. The other roles were assigned once we needed to split up work or tasks. This was all based on the different expertise and interests of the different team members, and of course, with full collaboration and insight of the whole team. The full list of responsibilities and roles as they were at the end of the project is listed below.

- Dmitry: Head of Testing, Technical Specialist and Nurse-Interface developer
- Gustavo: Communication Manager, Database Designer, Head Back-end Developer, Manual writer and Version Control Manager
- Muhammad: Head Nurse-Interface developer, Minute-taker and Document Manager
- Robert: Head Front-End Design, Presentation and Poster creator and Translator
- Sam: Project Leader, Translator, Head Client-Interface developer and Main Editor

The project report and general communication with the client were a group responsibility and were therefore applicable to and fulfilled by every member.

6.4 Communication

The online communication within the team was done through 2 major platforms. The first being Discord and the other WhatsApp. At the start of the project, expectations were set to ensure good communications and prevent irritation. These expectations were set to work/reply within 'standard' work-hours, meaning 8:45-17:30 on weekdays. Throughout the project, this was mostly adhered to, apart from the small intervention we had to plan because of one team member being particularly unresponsive during the agreed-upon times. After this intervention, online communications were working well.

In-person communication (or voice calls) came with its own challenges in this group. This was because 2 team members regularly clashed when decisions had to be made. To solve this, the project leader would, after hearing the arguments, make the eventual decision, and the 2 team members would try working on separate parts of the project. Apart from this, the in-person communications were smooth and enjoyable.

6.5 Risk management

Risk	Likelihood	Impact	Our Response
Requirement changes by Client	Medium	High	Early requirement analysis, continuous feedback sessions, so adaptations can be made early.
SQL Injections and Security breaches	Low	High	Using prepared statements and hashed passwords.
Database outages	Low	High	Keep the database local and internal, so the system does not rely on external servers.
Integration issues between front-end and back-end APIs	Medium	High	Defined clear RESTful endpoints early and tested API routes frequently with dummy data to ensure consistency.
Hardcoded organisational password in front-end	High	High	Acceptable for prototype, but documented as a critical risk for production; future iterations will store such data securely in the back-end or environment variables.
Loss of data or inconsistent database state	Medium	High	Used local backups during testing and ensured CRUD operations followed a consistent validation pattern before database writes.
Security vulnerabilities due to local database and exposed credentials	High	High	Limited deployment to local prototype only; planned migration to a secure hosted environment for future versions.
Learning curve or instability from using unfamiliar libraries (e.g., Rasa, WebStudio)	High	Medium	Quickly assessed feasibility within early iterations and pivoted to a simpler hand-coded JavaScript approach.
Non-responsiveness or layout issues on different devices	Medium	Medium	Restricted Nurse Dashboard to desktop use; responsive redesign planned for future iterations.
Lack of real-time updates in dashboard	Medium	Medium	Documented limitation; plan to implement live updates using WebSockets in later versions.
Client chat misbehaviour or failure of timed events	Medium	High	Implemented frequent local testing and logging of timed events; fallback notifications provided to nurses if necessary.
Unhandled API errors causing crashes or confusing UI	Medium	Medium	Implemented console error logging and basic user alerts for expired sessions or invalid requests.

Table 6.1: Overview of risks, their likelihood, impact and our response

6.6 Summary

Having outlined the methodology, timeline, roles, communication, and risk management practices that guided the project, it is now possible to reflect on the effectiveness of these approaches. The successes, challenges, and lessons learned from managing the project provide important context for evaluating the system's development, its limitations, and potential improvements. The following chapter discusses these reflections, critically analysing both the process and the final prototype and comparing this to the goals set at the start of the project.

Chapter 7

General Discussion

In this chapter, the results and process of the design and development process undertaken throughout the ten-week project are discussed. Discussing both the constraints encountered during our work on the project and the personal and technical insights gained through the process. This chapter aims to examine the effectiveness of the chosen methods, technologies, and design decisions in relation to the project’s objectives, as well as to reflect on the areas where improvements could be made. The discussion is divided into two main sections: Limitations, which addresses the challenges and restrictions that influenced the outcome, and Reflection, which considers how exactly the choices of tools and communication we made influenced the outcome of the project

7.1 Limitations

7.1.1 Testing

Testing practices throughout the project evolved in parallel with the system’s development and changing technical requirements. As features and integrations were added or modified, the testing process was frequently adapted to accommodate new conditions. While this iterative approach supported rapid prototyping and feature delivery, it also led to inconsistencies in how testing was planned and executed across different stages of development. The focus on functionality and implementation speed, though appropriate within the limited timeframe, ultimately constrained the depth and structure of the project’s overall testing framework.

A key limitation is in the area of unit testing, which remains significantly underdeveloped due to shifting requirements and the rapid pace of iteration. Only a small portion of the API receives consistent unit test coverage, leaving much of its functionality untested at a granular level. The API’s architecture, designed primarily with functional completeness in mind, introduces several challenges that complicate testing efforts. Its inherent statefulness, resulting from continuous interactions with the database and user-facing interfaces, makes it difficult to isolate components for reliable automated testing. Furthermore, the use of complex data objects such as input streams and the need to maintain live database connections introduce dependencies that standard unit testing frameworks struggle to replicate.

Compounding these difficulties, even basic test cases require cloned database schemas to simulate operational conditions. These setups are not only time-consuming but also risk introducing inconsistent or residual data if a test sequence fails unexpectedly (again, due to its stateful nature). This creates potential for “garbage data” within the testing

environment, reducing confidence in results and discouraging the team from expanding test coverage further. These structural complications highlight the importance of early design decisions for testability - an aspect that could be revisited in future iterations to improve maintainability and verification accuracy.

In place of comprehensive automated testing, we adopt a more ad hoc, on-the-fly approach to verification. This involves multiple team members manually testing API endpoints and system functionalities using tools such as Postman, as well as through direct interaction with the user interfaces. While this does indeed allow for quick validation of newly implemented features, this process depends heavily on the testers' intuition and familiarity with the system, which limits its thoroughness. Without systematic test cases or coverage tracking, this testing process can not guarantee that all potential failure conditions or edge cases were explored.

Finally, user testing was not conducted within the project timeframe. The primary cause was the late completion of key functionalities, driven by shifting feature requirements and integration delays, which left insufficient time to recruit representative users or expert groups. The absence of structured user evaluation meant that feedback on usability, accessibility, and workflow efficiency could not be gathered systematically. This lack of external testing limited the project's ability to validate its design decisions and identify pain points from the perspective of real-world users. If the platform were to progress toward a production-ready state, conducting targeted usability testing and structured peer reviews would be essential steps in ensuring both functional reliability and user-centred design.

7.1.2 Nurse Interface

The nurse interface successfully delivers all the core functionalities initially defined in the project scope, as well as several additional requirements identified during iterative development. It provides the essential tools for task scheduling, question management, and client monitoring, fulfilling its functional goals effectively. However, in its current state, the interface lacks a number of quality-of-life features that would make it more intuitive, efficient, and resilient against user error. While the system performs its required operations, several aspects of its user experience and design could be refined to align better with modern usability expectations and to support more seamless day-to-day use.

One notable limitation lies in the way discrete answer range questions are configured. At present, any flagged values within a question set must be manually specified, rather than defined through a range or comparative logic. This approach increases the potential for input mistakes, particularly when dealing with complex forms or multiple variations of the same question. Implementing a more flexible rule-based or range-based input mechanism would streamline the setup process and reduce repetitive manual entry, making the interface more efficient and less error-prone for nurses managing numerous data points.

Another major limitation concerns the task scheduling system, which currently uses a CSV-style text format to represent scheduled days rather than, say, a more visual, calendar-based interface. While this design functions adequately, it becomes increasingly cumbersome as the number of scheduled days grows - particularly for long-term care programmes that may span hundreds of days. A graphical calendar or timeline view would provide a clearer overview of ongoing and upcoming tasks, allowing users to manage and adjust schedules with greater ease and precision. The current method, though technically sufficient, demands unnecessary effort from users and can contribute to fatigue or scheduling errors over time.

Additionally, the task scheduler's inability to edit or reorganise existing tasks represents another practical shortcoming. At present, nurses can only delete and recreate tasks to

make adjustments, rather than directly editing task parameters or rearranging their order. This restriction can inhibit workflow continuity and may lead to inefficiencies, especially when frequent updates to task lists are required. Iterating with in-place editing and sorting capabilities would greatly enhance usability and reduce the risk of data inconsistencies introduced by repeated deletion and re-entry.

Finally, although the interface achieves its required functionality, it was developed using minimal styling and without a dedicated UI library, which is atypical for modern web applications. While this approach simplified the development process and ensured the system remained lightweight, it also limited visual consistency, discoverability of features, and the overall intuitiveness of the interface. This minimalist implementation can be viewed as both a practical decision and a limitation, as it sacrifices some of the polish and ergonomic benefits that a more structured design system or UI framework could offer.

Overall, while the nurse interface meets the project's functional objectives, it does so in a form that prioritises technical completion over usability refinement. These design trade-offs, though understandable within the project's scope and time frame, highlight key areas for improvement should the platform be developed further into a production-ready system.

7.1.3 Client Interface

While the client-facing interface succeeds in delivering the essential chat functionality, several limitations in its current implementation reduce its overall accessibility and usability. These constraints primarily concern the current lack of accessibility settings and the absence of certain design features that would allow the interface to adapt effectively to different user needs and device types. As a result, the interface is limited in its usability, which is a crucial concern given our requirements of ease of use.

A significant limitation is the absence of a dark mode and high-contrast mode, which are essential for improving visual comfort and reducing eye strain, particularly for users working in low-light environments or with visual sensitivities. Without these options, the visual experience is restricted to a single light-themed design that may not suit all contexts or preferences. Incorporating such modes would not only improve accessibility but also align the interface with modern usability standards expected in modern web applications.

The interface also lacks screen reader support, which poses a barrier for users with visual impairments who rely on assistive technologies to navigate digital environments. This omission limits the inclusiveness of the system, making it difficult or impossible for users with visual disabilities to engage meaningfully with the chat functionality. As well, a related concern is the absence of dyslexia-friendly text options. The current styling does not offer adjustments to font style, spacing, or background overlays, which are known to aid readability for users with dyslexia or other reading difficulties. Implementing such customisation and support options would help to remove unnecessary barriers to communication and make the system more accommodating for a broader range of users.

Finally, the interface is somewhat lacking in its usability on different resolutions, limiting its use on smaller screens such as tablets and mobile devices. The layout and text scaling do not adapt perfectly to different viewports, resulting in reduced readability and interaction comfort when the system is accessed outside a desktop environment. This rigidity is also a significant concern, as users today increasingly expect consistent and functional experiences across devices. Ensuring that the interface is responsive would significantly enhance its practicality and overall reach.

7.2 Reflections

7.2.1 Team Inner Communication

As mentioned before, team communication organisations were primarily managed through a combination of WhatsApp and Discord, each serving distinct roles in the workflow. WhatsApp was used for quick coordination, scheduling meetings, and organising work sessions, allowing the team to stay connected and responsive outside of development hours. Discord, on the other hand, functioned as the main hub for in-session collaboration. Dedicated channels were created to separate discussions on backend development, frontend design, documentation, and testing, ensuring that active conversations and resource sharing remained organised and accessible. This division of tools provided both immediacy and structure, balancing casual communication with the need for focused project discussions.

The team operated under a democratic and consensus-based decision-making model, where most major decisions were agreed upon collectively. Specific, well-defined tasks were assigned to individual members according to their technical strengths and availability, but overarching decisions regarding design direction, architecture, or feature scope were typically reached through group discussion. This inclusive structure proved valuable in maintaining team cohesion and preventing individuals from being isolated or assigned to unfamiliar tasks. Every member was allowed to contribute to planning and execution, creating a sense of shared ownership and responsibility across the project. This approach also facilitated rapid iteration, as members could quickly step in to assist others or reallocate effort as priorities shifted.

Compared to a more hierarchical or role-bound system, this flexible model provided significant adaptability. In a project characterised by rapidly changing requirements and evolving technical goals, the ability to reassign focus dynamically was a major strength. Rather than having specialists idle when their specific tasks were paused or completed, the team could pivot collectively, redistributing workload where it was most needed. This adaptability helped the group maintain momentum even during periods of uncertainty or redefinition of objectives, a factor that was crucial to meeting the project’s evolving scope.

However, the same lack of formal structure that enabled flexibility also introduced communication challenges. Without clearly defined roles or a formal hierarchy, there were occasions where responsibilities overlapped or were left ambiguous. Misunderstandings occasionally arose regarding task ownership, leading to duplicated work, idle time, or progress bottlenecks while the team clarified who was responsible for what. These issues, while not constant, highlighted the limitations of relying solely on informal communication channels and consensus-driven coordination when working under tight deadlines.

Another difficulty stemmed from the technical autonomy granted to each team member within their assigned tasks. While individuals were encouraged to follow general project guidelines, they also had broad discretion in selecting tools, libraries, or implementation methods. In several instances, this led to technical decisions being made independently and diverging from the team’s original architectural plan. Although such deviations were often made in good faith and supported by sound reasoning, they created inconsistencies or integration issues later in development. These problems were compounded by the absence of detailed documentation for shared technical standards and by relatively loose communication practices concerning technical changes. A more structured approach to documenting and communicating such decisions could have mitigated these complications.

Despite these occasional setbacks, the overall communication framework proved effective in supporting progress throughout the project. The openness and inclusivity of the team’s collaborative style allowed members to adapt to frequent changes in scope and

requirements with minimal friction. The combination of accessible communication tools, mutual trust, and shared responsibility ensured that challenges were addressed collectively rather than in isolation. While the experience revealed several opportunities for improving coordination and decision-making protocols, the system ultimately contributed to the team's ability to deliver a functioning prototype under evolving conditions.

7.2.2 Project Deadlines

Project deadlines were established for major milestones and core functionalities, serving as reference points to guide overall progress. These high-level deadlines marked significant stages in development—such as the completion of proposals and designs, the implementation of key frontend features, and the integration of the client and nurse interfaces. While these targets provided direction and helped keep the team aligned toward shared objectives, the planning process lacked finer-grained timelines between these broader milestones. As a result, progress within each phase was not consistently structured or measured, which affected both the pacing of work and the even distribution of effort over the project's duration.

The absence of granular deadlines was largely a consequence of uncertainty during the early stages of the project. When initial timelines were set, the specific implementation details were still undefined, making it difficult to produce accurate estimates for individual components. This uncertainty was tied closely to the project's evolving requirements and the iterative nature of the design process. Because the technical architecture and feature specifications were still in flux, establishing detailed schedules at that stage would likely have been premature and potentially misleading. However, this ambiguity also limited the team's ability to plan and manage workload effectively once development began in earnest.

Without smaller, well-defined checkpoints, the team occasionally experienced uneven distribution of work and time pressure. Some tasks extended longer than anticipated, while others were compressed into short periods near milestone deadlines. This pattern sometimes resulted in rushed implementation or incomplete features that required revision later in the development cycle. The lack of intermediate evaluation points made it harder to identify emerging bottlenecks early, and opportunities for proactive course correction were occasionally missed. While the team successfully met the major deadlines overall, this came at the cost of increased stress and occasional compromises in polish or code quality.

It is important to note, however, that this flexible approach was not without its advantages. The absence of rigid sub-deadlines allowed the team to adapt more easily to shifting requirements and evolving technical insights. As new challenges or opportunities arose, the group was able to redirect effort and adjust priorities without being constrained by overly strict scheduling. This adaptability proved valuable in a project characterised by rapid iteration and continuous change. The broader milestones served as a useful "North Star", keeping the team oriented toward long-term goals even as short-term details fluctuated.

7.2.3 Client Requirement Communication

Client communication was simultaneously one of the most successful and most challenging aspects of the project. Throughout development, the team maintained a consistent line of contact with both the client and the project supervisor, holding regular in-person meetings to demonstrate progress and clarify expectations. These sessions were constructive and collaborative, with the client and supervisor both remaining responsive and willing to provide feedback or guidance whenever requested. This open communication environment

allowed the team to stay aligned with the client’s overall vision and to make iterative adjustments based on their evolving needs and observations.

Despite this strong engagement, several significant communication challenges emerged over the course of requirement definition. One of the primary difficulties stemmed from the fact that neither the client nor the supervisor nor any members of the development team were native English speakers. While all parties communicated effectively overall, subtle ambiguities in wording and phrasing occasionally led to misinterpretations of intent. This was particularly evident in discussions surrounding the scope and purpose of key system features, where differing understandings of certain technical or conceptual terms caused delays and confusion.

The most notable example of this was the debate surrounding the terms "conversational" and "Chat Agent". These terms were central to defining the system’s interaction model but carried different implications for each stakeholder. For the client, "conversational" referred to a natural and guided interaction flow between nurse and client interfaces, whereas some team members initially interpreted it as referring to an autonomous, AI-driven chatbot system. Given the team’s awareness of the ethical and technical concerns surrounding the use of AI in healthcare contexts, considerable caution was applied in exploring this functionality. Over the course of roughly three weeks, multiple design proposals and prototypes were developed, reviewed, and subsequently discarded as the team and client worked to reconcile their differing expectations. This prolonged discussion significantly delayed progress during the early stages of development, but ultimately resulted in a more clearly defined and responsible feature scope.

The root cause of these difficulties lay in the vagueness of the initial requirements and the lack of a fully solidified shared understanding of the project’s intended features. Early requirement documentation focused more on the desired effects or outcomes, rather than on defining specific, measurable functionalities. This ambiguity left room for interpretation and debate later in the process, especially when new features were introduced or existing ones revised. In hindsight, the team recognises that more detailed requirement gathering and specification at the outset would have greatly reduced confusion and rework during development.

For future projects, a more structured and deliberate approach to requirement communication will be essential. Conducting a comprehensive series of initial workshops or requirement-definition meetings would help ensure that the client’s vision is translated into concrete, well-scoped features early on. Supplementing verbal communication with visual aids such as mock-ups, user stories, and flow diagrams could also help bridge potential language or conceptual gaps. Establishing a shared glossary of key terms at the beginning of the project would further reduce the likelihood of similar misunderstandings. By investing more time in these early stages, future teams can ensure that the project’s direction is understood uniformly across all stakeholders, resulting in a smoother development process and more efficient collaboration overall.

7.3 Future Work

While the final result overall does satisfy the overall functional requirements that were arrived at in the design process, there is still much room for improvement, as discussed before. The key areas for improvement are in the client and nurse interfaces, along with several key coding standards that must be improved for developer ergonomics and maintainable design.

7.3.1 Client Interface

First key improvements would have to be implemented in the client interface. Chiefly, all of it's missing a variety of accessibility features as discussed in [7.1.3](#), where chief among them, the following must be implemented to improve ease of use and accessibility to clients of varying mental and physical states:

1. Dark Mode
2. Reader Support
3. Dyslexia friendly font options

As well, the bare minimum use of any styling at all ought to be resolved at one point or another. To be done through using modern frameworks to bring UI components on par with a more modern look that most clients would anticipate. Though this is a purely aesthetic issue, this would be a low-priority task.

7.3.2 Nurse Interface

The second key area of improvement is the nurse interface. As discussed in [7.1.2](#), while the interface does implement all of the strictly necessary functionality, it does not do so in a strictly intuitive or convenient way. This will result in higher erroneous input from the end-user nurses and worsen the UX in general. As such, the following must be implemented:

1. Task Sorting (visual order)
2. Task Editing
3. Task Day Setting (to not be a CSV interface)
4. Proper history for all events and dismissed items, be it flags or messages
5. Live updates (currently must refresh for new data)

Aesthetics are not as much a concern here, even less than mentioned in the previous section. However, a low-priority enhancement in any future iterations can also be to make use of a proper UI framework for the interface, both for the ergonomics of use and looks.

7.3.3 Coding Standards

As the code was made primarily for functionality and rapid iteration, it achieved its purpose at the cost of maintainability and standardisation. As a result, every part of the codebase suffers from low code reuse and a lack of abstractions or separations. Resulting in a lot of boilerplate and large code files, both in the front and the back. Additionally, the lack of established naming, structure, and documentation standards means that anyone trying to work on someone else's work will waste some time adapting to their style. So, to remedy these ongoing inefficiencies in working on this codebase, the following must be done:

1. Proper Naming Conventions for each part, established and enforced
2. Proper Coding Style Conventions for each part, established and enforced
3. Proper Documentation requirements for each part, established and enforced
4. Code refactored to comply with new standards
5. Codebase refactored to allow for more code reuse and abstraction

7.4 Summary

The discussion has reflected on both the technical and organisational aspects of the project, with careful consideration of how well the initial project goals were achieved. While the system meets the functional requirements, areas such as accessibility, interface intuitiveness, testing coverage, and code maintainability require further attention. Equally important, reflections on team communication, deadline management, and client requirement handling provide insights into how project execution could be improved in future iterations. Collectively, these observations tie the project outcomes back to the original objectives and set the stage for the conclusions, offering guidance for continued development and refinement.

Despite the limits encountered and various side effects of our process, the project achieves all the goals it set out to fulfil. The team successfully managed to design and develop a system that, in the context of a detox program, allows the nurse to manage and monitor the clients through a dedicated interface; the clients themselves to have an easy to use, easy to understand chat interface for a conversational agent that assists them in their routine; and tie all of it with a back-end system that can manage and control all of that information and interaction. While some of the minor points can be improved upon as discussed, the overall objectives of the project were fulfilled.

Chapter 8

Conclusion

This project successfully delivered a functioning prototype for the Detox@Home initiative, clearly showing the feasibility and usefulness of such a system. Throughout the course of the project, there were multiple obstacles and changes that needed to be made. All of these accumulated into a final product that has a lot of differences compared to the initial design and the proposal. All of these changes were made for good reason and were made in discussion with the supervisors. However, despite the product looking different than expected, the goals and requirements set for the project were achieved. Clients can easily use the Chat Agent to ask and answer questions. The Chat Agent will respond to these questions and answer naturally, and notify the nurse of any concerns. The nurse, in turn, can see each client's progress, updates, and irregularities easily and clearly.

It is now for the Detox@Home initiative to take this project and continue developing and extending it to better fit the needs of both nurses and clients in the world of detoxing. The team is proud to have contributed to a project with meaningful social impact and is happy to have played a part in supporting this positive change.

Bibliography

- [1] ClickUp. Kanban-board, November 2025. URL: <https://clickup.com/features/kanban-board>.
- [2] Detox@Home. Project, Oktober 2025. URL: <https://detoxathome.nl/>.
- [3] L.A.W. Spelbos. Development of a smart pill dispenser to support patients with substance use disorder during home detoxification. Thesis, University of Twente, Enschede, July 2024. URL: <https://purl.utwente.nl/essays/101875>.
- [4] Journalistiek Team. Thuismonitoring bij ontwenning verslavende middelen. *ICT Health*. URL: <https://www.icthealth.nl/nieuws/thuismonitoring-bij-ontwenning-verslavende-middelen>.
- [5] Tactus verslavingszorg. Stappenplan detox@home alcoholdetox. Sensitive document outlining a concrete detox program, in this case for alcohol addiction.
- [6] Tactus verslavingszorg. Werkwijze ambulante detox. Sensitive document about the steps to take in detoxing.
- [7] WebStudio. Webstudio – advanced open source website builder, Oktober 2025. URL: <https://webstudio.is/>.

Appendix A

Project Proposal

The project proposal follows the context->problem->goals->plan format, with functional requirements also included in the plan. In this proposal, we try to explain the context of the problem, the problem itself and how we are planning on solving it. With this, also the goals we set for ourselves. The functional requirements follow the MoSCoW method in setting priorities.

A Context

Detoxing is a process in which someone with an addiction is guided through quitting the harmful substance they are currently addicted. This can be done in special facilities, but also at home. When done at home, they need regular monitoring to ensure that they do not relapse or have health-related problems.

Definitions:

Chat agent: A computer program that checks in, notifies and chats with clients.

Nurse: A real, human health practitioner or nurse, caring for a client.

Client: A real, human receiver of care, i.e. the patient that is detoxing.

B Problem

Right now, this is done by nurses and costs a lot of time and availability. To help nurses and clients in this process, technology can offer a hand. This project will realise a platform where a client is able to communicate with a Chat Agent in a human-like fashion. This highly adaptable Chat Agent will guide the user through the process by taking tests, establishing a routine and answering questions the client has. Nurses will also have a dashboard where they can monitor the client's progress and manage their treatment plan. This should relieve the nurse of some of their duties and make the whole process more pleasant and effective for both parties.

C Goals

The goals this project aims to achieve:

- Develop a website for clients to use where they can chat with a Chat Agent.

- Develop a Chat Agent that can be trusted to communicate with sensitive clients.
- Develop a website for nurses where they can monitor clients.
- Connect the client and nurse side, where the data is stored in a database.

D Plan

Detox assistant and monitoring tool - should know as much as possible about the detox process, give tools for medical professionals to specifically guide the Chat Agent, and be helpfully interactive in common cases, with the emphasis on logical correctness. In summary, a way to continue a detox routine supervised, but without the need for direct physical visits.

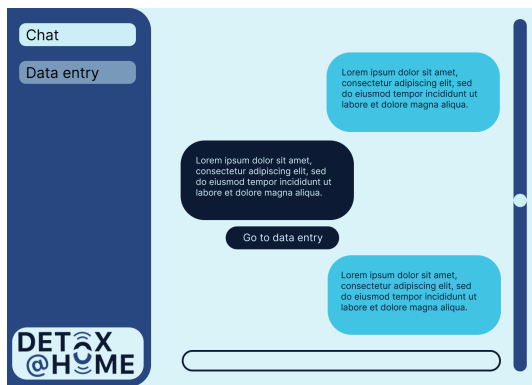
E Functional Requirements

- The system must have a Chat Agent that closely mimics human speech.
- The Chat Agent must be able to check in on clients periodically.
- The Chat Agent must be able to ask questions specified by the nurse.
- The Chat Agent must be able to respond appropriately to the client regarding their behaviour. A response is appropriate when it fits the context but also maintains the client's well-being.
- The system must store answers and data derived from chat interactions.
- The system must store all relevant medical data from the client.
- The system must store questionnaires for the agent to turn into chat topics.
- The system must flag anomalous medical data.
- The system should alarm practitioners based on flags.
- The system must be housed as a Web application.
- The system must have a Dutch interface.
- Clients must be able to communicate with the Chat Agent.
- Clients must be able to manually input data.
- Clients could be able to enter data through chat messages.
- Clients must only be able to access the application via code generated by the nurse.
- Clients must be able to view the chat on mobile devices and on a computer.
- Clients must be able to understand everything related to the app with a basic understanding of the Dutch language.
- Clients must be able to use the app when they have a basic understanding of mobile devices.

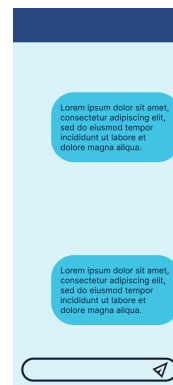
- Nurses must be able to view webpages on computers.*
- Nurses could view webpages on phones and tablets.
- Nurses must be able to use an account with a username and password.
- Nurses must be able to view all their clients in one overview.
- Nurses must be able to monitor each client individually. Containing information about their progress, health and plan.*
- Nurses must be able to configure detox programs for the client so that these can be implemented by the chatting agent.*
- Nurses must be able to (re)view the chat history of a client.

Appendix B

Chat Mock-Up

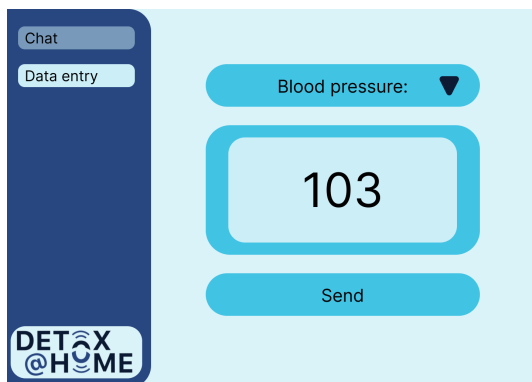


(a) On desktop/wide screen

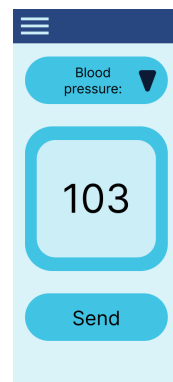


(b) On mobile/narrow screen

Figure B.1: Initial chat interface design



(a) On desktop/wide screen



(b) On mobile/narrow screen

Figure B.2: Initial health data entry interface design

Appendix C

Dashboard Mock-Up

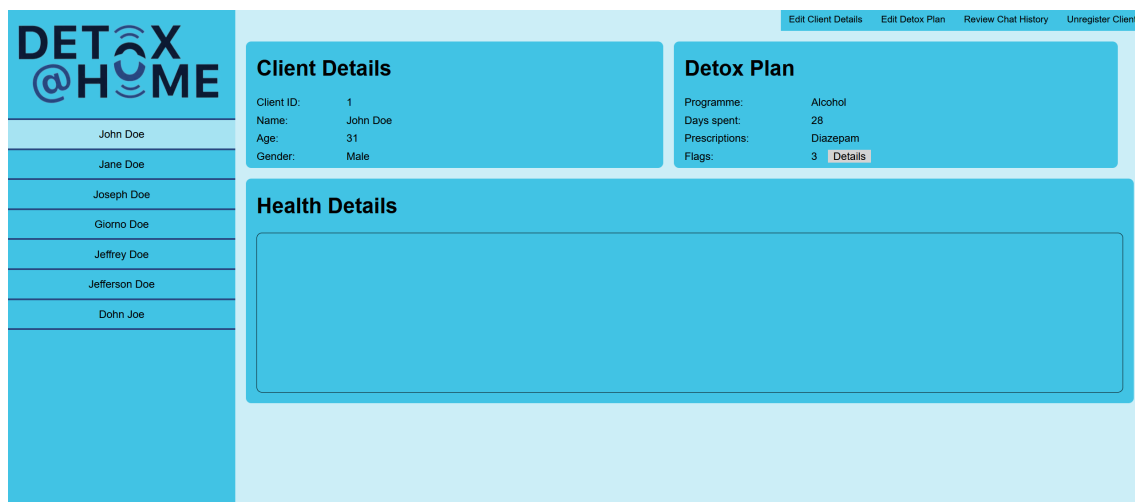


Figure C.1: Initial design of the Nurse Dashboard

Appendix D

Initial Database Schema

NURSES				PATIENTS			
	column name	type	constraint		column name	type	constraint
	username	charvar	PK		nurse_ID	charvar	FK (references NURSES.username)
	password	charvar			patient_ID	charvar	PK
	first_name	charvar			protocol	json	
	last_name	charvar			chat_history	json	
					age	integer	
					first_name	charvar	
					last_name	charvar	
HEALTH_DATA							
	column name	type	constraint				
	patient_ID	charvar	FK (references PATIENTS.patient_ID)				
	blood_pressure	charvar					

Figure D.1: Original schema of the SQL database

Appendix E

Additional Dashboard Screenshots

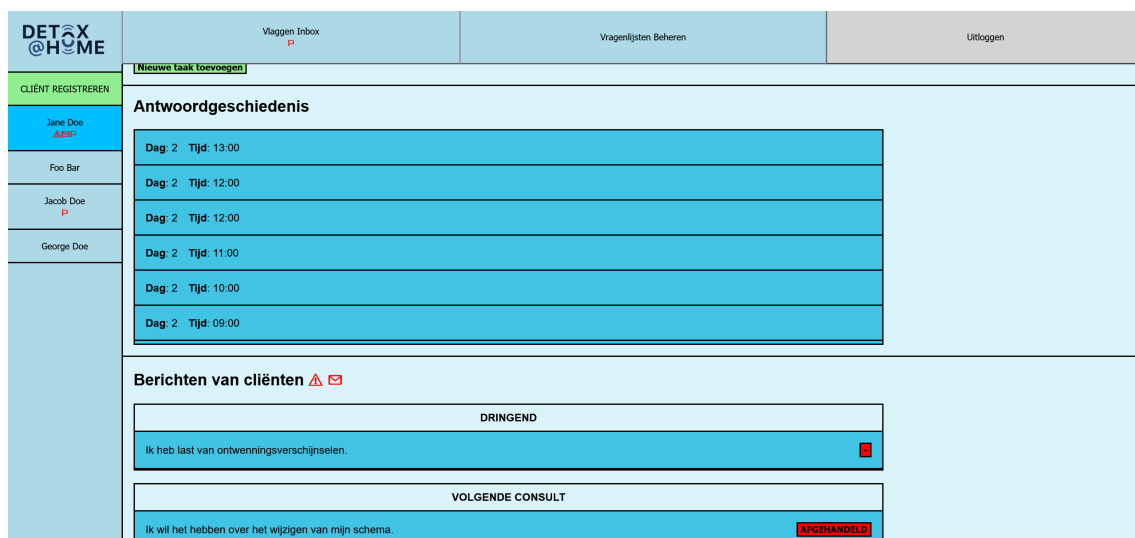


Figure E.1: Answer History and Contact Messages panes

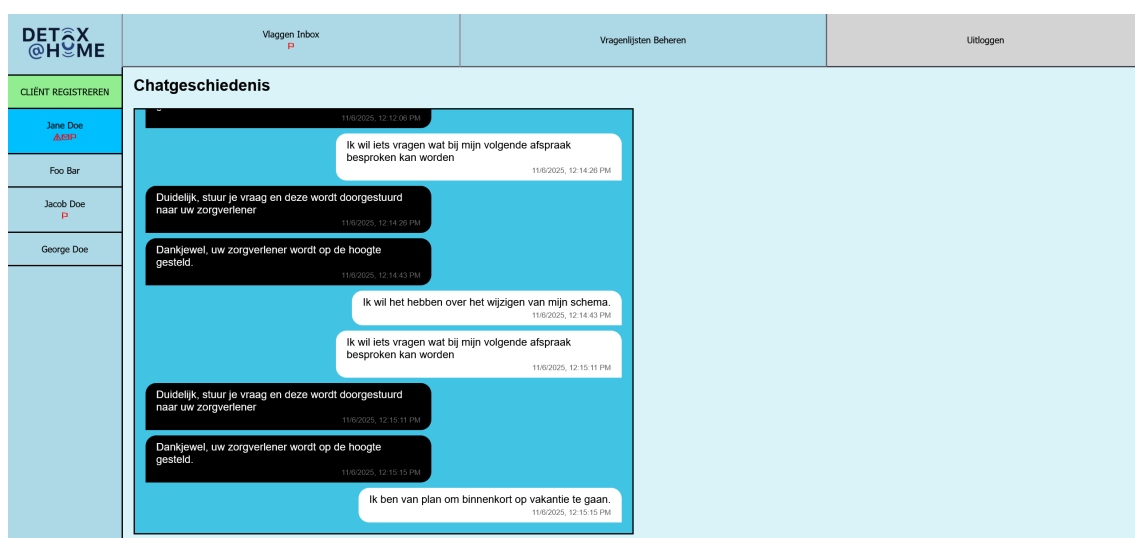


Figure E.2: Chat History pane



Figure E.3: Questionnaire Manager

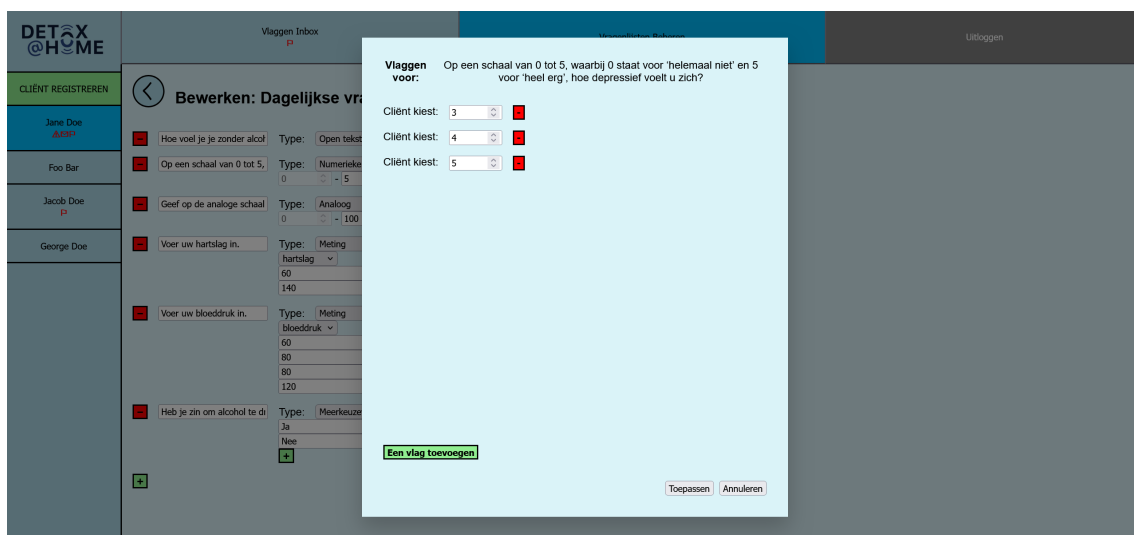


Figure E.4: Flag Configuration Modal


	Vlaggen Inbox PS		Vragenlijsten Beheren	Uitloggen
	<div> <div>CLIENT REGISTREREN</div> <div> <div>Tijd: 2025-11-06 12:19:02</div> <div> <div>Betreffende:</div> <div>Gebruiker heeft niet gereageerd</div> </div> <div> <div>Waarde:</div> <div></div> </div> <div>AFRONDEN</div> </div> </div>			
	Jane Doe AB901			
	Foo Bar			
	Jacob Doe PS			
	George Doe			

Figure E.5: Flag inbox

Appendix F

Final Database Schema

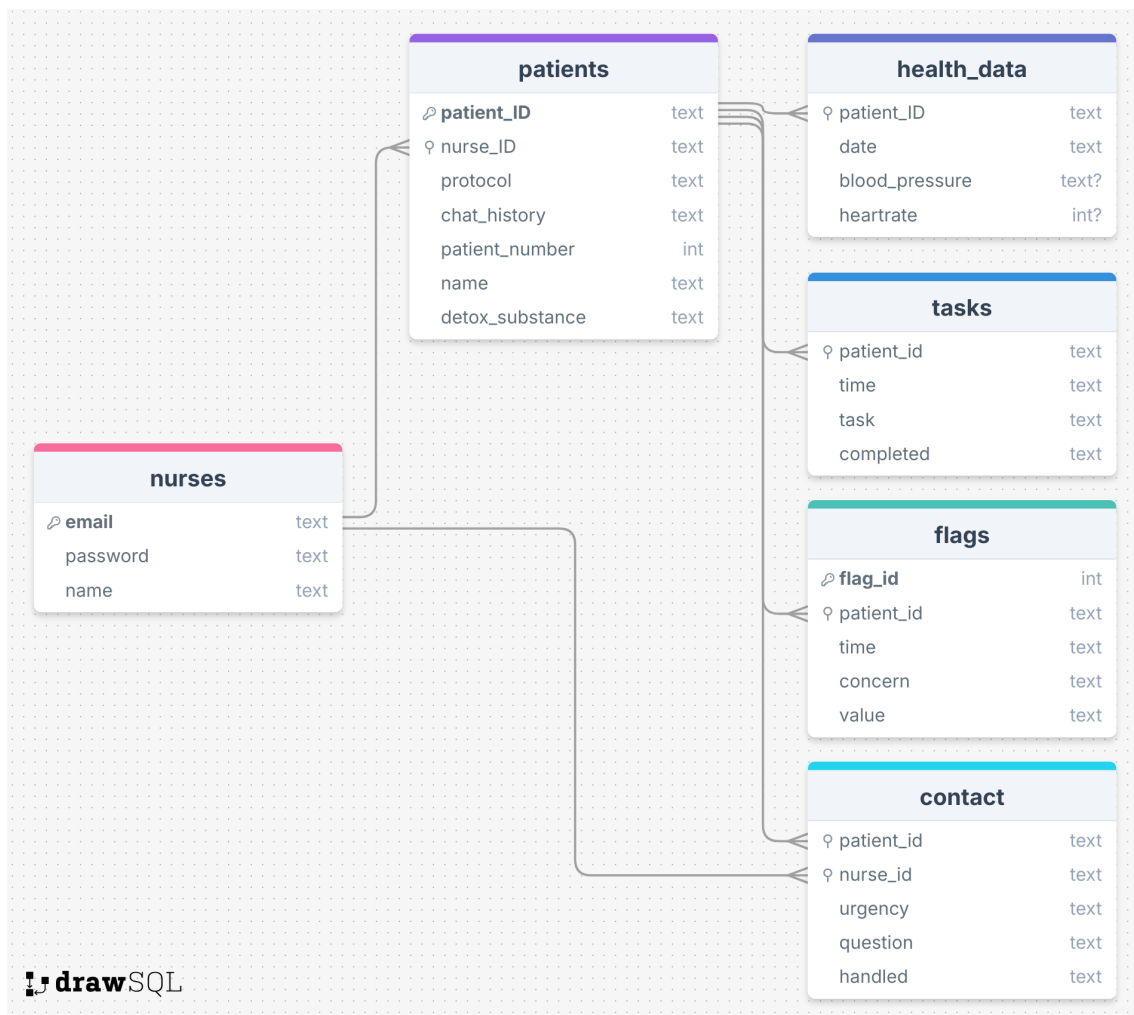


Figure F.1: Final schema of the implemented SQL database