

The Madboks Project

- A booking and event management platform for a non-profit organisation fighting against food waste

Bence Szabo, Lojain Ajek,
Louise Foldøy Steffens, Sara Selman & Sofia Gran

Software, Group 1
2024-December
Semester Project, P7





Software
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY STUDENT REPORT

Title:

The Madboks Project - *A booking and event management platform for a non-profit organisation fighting against food waste*

Theme:

Internet and sustainability

Project Period:

The Fall semester of 2024

Project Group:

Group 1

Participant(s):

Bence Szabo
Lojain Ajek
Louise Foldøy Steffens
Sara Selman
Sofia Gran

Supervisor(s):

Tung Kieu

Copies: 1**Page Numbers:** 130**Date of Completion:**

December 19, 2024

Abstract:

This project documents a collaboration with the non-profit organisation Madboks, whose mission centers on sustainability through the reduction of retailer-level food waste. The project focuses on developing a unified web application to enhance and streamline Madboks' daily operations to support this mission. This paper explores the strategies employed to improve operational efficiency, the methodologies used to map out and understand the current system, the technical and architectural decisions guiding the application's development, scalability and deployment pipeline, and the user experience (UX) design principles that informed its creation. Additionally, it evaluates the final product's quality through feedback from both administrators and users, assessing its effectiveness in advancing Madboks' mission and goals.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Preface

This report documents the work on the 1st-semester project at the Software Masters of Science Program at Aalborg University in Copenhagen. The theme of the semester project is 'Internet' and the purpose of the project module is to contribute to the students gaining knowledge about and being able to develop an Internet application or service. In addition, the semester project should, in some way, involve one or more of the United Nation's sustainable development goals (SDGs). The project presented in this report worked on creating a website for the non-profit volunteer organisation Madboks, which is dedicated to reducing food waste through efficient food redistribution. This contributes to the SDG nr. 12, "*Ensure sustainable consumption and production patterns*".

Gratitude is extended to the supervisor, Tung Kieu, for providing invaluable professional knowledge, guidance, and constructive feedback throughout the development and writing process. Also, a heartfelt thank you to Roxana Zlate, the founder of Madboks and the product owner of this project. The authors of this rapport are grateful for collaborating with an organisation that makes a real-life impact and actively contributes to a more sustainable community.

Table of Contents

1	Introduction and motivation	5
2	State of the Art	7
2.1	Too Good To Go	7
2.2	Facebook Events	8
2.3	Summary	10
3	Analysis	11
3.1	System Definition	11
3.2	Rich picture	12
3.3	FACTOR criterion	13
3.4	Problem Domain Analysis	14
3.4.1	Classes & Events	14
3.4.2	Structure	17
3.4.3	Behaviour	20
3.5	Application domain analysis	23
3.5.1	Actors	23
3.5.2	Usage	24
3.6	Summary	24
3.7	Problem statement	25
4	Design	26
4.1	Requirements	26
4.1.1	MoSCoW	27
4.2	Database diagram	29
4.3	System Architecture	30
4.3.1	N-Layers and N-Tiers Architecture	30
4.4	Navigation	31
4.4.1	Guest Navigation	31
4.4.2	Customer Navigation	32
4.4.3	Admin Navigation	32
4.5	UI design	33

4.5.1	UI mock-up	33
4.5.2	Summary	34
5	Implementation	35
5.1	Technology Stack	35
5.1.1	Development environment	35
5.1.2	Application Technology	36
5.1.2.1	Database	37
5.1.3	Ensuring security with Cloudflare Turnstile Captcha	37
5.1.4	DevOps and CI/CD	37
5.1.4.1	Development lifecycle	37
5.1.4.2	Frontend pipelines	38
5.1.4.3	Backend pipelines	39
5.1.5	Hosting on server	41
5.1.6	Docker	41
5.2	Application of Agile Principles	41
5.3	Sprint 1	43
5.3.1	Sprint Planning	43
5.3.2	Frontend items	43
5.3.3	Backend items	44
5.3.3.1	S1B1, S1B2, S1B3: CRUD for events, locations and reservation	44
5.3.4	Sprint Review	46
5.4	Sprint 2	46
5.4.1	Sprint Planning	46
5.4.2	Frontend items	47
5.4.2.1	S2F1: Homepage (second iteration)	47
5.4.2.2	S2F3: Email service frontend and corresponding forms	48
5.4.2.3	S2F5: Login and signup UI connected with Supabase auth	49
5.4.2.4	S2F6: Connect upcoming events, your events and reservation with backend/database	49
5.4.3	Backend items	51
5.4.4	S2B2: Email service backend and email templates	51
5.4.5	Sprint Review	52
5.5	Sprint 3	53
5.5.1	Sprint Planning	53
5.5.2	Frontend items	53
5.5.3	S3F1: Mobile Compatibility	53
5.5.4	S3F2: Cloudflare Turnstile	53
5.5.5	S3F4: Admin dashboard makeover, timeslot fix, and edit active and upcoming events	54
5.5.6	S3F5: 'Event' page updates - UI fix and pop-up to edit/cancel booked events.	56
5.5.7	S3F6: Reservation with timeslots	57
5.5.8	S3F7: Location page for admin	58

5.5.9	Backend items	59
5.5.10	S3B1: Cloudflare Turnstile Captcha	59
5.5.11	S3B3: Setup of pre-production server	59
5.5.12	S3B4: Timeslots/Locations/Events/Reservations makeover	60
5.5.13	S3B5: Email updates (send multiple emails at once, add personal information to formatting using Handlebars)	64
5.5.14	S3B6: Docker Setup	66
5.5.15	Sprint Review	66
5.5.15.1	User acceptance test	66
5.6	Sprint 4	68
5.6.1	Sprint Planning	68
5.6.2	Frontend items	68
5.6.3	S4F3: More mobile compatibility	68
5.6.4	S4F4: Email updates	70
5.6.5	Sprint Review	70
6	Quality Assurance	71
6.1	Unit tests	71
6.2	User Tests	71
6.2.0.1	Product owner and admin tests	72
6.2.0.2	Volunteer/customer tests	73
7	Overview of the final product	75
8	Discussion	81
8.1	Process	81
8.1.1	Reflections on the development process	81
8.2	The product owner's reflections on the development process	82
8.3	How QA could have been improved	82
8.3.1	Integration testing	83
8.3.2	Load tests	83
8.3.3	Unit tests on the frontend side	83
8.3.4	Static code analysis (CodeScene)	84
8.4	Future work	84
8.4.1	Security	85
8.4.2	Expanded User Testing	86
8.5	Scalability	87
8.6	Sustainability	87
9	Conclusion	89
A	Screenshots	91
B	Pipelines	93
B.1	CI pipeline web	93

B.2	CI/CD pipeline web -preprod	95
B.3	CI pipeline backend	98
B.4	CI/CD pipeline backend	99
C	Transcription	102
C.1	First Meeting	102
D	Analysis	105
E	Navigation	107
F	Product owner final evaluation	109
G	User Stories	110
H	Sprints	111
H.0.0.1	S1F1: Homepage (first iteration)	111
H.0.0.2	S1F2: Navigation bar (hardcoded first iteration)	112
H.0.0.3	S1F3: Upcoming events component (mock data)	112
H.0.0.4	S1F4: Your events component (mock data)	113
H.0.0.5	S1F5 Location creation	114
H.0.0.6	S1F6: Event creation (first iteration)	114
H.0.0.7	S1F7: Reservation page	114
H.0.0.8	S2F2: Volunteer page	116
H.0.0.9	S2F7: Event popup, showing information and description of the event	117
H.0.0.10	S2F4: Admin dashboard - display events/locations and edit location functionality	117
H.0.1	S2F8: Connect navigation bar with auth	117
H.0.2	S2B1: Setting up the server and hosting + pipelines	117
H.0.3	S2B2: Email service backend	118
H.0.4	S3F3: About us page	118
H.0.5	S3F4	119
H.0.6	S3B4	120
H.0.6.1	S3B4-1	120
H.0.6.2	S3B4-2	121
H.0.6.3	S3B4-3	121
H.0.6.4	S3B4-4	122
H.0.6.5	S3B4-5	124
H.0.6.6	S3B4-6	125
H.0.7	S4F1: User test fixes and small text and layout updates	126
H.0.8	S4F2: Configuration fix to not expose source code in the browser	126
I	Design Criteria for the Madboks Platform	127
Bibliography		129

Chapter 1

Introduction and motivation

Food waste is a pressing global issue with significant environmental, economic, and social consequences. Denmark, despite being a leader in sustainability initiatives, discards approximately 700,000 tonnes of food annually, with a substantial portion still fit for consumption [13]. This issue extends beyond resource inefficiency, directly impacting food insecurity — a pressing challenge for vulnerable populations that could be alleviated through improved food redistribution systems.

A staggering 36% of total waste in the Capital Region of Denmark is residual waste, half of which is food waste. Residual waste refers to waste materials that remain after all reusable, recyclable, or compostable components have been separated or processed. This amounts to 226,000 tonnes of wasted food annually, with 60% being preventable at the retail and household levels [5]. Such waste not only represents a loss of economic and functional value but also exacerbates food insecurity. Schneider highlights in their paper "*The evolution of food donation with respect to waste prevention*", the potential of food donations as a form of "*urban mining*," where edible food is redirected for human consumption instead of being wasted. However, achieving this requires overcoming logistical, social, and regulatory barriers with region-specific solutions [16].

Schneider further emphasizes that food donation can serve as a crucial strategy for waste prevention, highlighting the need for effective systems to facilitate this process.

"The donation of food which is still edible can be seen as a specific application of urban mining as food is recovered for its original purpose - human intake" [16]

Food waste impacts various stakeholders, including food producers, retailers, and consumers. It also disproportionately affects low-income families and individuals facing food insecurity. Effective food donation systems can bridge this gap, but current efforts often lack scalability and operational efficiency, leaving much of the potential untapped [17].

Madboks aims to address this issue by connecting surplus food from retailers to those in need. As a non-profit organisation without a dedicated tech team, Madboks only operates with the help of free and publicly available software solutions. This project focuses on supporting Madboks by recognising their digital shortcomings and providing a suitable solution to amplify their impact, and contribute to global sustainability efforts, particularly the United Nations' Sustainable Development Goals (SDGs):

- SDG 2: Zero Hunger
- SDG 11: Sustainable Cities and Communities
- SDG 12.3: Global Food Loss and Waste

Madboks already has an engaged community, with 10,000 followers on Facebook and 3,000 on Instagram, alongside a peak user interest when food donation forms are shared. Madboks have a solid customer base with several hundreds of donors and customers attending their events every week. Given the sheer size of Madboks' operations, the importance of scalability both in terms of the number of users and features is worth emphasising C.1.

This project aims to enhance Madboks' food donation efficiency by strengthening their digital presence and aligning their operations with Copenhagen's vision for a circular economy [5].

Disclaimer: The authors utilised ChatGPT, Perplexity AI and GitHub Copilot to assist in the writing of the report and accompanying source code.

Chapter 2

State of the Art

This chapter examines state-of-the-art solutions available on the market, aiming to uncover design principles and ideas that can inspire improvements to Madboks' existing solution. By highlighting key strengths and best practices, it lays the groundwork for enhancing the Madboks booking system from a design perspective.

The research focuses on solutions that address food waste reduction and other state-of-the-art systems with features relevant to event creation and display.

2.1 Too Good To Go

One of the most prominent and widely used solutions in Denmark and internationally for redistributing otherwise discarded food is the Too Good To Go mobile app. The app allows users to purchase surplus food at reduced prices from local restaurants, cafés, supermarkets, and bakeries.

This platform provides valuable insights into how technology can support organisations focused on reducing food waste. Too Good To Go's primary goal is to prevent surplus food from being discarded by offering it to users. Key features of the app include a real-time surplus food listing, where businesses post "goodie bags" available for pickup within a specific time window. The app also features a user-friendly interface, enabling users to easily locate and purchase surplus food from nearby locations. The main screens of the app are shown in figure 2.1.

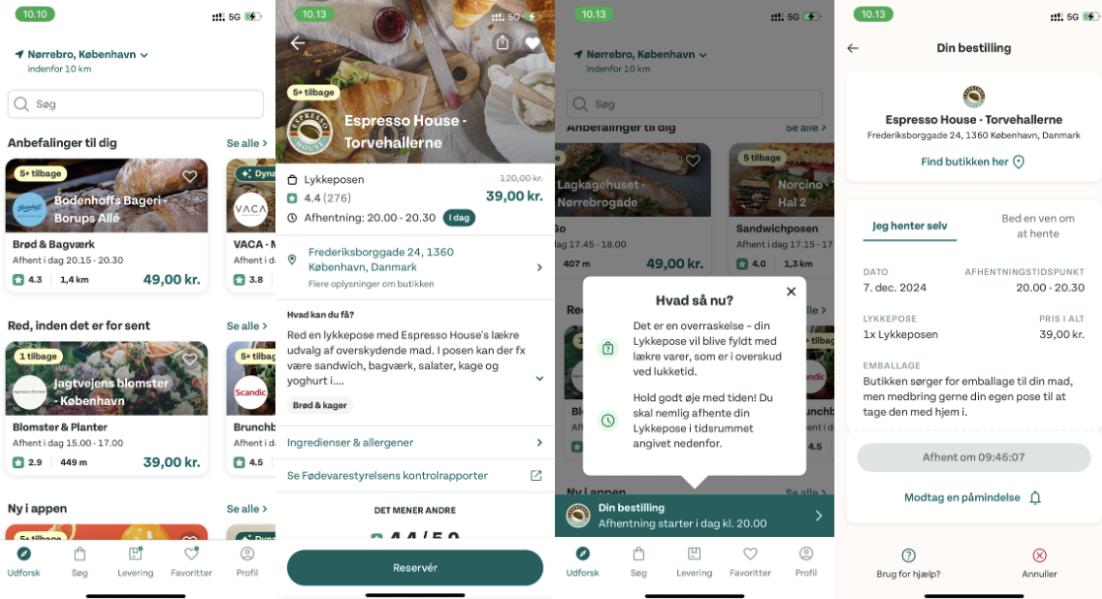


Figure 2.1: Screens from the Too Good To Go booking process: 1. The explore page. 2. An event's details page. 3. A confirmation page. 4. The reservations page.

The app offers valuable inspiration for Madboks in designing a platform for booking food boxes on mobile devices. Some design elements that could be incorporated into the new Madboks platform includes the following:

- Explore page: Displays events as cards with essential information and swipe functionality for more options (see Screen 1 in Figure 2.1).
- Reservation page: Pops up upon selecting an event, showing detailed information and a booking option (see Screen 2 in Figure 2.1).
- User reservations: Allows users to view their booked events and cancel them if necessary (see Screen 4 in Figure 2.1).

2.2 Facebook Events

Facebook is the world's leading social media platform and a primary communication channel for countless organisations and initiatives. With features like Marketplace, Events, Dating, and Groups, it offers diverse functionalities. Given that Madboks aims to deliver its highly event-driven services to both desktop and mobile users, Facebook serves as a valuable source of design inspiration.

Madboks' existing solution partially relies on Facebook for its operations. Since users are

already accustomed to Facebook's interface, examining how its relevant components are designed provides crucial insights.

Adopting familiar design elements and components for similar purposes helps reduce the learning curve and makes the platform feel more intuitive for users.

The Facebook Events page (see Figure 2.2) provides an example of how to display both attended and upcoming events. For Madboks, a similar approach could be adopted: showing booked events vertically with detailed information and options, while upcoming events could be displayed horizontally with less information.

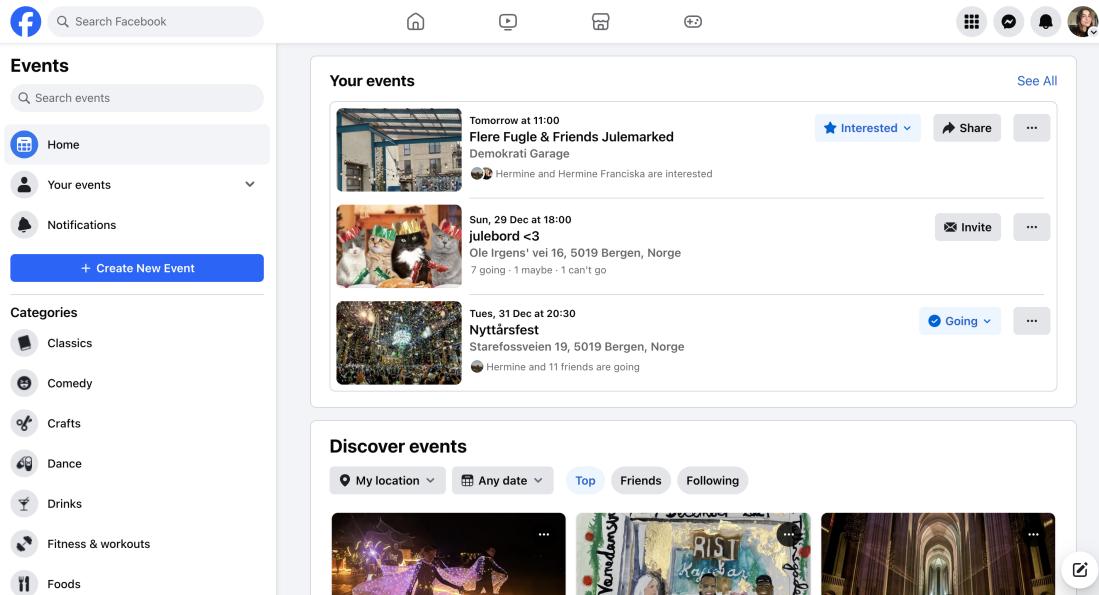


Figure 2.2: Facebook's Events page

Facebook Marketplace provides a useful example of how this could be implemented. The listing creation page (see Figure 2.3) allows users to input listing details on the left while previewing the final output on the right. This approach could simplify event creation for Madboks administrators, offering ease of use and an intuitive layout.

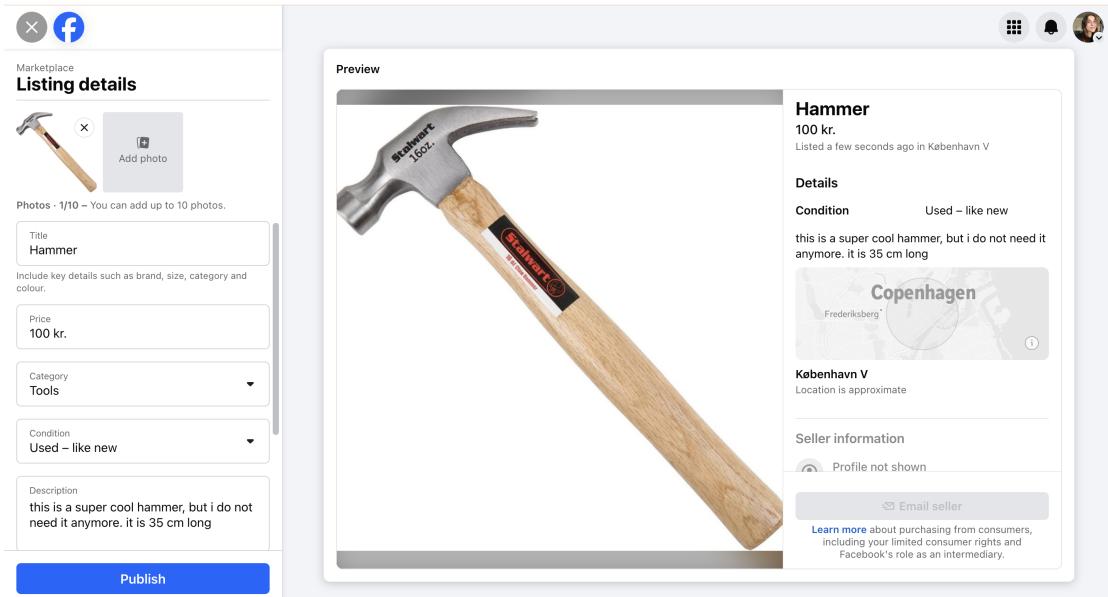


Figure 2.3: Facebook's create listing page

2.3 Summary

The hands-on examination of Too Good To Go and Facebook uncovered specific design and feature ideas that could enhance the new Madboks platform. From Figure 2.1, the Too Good To Go app offers inspiration for mobile designs, such as the explore page (Screen 1), event reservation page (Screen 2), and reservation management page (Screen 4). These features emphasise clean design and easy navigation.

Facebook, with its focus on mobile and desktop usability, serves as a valuable reference for designing the Madboks platform for both types of devices. The Events page (Figure 2.2) provides insights into effectively displaying booked and discoverable events, while the create listing page (Figure 2.3) offers a potential template for event creation by administrators.

While these examples provide inspiration, Madboks' unique domain-specific challenges must be addressed to ensure the platform meets its needs. Ongoing collaboration with the product owner and an analysis of the current system's shortcomings will ensure a solution that adheres to proven design principles while catering specifically to Madboks' requirements.

Chapter 3

Analysis

Before design and development can begin, the fundamental goals of the project must be well understood. Requirements for the new system, should be established in agreement with the product owner and represent the needs of Madboks and its customers.

This chapter begins by mapping out and establishing a clear understanding of the current Madboks system, identifying its strengths and weaknesses through system definition, a rich picture, and the FACTOR criterion. Based on these insights, new functionalities are introduced and evaluated through problem domain and application domain analysis. This process narrows the focus to address the core needs for the future system, while considering the limitations of the current system. Additionally, this approach will inform decisions related to system architecture, data models, and implementation technologies. The analysis of the problem and current system is guided by principles outlined in the Object-Oriented Analysis and Design book, OOAD, [9].

All information for this analysis was gathered from the product owner during the initial meeting, where they outlined the current system's functionality and their vision for the future system. The full transcription of this meeting can be read in Appendix C.1.

3.1 System Definition

The system definition is the process of outlining and clarifying the scope, goals, components, and interactions of Madboks. This is important for obtaining an understanding of Madboks' purpose, thus ensuring a clear vision of the system's objectives and functionality.

Madboks is a Copenhagen-based non-profit organisation dedicated to preventing food waste by collecting unsold items from local food retailers and distributing them to customers in exchange for donations. The system includes all activities related to food collection, sorting,

logistics, customer reservations, and stakeholder communication. All processes are handled manually by volunteers using basic tools like Excel for logistics, Google Forms for managing reservations, and Facebook/Instagram for real-time communication with customers about availability, cancellations, and updates. The system promotes sustainability and involves stakeholders such as admins, volunteers, customers, and food retailers.

3.2 Rich picture

A rich picture is a visual representation and mapping of the system definition, that gives an overview of the people, objects, processes, structures, and challenges in the system's problem and application domains. It is used to illustrate the interactions, relationships, and elements and helps us gain a shared understanding of the situation [9].

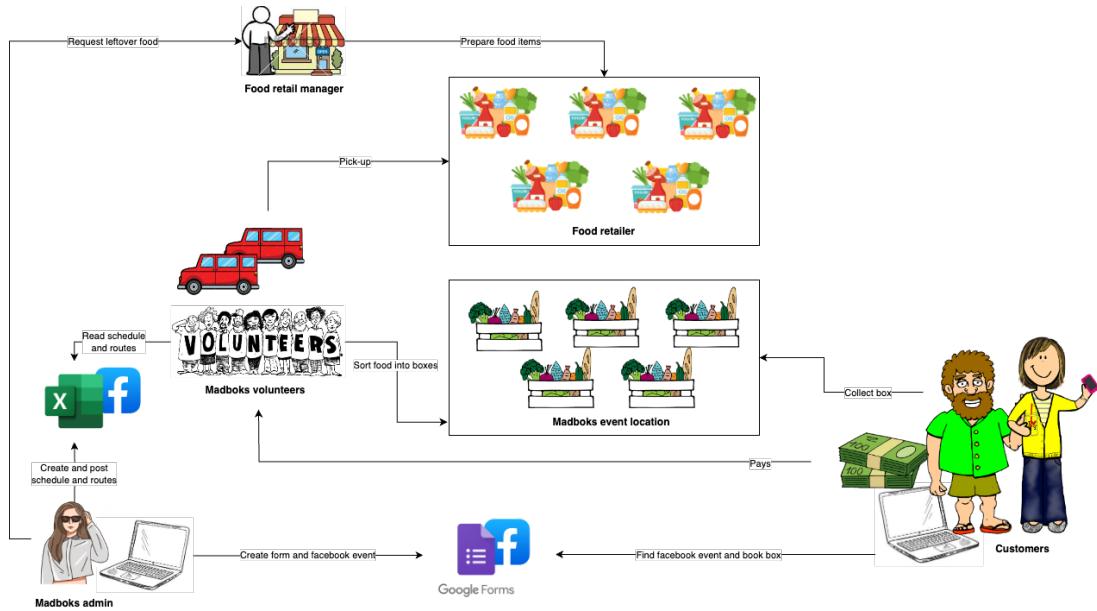


Figure 3.1: Rich picture describing Madboks's system

The process starts with one of the Madboks admins who contacts the local food retailer collaborators, and organise pick-up of their left-over food items. The admin then creates routes and schedules, ensuring that they have enough volunteers and cars to pick-up the food items at the retailers at the agreed time. The day before the event, one of the admins publishes a Google Form booking sheet, and a Facebook event that links to the form. The event is then promoted on their Facebook and Instagram accounts.

On the day of the event, volunteers drive to various retailers to collect the food. If they see that there are more boxes than assumed, or people are contacting them to cancel their reservation,

they manually add boxes to the Google form, and make posts on their social media to inform customers. Meanwhile, the food items are then sorted into boxes, and the customers who have booked through the Google Form can pick up their box and pay their chosen donation. This cycle persists for every box-collection event.

3.3 FACTOR criterion

In this section, the FACTOR criterion is introduced to define and understand the system architecture in a concise form, building on insights gathered from the system definition and rich picture. The FACTOR acronym represents six essential elements that must be considered when designing a digital solution: Functionality, Application Domain, Conditions, Technology, Objects, and Responsibility [9].

Functionality

Routes for collecting food are organised, and the number of boxes available for purchase is manually adjusted and communicated by volunteers. Customers reserve boxes and boxes are distributed at multiple locations at selected time slots. Social media is used to manage cancellations, adjustments in box availability, and communication with customers.

Application Domain

Preventing food waste by collecting unsold items from local food retailers, sorting them in boxes, and selling them at a symbolic price. This is done by administering the collection of food, sales of food boxes, managing reservations, and communicating availability, collection times, and locations.

Conditions

As a non-profit relying on volunteers with varying levels of technical expertise. Decisions are driven by the goal of reducing food waste and supporting sustainability.

Technology

Madboks relies on publicly available solutions, namely Excel, Google Forms, and Facebook/Instagram. Excel is used to calculate routes and box collection logistics, while Google Forms is utilised to manage customer reservations of boxes. Lastly, Facebook handles real-time communication of updates to customers and changes within the organisation.

Objects

The stakeholders are admins, volunteers, customers, and food retailers. The other objects are spreadsheets, forms, food boxes, events, and social media posts.

Responsibility

The system works as an administrative tool and communication medium supporting the daily operations of Madboks.

From the FACTOR criterion, it can quickly be determined that Madboks faces inefficiencies

due to manual processes. When a customer orders a box, they have to choose a pickup time. One limitation that Google Forms has, is that one cannot set limitations on how many users can book for each timeslot, causing overcrowding and logistical strain. In the meeting with the product owner and founder of Madboks, Roxana Zlate, they pointed out that "*Google add-ons aren't the best, so we can't easily automate the process of limiting bookings per time slot*", (Zlate, 2024 C.1).

Volunteers must manually update box availability, a time-consuming and error-prone task. Tools like Google Forms and Facebook lack integration and automation, leading to fragmented workflows and limiting scalability. These inefficiencies should be considered throughout the analysis to determine how to address them effectively.

3.4 Problem Domain Analysis

This section introduces the problem domain analysis, focusing on modelling the key aspects of the system. This step in system development defines the scope and its interaction with the real world, focusing on the specific part of the environment the system will manage, monitor, or control. The purpose of this analysis, is to identify and model the classes, events, structures, and behaviours relevant to the system's context [9].

During the initial meeting with the product owner Roxana Zlate C.1, the scope of the future system was drafted. While third-party tools like Google Forms and Facebook work well individually, they lack a unified platform to streamline all processes. They mentioned in the meeting that "*Google add-ons aren't the best*", and "*people can't easily edit and adjust their booking*", (Zlate, 2024 C.1). They also pointed out that managing cancellations through Facebook is inefficient and time-consuming C.1. The future system aims to consolidate these functionalities into a single, cohesive platform, providing greater flexibility and customisation.

3.4.1 Classes & Events

Classes and events are foundational concepts in system modelling, used to represent the entities and actions within a problem domain. Identifying them involves compiling a list of potential candidates and systematically evaluating their relevance to the system using criteria outlined in the book [9].

For Madboks' system, all objects involved in the organisation's workflow should be considered to model their real-world scenarios. Classes were analysed and evaluated based on whether unique objects can be identified from them if they contain distinct information, whether they represent multiple objects, and if they participate in a suitable and manageable number of events [9]. All identified classes for Madboks can be seen in table D.1 in the Appendix.

Selection process of candidate classes

The selection of candidate classes was based on their alignment with the future system's scope

and their relevance to addressing the core functionalities, including reservations and event management. Classes that did not contribute directly to creating an understanding of the current system and the functionalities of the future system were excluded.

The Volunteer class represents distinct individuals with specific attributes like availability and tasks, while the Customer class uniquely identifies end-users managing reservations. The Food Box class merges Organisation Box and Customer Box, simplifying functionality while maintaining unique attributes like content and reservation status. The Transaction class tracks payments with clear details such as amount, and the Location class identifies venues with attributes like address and capacity. The Admin class represents system administrators with defined responsibilities. Lastly, the Event and Reservation classes uniquely identify core system workflows.

The Social Media Post, Social Media Account, and Google Forms classes were excluded as the future system focuses on integrating communication and reservations into a single platform, eliminating reliance on third-party tools like Facebook for updates and Google Forms for bookings. Rental Car, Food Retailer, and Excel Sheets were excluded as transport and external logistics, including food transportation tracking, fall outside the system's scope. Food Item was omitted to avoid complexity, being better represented as attributes of Food Box. Device was deemed unnecessary as it is implicit in user interactions.

In conclusion, the selected classes can be seen in table 3.1

Volunteer	Customer	Location
Food Box	Reservation	Transaction
Admin	Event	

Table 3.1: Table of the selected classes

Candidate Events

To identify relevant events, the criteria outlined in the OOAD methodology were applied, ensuring events are instantaneous, atomic, and clearly identifiable at the time they occur [9]. In Appendix D, table D.2 outlines the candidate events, which represent actions that impact the selected classes.

Selection process for candidate events

The selection of candidate events focused on identifying distinct and meaningful actions that directly impacted the selected classes. The selected events must align closely with the system's scope, focusing on reservations and event management.

`RESERVATION CREATED`, `RESERVATION MODIFIED`, and `RESERVATION CANCELLED` are included because they represent key stages in the lifecycle of customer bookings, allowing for seamless management of reservations. Similarly, `EVENT CREATED`, `EVENT MODIFIED`, and `EVENT CANCELLED` are integral for administrators to manage and update events, ensuring flexibility and adaptability in event planning.

`BOUGHT` is included to track completed transactions for food boxes, ensuring payment accountability within the reservation workflow. `PREPARED` was selected as it reflects a crucial step in food distribution, documenting the preparation of food boxes for customers. `LOCATION CREATED`, `LOCATION MODIFIED`, and `LOCATION DELETED` were selected to facilitate the accurate management of venues. `SIGNED UP/IN`, `SIGNED OUT`, and `ACCOUNT DELETED` were included to support authentication and account management for customers. This will allow customers to securely manage their reservations and account information.

`SHIPPED` and `DISTRIBUTED` involve external logistics not managed by the system. `PROCESSED` pertains to manual payments, intentionally left out. `ASSIGNED` handles volunteer allocation, which is not central to reservations or event management.

The list of the selected events can be seen in table 3.2.

Reservation Created	Reservation Modified	Reservation Cancelled
Event Created	Event Modified	Event Cancelled
Location Created	Location Modified	Location Deleted
Bought	Prepared	Signed Up/In
Signed Out	Account Deleted	

Table 3.2: Table of selected events.

Event table

The event table highlights the interactions between the selected events and classes, creating the foundation for modelling the system's behaviour later in the analysis.

The selected classes are the primary entities, while the events capture the key interactions and actions affecting these entities. Table 3.3 provides an overview of their relationships, helping to determine their structure and, thus, providing a foundation for the system design.

Events/Classes	Admin	Volunteer	Customer	Food Box	Reservation	Transaction	Location	Event
Reservation Created			X	X	X			
Reservation Modified			X	X	X			
Reservation Cancelled			X	X	X			
Event Created	X						X	X
Event Modified	X						X	X
Event Cancelled	X						X	X
Location Created	X						X	
Location Modified	X						X	
Location Deleted	X						X	
Bought			X	X	X	X		
Prepared		X		X				
Signed Up/In	X	X	X					
Signed Out	X	X	X					
Account Deleted	X	X	X					

Table 3.3: Event Table

3.4.2 Structure

As previously mentioned, the structure of the problem domain is made to build a foundation for the system design. It represents the essential entities, their relationships, multiplicity, and hierarchy within the Madboks system. By the OOAD principles [9], the relationships between classes are modelled as generalisations, associations, and aggregations, representing the structural and functional connections within the system. The system is further divided into logical clusters to improve modularity and clarity.

Generalisation

Generalisation is used to capture shared attributes and behaviours among classes while allowing for specialisation. In the Madboks system, the `User` class serves as a superclass for `Admin`, `Volunteer`, and `Customer`.

The `User` class contains shared attributes such as name, contact information, and login credentials, while specialising in:

- `Admin`: Managing the overall system operations, including creating, modifying, and cancelling `Events`, as well as overseeing `Reservations` and coordinating with `Volunteers`. Admins are also responsible for updating `Locations`.
- `Volunteer`: Assisting with event logistics and is linked to `Event` through an aggregation relationship.
- `Customer`: Responsible for making reservations and completing transactions. Customers interact with `Reservations`, which link them to `Food Boxes` and `Events`.

This structure supports scalability and modularity, allowing for the addition of new user roles. For instance, the `Volunteer` role could be extended to include specialised types such as `Event Coordinator`, who manages logistics during events, or `Food Collector`, who focus on picking up food donations.

Aggregation Structure

Aggregation represents a "has-a" relationship between classes, where one class is composed of other independent classes. This approach ensures that the system can manage these components separately, allowing for flexibility in reusing or modifying classes without affecting the overall structure.

The Event class aggregates both Volunteer and Location, representing its reliance on volunteers for logistical tasks and venues for hosting the events. Each Event can involve multiple Volunteers (1..*) and a single Location (1), while a Volunteer can assist in multiple Events (0..*), and a Location can host multiple Events (0..*).

The Food Box class aggregates Reservation, as each Food Box can either be reserved or remain unreserved. A Reservation is always tied to one specific Food Box (1), while a Food Box may or may not have an associated Reservation (0..1). Similarly, the Customer class aggregates Reservation, as a Customer can make multiple Reservations (1..*), but each Reservation is tied to a single Customer (1).

This aggregation ensures scalability and adaptability, allowing independent updates to, for example, volunteers and locations without disrupting event functionality, supporting a modular design.

Association Structure

Association is used in the Madboks system to represent general relationships between classes, capturing how objects interact with each other without implying ownership.

The Admin class is associated with Event, signifying the role of admins in managing events. An Admin can oversee multiple Events (1..), while each Event is managed by one Admin (1).

The Transaction class is associated with Event, Customer, and Reservation, representing the connections involved in payment handling. A Transaction is linked to a specific Customer (1) who makes the payment and references a Reservation (1) to specify the related booking. Additionally, Transactions can reference an Event (0..1) for financial tracking.

These associations ensure that the interactions between classes are explicitly modelled. They define relationships, such as the role of Admin in managing Events and Transaction handling payments for Reservations and Customers. By avoiding tight coupling, the design preserves the separation of concerns, allowing classes to function independently. This approach supports scalability, enabling new features or relationships to be added without disrupting the existing structure.

Class diagram of Problem Domain

The class diagram in 3.2 provides a structured view of the Madboks system, modelling its entities and their relationships. It is divided into two clusters: the Box Sale Cluster and the

Profile Cluster, ensuring modularity and clarity in capturing the system's functionality.

The Box Sale Cluster represents the operational workflow, including event planning, food box management, reservations, and transactions. Event is central to this cluster, occurring at a specific Location and linked to Food Box through an aggregation relationship. Reservation ties Customers to Events and Food Boxes, while Transaction handles payment processing.

The User Cluster focuses on user management, with User as a superclass for Customer, Volunteer and Admin. Customer interacts with reservations and transactions, while Volunteer assists with Events. Lastly, the Admin is managing the operations for the system.

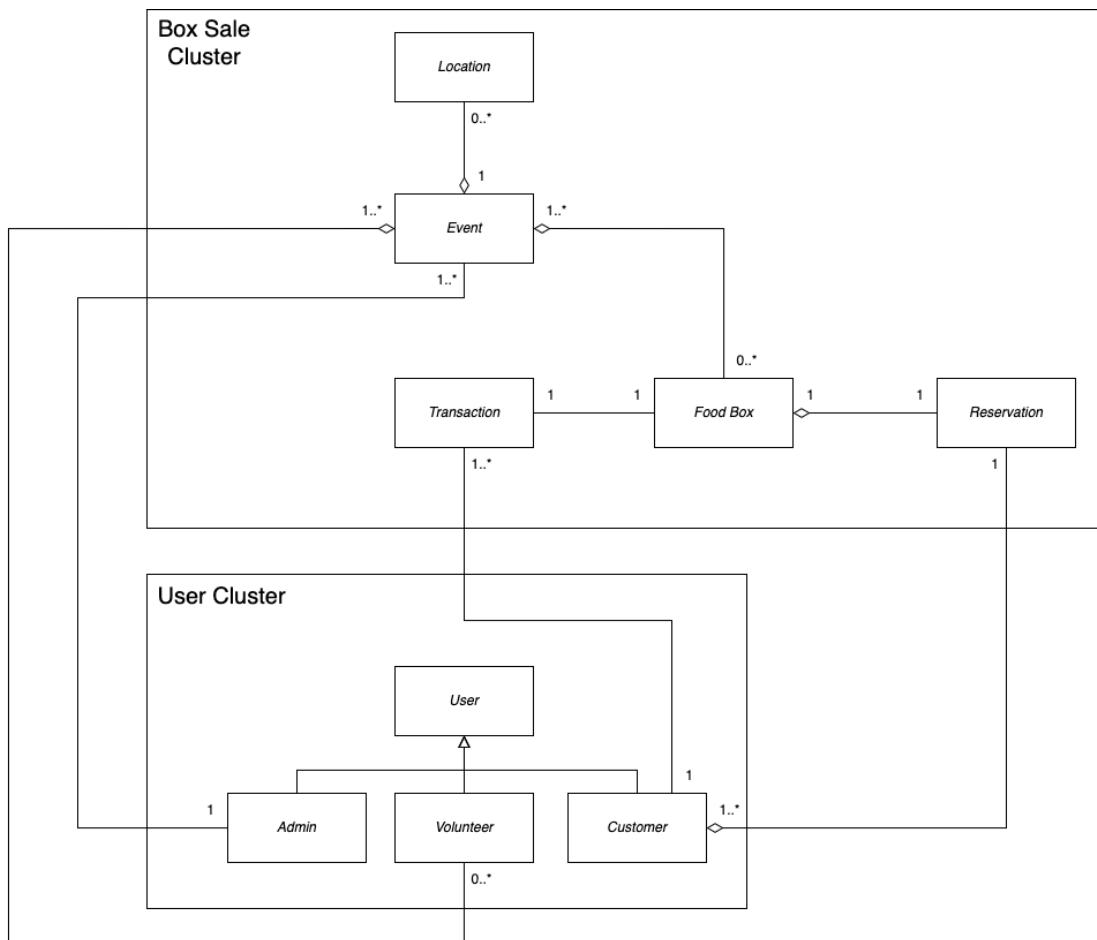


Figure 3.2: Problem domain class diagram

The Transaction and Food Box classes remain in the class diagram to understand the sys-

tem's structure and interactions. However, they will be excluded from the analysis and, consequently, from the system design. While they helped in modelling the system's context and relationships, they do not align with Madboks' scope. The *Transaction* class involves payment processing, which is not managed by Madboks, as the system focuses on food distribution. Similarly, the *Food Box* class represents a physical item that is not directly managed within the system. Instead, the system tracks reservations and events. This shifts the aggregation relationship from *Food Box* to *Event*, as reservations now serve as the connecting entity. Excluding these classes ensures the focus remains on reservation management, event operations, and user interactions, aligning the system design with Madboks' scope.

3.4.3 Behaviour

The purpose of this section is to model the dynamic aspects of the problem domain by identifying events, state changes, and attributes for each class in the class diagram. This analysis clarifies how the classes interact through events, ensuring that their behaviours and relationships are defined.

To achieve this, the event table is used to describe the behaviour patterns for each class, mapping how they respond to specific events. From these behaviour patterns, the relevant attributes are extracted for each class, making sure that the diagram accurately reflects the system's states and transitions.

As seen in the event table and class diagram, reservation has a relation to the *Customer* and the *Event*. The reservation starts in an idle state until a customer creates it and the reservation is active. Here, the reservation can be modified by an unlimited amount until it either gets cancelled by the customer or the event has been concluded, thus marking the reservation fulfilled. From the state diagram in figure 3.3, the reservation gets the first draft of its attributes, also shown in the figure.

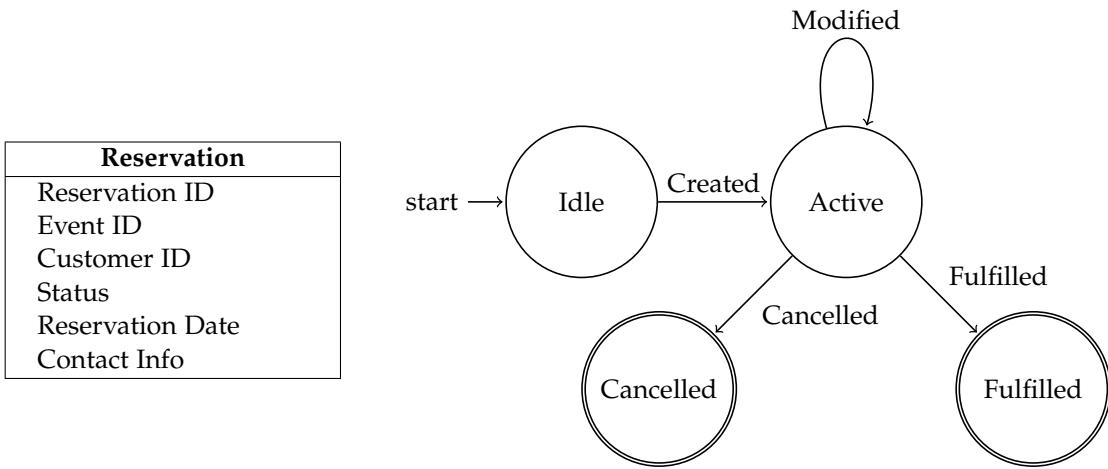


Figure 3.3: State Diagram for Reservation Class

The Event class has a similar state diagram as the Reservation, where here the Admin is making the operations on the event, as shown in figure ???. The Event can be modified in the active state. An Event ends either with being cancelled by the Admin or the event has taken place.

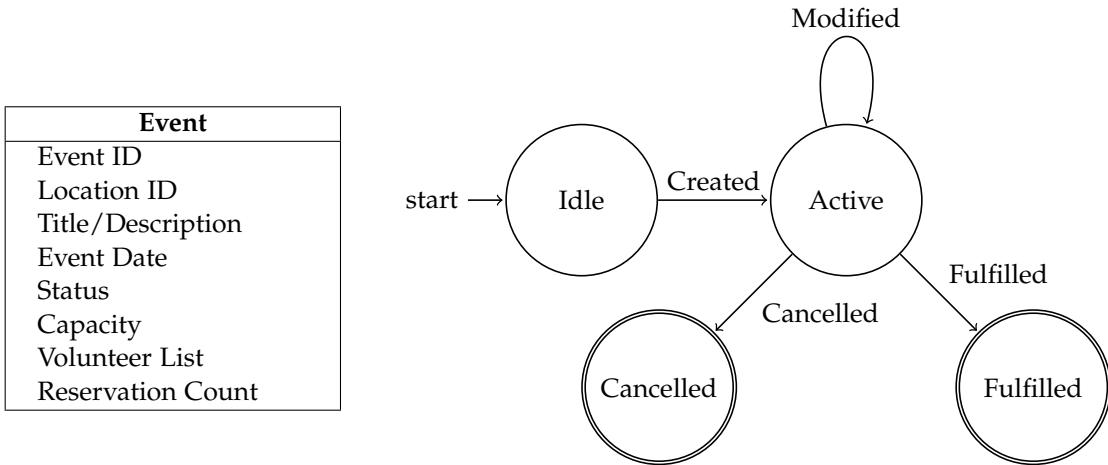


Figure 3.4: State Diagram for Event Class

Customers start in the *inactive* state until they sign up or proceed to the *guest* state. While being active or a guest, they can perform actions such as making reservations. A signed-in Customer account ends when they sign out or delete their account permanently, while a guest becomes inactive to the system when they have no active reservations. This can be triggered by the specific event reservation being cancelled.

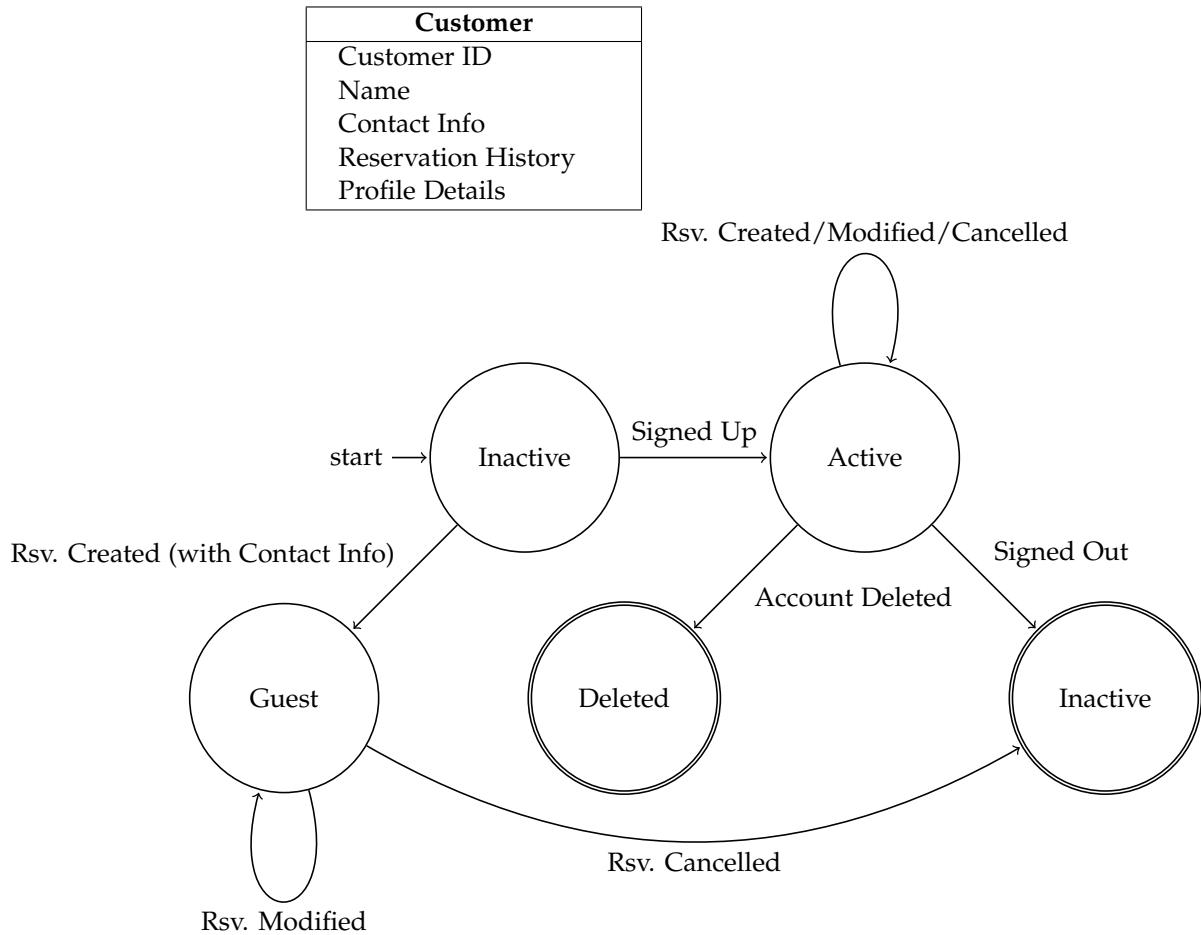


Figure 3.5: State Diagram for Customer Class: Notifications and General Activity

The Location class follows a structure similar to the Event class, where locations are allocated or updated as needed. A Location starts in the *unallocated* state and transitions to *allocated* when assigned to an event. Locations can be updated while *allocated*, reflecting changes in details such as capacity or time. The lifecycle of a Location does not include an explicit end-state unless it is removed from the system.

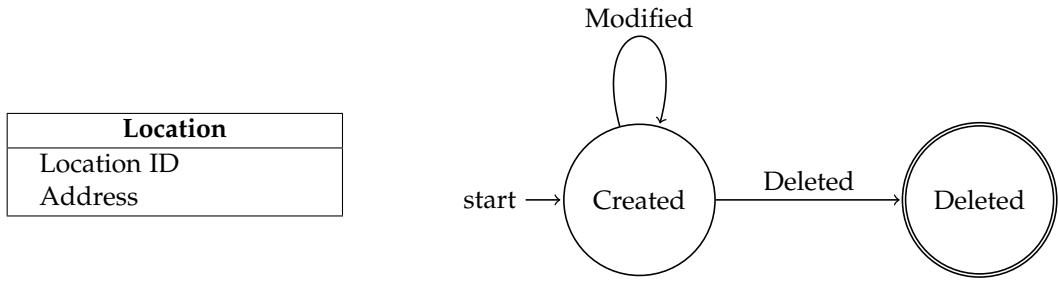


Figure 3.6: State Diagram for Location Class: Assignment and Updates

The Volunteer class does not have a state diagram or events, since it is not directly linked to the scope of Madboks, reducing the third-party tools for customers and the food distribution events. Therefore, it would be redundant to include the possible events of the volunteers.

Lastly, the findings from this section will be used in the system design and implementation phases later in the report. The defined state changes and attributes ensure that all relevant behaviours are accounted for.

3.5 Application domain analysis

In this section, the Application Domain Analysis examines how the system interacts with users and external stakeholders. The purpose is to translate the structural and behavioural models from the problem domain into practical usage scenarios. This analysis identifies actors and use cases, aligning the system's design with user needs.

The current system's actors are discarded after the findings from the problem domain analysis because the scope of the future system was narrowed down to only facilitate interaction between the system and the Admin, Customer and, subsequently, Volunteer.

3.5.1 Actors

Actors are external entities that interact with the system. Identifying actors provides an overview of who uses the system and how they interact with the internal components identified in the problem domain.

Customer

The Customer is a primary actor, as they are the end users who interact directly with the system to reserve food boxes. Additionally, customers engage with events to buy the food boxes.

Admin

The Admin is another primary actor, responsible for managing the system's operations. The admin's role is critical for logistical oversight, event management, and maintaining the system

integrity.

Volunteer

The **Volunteer** serves as a supporting actor, assisting with logistical tasks such as preparing and distributing food during events. Volunteers are directly linked to events. Although they do not initiate system actions like the **Admin** or **Customer**, their involvement is necessary for the success of events.

3.5.2 Usage

Once these actors are identified, the use cases can be determined. Use cases describe how actors interact with the system to achieve specific goals. They define the system's behaviour in various scenarios by capturing the actions between the actors and the system. Use cases are essential for analysing requirements, bridging the gap between high-level concepts and practical design, and ensuring the system supports user needs effectively.

The **Customer** interacts with the system to manage reservations and participate in events. The main use cases for this actor include creating a reservation, where the **Customer** selects an event and reserves a food box, with the system validating availability and confirming the booking. The **Customer** can also modify a reservation by updating details such as time. Another use case is cancelling a reservation, where the system adjusts availability and sends relevant notifications as confirmations. Additionally, the **Customer** receives notifications about their reservations when they create one.

The **Admin** is responsible for managing the system's operations. A key use case is creating an event, where the **Admin** inputs details like date, location, and capacity, and the system makes the event available for reservations. The **Admin** can also modify events, such as updating locations or capacities, with the system adjusting associated reservations automatically. Cancelling an event is another use case, which prompts the system to notify all affected users by a notification. The **Admin** also assigns locations to events.

The **Volunteer** supports event logistics and operations. One use case involves viewing assignments, where the **Volunteer** checks their tasks or events, with the system providing details like location and responsibilities. Updating availability is another use case, where the **Volunteer** informs the system of their availability, allowing it to adjust task allocations.

3.6 Summary

The analysis of Madboks' current system highlights significant challenges due to its reliance on manual processes. Tools like Google Forms and Facebook, used for reservations and communication, lack integration and automation, resulting in inefficiencies, overbooking, and increased administrative workload. While this setup has allowed Madboks to operate effectively at a small scale, it presents challenges in terms of scalability, efficiency, and real-time coordination.

The problem domain analysis addressed these challenges by introducing structured entities such as `Reservation`, `Event`, and `Location`, which replaced inefficient tools and aligned with the system's operational goals. These entities model key workflows, including reservation management, event coordination, and location assignments. The application domain analysis expanded on this by identifying primary actors—`Customer` and `Admin`—and their interactions with the system through specific use cases like creating reservations and managing events.

By narrowing the scope to manageable elements and excluding features like payment processing and food box handling, the analysis ensured a focused draft for improving operations. These findings provide a plan for transitioning to the design phase, where the system's structure, behaviours, and interactions can be translated into concrete requirements for a scalable, user-centred solution.

3.7 Problem statement

The problem statement for this project represents the culmination of research and analysis conducted thus far, serving to define the overarching goal and direction for the development of the solution.

How can food waste among Danish retail stores be minimised by developing a new localised, web-based platform for Madboks that enables them to create and manage events for distributing near-expiry food items and allows users to conveniently book them? The primary evaluation metrics of the solution's success should be its ability to address operational bottlenecks, the user-friendliness of its interface, and scalability, all while emphasising timely food distribution and waste reduction.

Chapter 4

Design

This chapter outlines the design of the new Madboks system, covering requirements definition, database diagram, system architecture, UI design, and design criteria. These design decisions are informed and iterated upon following the analysis of the current Madboks system 3 and insights from the meeting with the product owner C.1. This section also serves to bridge the gap between the project's objectives and its technical implementation. Everything documented so far should now aid in giving a more concrete plan for what is to be implemented, and how.

In the analysis 3, the scope was reduced to only focus on the administration of events on the admin side and the customers' booking experience. This means that other parts of the Madboks organisation, such as retailer communication, logistics and volunteer scheduling are excluded. Reducing the scope and creating realistic expectations for the product owner, increases the probability for the project to succeed with the project and being able to deliver a solid solution.

4.1 Requirements

When considering what requirements the new system should have, inspiration is drawn from the state-of-the-art analysis 2, where design ideas from both Too Good To Go and Facebook were presented. In addition, the analysis of the current solution 3, is essential to identify requirements that ensure that all functionality of the current solution is included, as well as identifying requirements that will improve the system.

Following the analysis, the development team identified key elements that significantly enhance/improve the UX flow, reduce manual labour, and ensure scalability. These elements should be discussed with the product owner and refined into functional requirements.

4.1.1 MoSCoW

To propose suggestions for prioritisation, the MoSCoW prioritisation method is used. This method divides the identified needs into four priority levels, as listed below:

- **Must have:** These are the functions that must be fulfilled for the system to be operational.
- **Should have:** These are the functions that should be fulfilled, as they may add significant value.
- **Could have:** These are the functions that are nice to have, but do not have a major impact if left out.
- **Won't have:** These are the functions that will not be implemented.

This method provides a clear understanding of what should be prioritised from highest to lowest importance.

Must Have	Should Have	Could Have	Won't Have
Event overview: Store and display event information, allowing admins to create, edit, and delete events while customers access relevant details. Origin: Analysis	Event template: Enable admins to create events quickly with consistent default values. Origin: Product Owner	Event template: Allow customers to sign up for a waiting list. Origin: Product Owner	Payment: Enable customers to pay for bookings online, which is costly for a small non-profit. Origin: Product Owner
Booking system: A booking process for when a customer or guest reserve a food box for an event. This includes editing and cancelling the reservations for the logged in user. Origin: Analysis	Contact form: Allow customers to contact Madboks through the website. Origin: Product Owner	Schedule events: Enable admins to schedule when events are published. Origin: Product Owner	Internal communication: Replace WhatsApp and Facebook with integrated communication, which introduces GDPR and scalability challenges. Origin: Product Owner
Accounts: Support account creation for admins and customers, enabling event management and bookings. Origin: Analysis	Volunteer sign-up: Enable customers to request volunteer opportunities to get the email with information. Origin: Product Owner	Recurring events: Automatically generate recurring events. Origin: Product Owner	Logistics: Facilitate managing of routing and communication with retailers, which is not feasible. Origin: Product Owner
Email service: Provide booking confirmations to customers. Origin: Analysis	Reservations for non-users: Not logged-in users can cancel or edit reservations through a link provided in the confirmation email. Origin: Product Owner	Other types of events: Allow admins to create events beyond food distribution. Origin: Product Owner	Mobile app: Develop a mobile app for easier access, which is outside the project scope. Origin: Product Owner
Location management: To manage an event, the admin must be able to perform CRUD operations on a location. Origin: Analysis			Volunteer management: The system could be scaled to also include volunteer management, such as work scheduling. However, this is not in the scope of this project, and will not be included. It is still included in the analysis for creating a basis for future implementation. Origin: Product Owner
Compatibility: The website must be accessible on both desktop and mobile devices. Origin: Product Owner			

Table 4.1: MoSCoW Prioritisation

From the MoSCoW, user stories are created to make the development process more customer-centric and align with an Agile workflow. Each feature is broken down into smaller, actionable

tasks that reflect the needs and priorities of the product owner. The user stories are seen in Appendix G.

4.2 Database diagram

The database diagram provides a detailed overview of the system's structure and the relationships between the different tables in the database. The diagram builds on the class diagram from the problem domain analysis 3.2 where additional tables, attributes and types are introduced. The tables for the system are: Event, Location, Timeslots, Reservation and User. Figure 4.1 illustrates the tables and the way they are saved in the database. The Users table is not displayed in the diagram, since it uses Supabase own 'user' authentication table, which has its own schema [15]. However, the user_id in the Reservations table refers to the id of the users in the Supabase 'user' table. Furthermore, the schema for the Reservation table enables, that a user can create multiple reservations.

The Event table is the central connection point of the diagram. It has a connection to Locations, Reservations, and Timeslots. This is determined from the analysis, where the Event has a relation to both Reservation and Location. The schemas for the Reservation and Timeslot table is designed to handle one to many relationship with the Event table. Similarly, the Location and Event adapt to an one to many, since there can only be one location for an event, whereas a location can be used for multiple events.

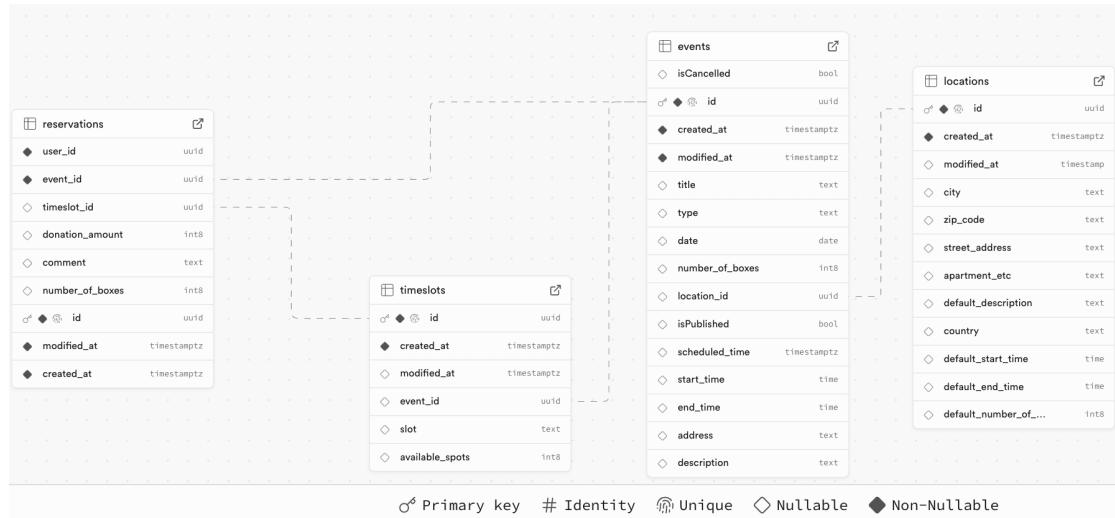


Figure 4.1: Class Diagram

4.3 System Architecture

The system should be built on a scalable architecture that ensures the system can handle growth in the customer base while staying performant, easy to maintain, and flexible to be able to introduce new functionalities. N-layer and n-tier architecture principles are utilised to comply with the requirement of being able to scale by enabling separation of concerns.

4.3.1 N-Layers and N-Tiers Architecture

The n-tier architecture divides the system into distinct physical tiers, each deployed on separate servers to ensure a clear separation of responsibilities. In this setup, the presentation and application tiers are hosted on an Ubuntu server using different ports, while the data tier operates on Supabase's dedicated servers. This design enhances scalability and reliability by allowing additional tiers to be incorporated as needed, distributing risk across multiple servers rather than relying on a single one [12].

Presentation (Frontend server):

The Presentation Layer in the Presentation Tier handles user interfaces, interactions, and data transmission to the application layer. It manages front-end logic like UI rendering, input validation, and API communication via Axios than relying on a single one.

The Presentation Tier hosts the layer, implemented as a React application styled with Tailwind, running on an Ubuntu server with Nginx. Security measures include input validation to prevent cross-site scripting and injection attacks, along with secure protocols for data transmission between frontend and backend than relying on a single one [12].

Application (Backend server):

The Application Layer in the Application Tier manages the system's business logic, processes data, and mediates communication between the presentation and data layers. It handles tasks such as user authentication, authorisation, and maintaining data consistency, while also interacting with external services like Brevo SMTP for emails and Cloudflare for Captcha validation. API requests defined with Fastify facilitate these operations than relying on a single one.

The Application Tier hosts this layer as a Node.js application deployed on an Ubuntu server. This setup ensures secure and reliable handling of API requests than relying on a single one [12]..

Data (Supabase):

The Data Layer in the Data Tier is responsible for storing, managing, and providing access to the system's data. It facilitates CRUD operations (Create, Read, Update, Delete) and ensures data integrity and consistency. This layer interacts with the application layer through secure APIs from Supabase.

The Data Tier hosts this layer on Supabase's servers, utilising a PostgreSQL database. Su-

pabase provides secure APIs to facilitate interactions with the application tier for data access and manipulation [12].

Scalability

By combining n-tier and n-layer architectures, Madboks achieves a system that is both scalable and maintainable. The architecture allows each tier to scale independently and add new layers without disrupting the existing setup. For instance, the frontend can handle increased user loads without affecting backend performance.

The system architecture of the new Madboks system can be seen in figure 4.2.

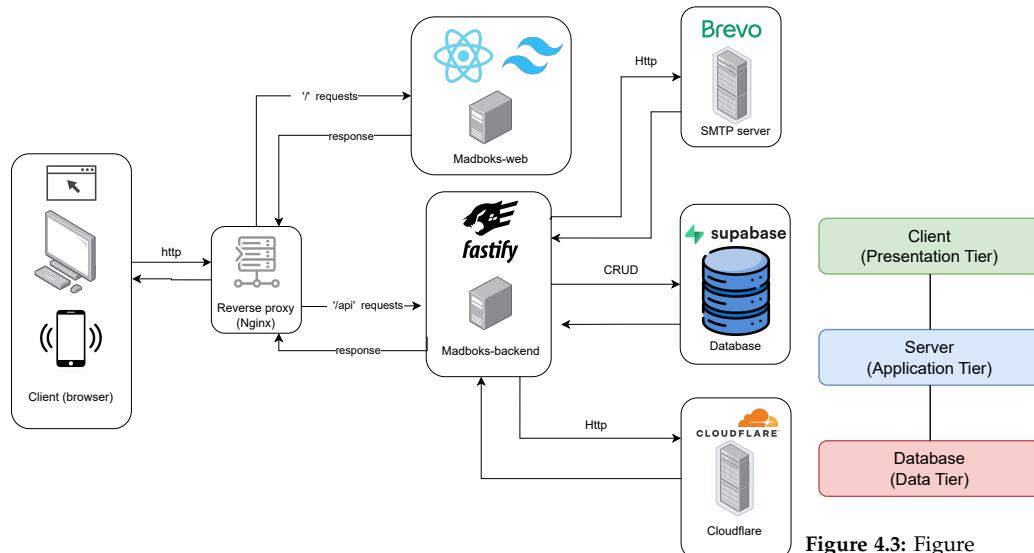


Figure 4.2: Figure showing the system architecture of the new Madboks system

Figure 4.3: Figure showing the tiers used for the system

4.4 Navigation

With the architecture outlined, the focus in design can shift to how the user should navigate the application. This will result in obtaining an idea of which UI elements are needed to be able to design them. To get an overview of the flow of the web application, a navigation diagram is created based on the principles from OOAD [9]. As with the rest of the diagrams and mock-ups created during the design phase, these are expected to provide a good starting point for the implementation process.

4.4.1 Guest Navigation

This navigation diagram for guests at figure 4.4 shows how unauthenticated users interact with the system. The process starts at the Homepage (Not logged in), which serves as the

central hub. From the homepage, users can access the Auth pop-ups by either clicking the Join button or the Login button. If the sign-up or login process is cancelled, users return to the homepage. A successful login transitions users to a logged-in state. Guests can also navigate directly to key pages, including Events, Booking Page, News, Volunteer Sign-Up, and About. The diagram highlights two main flows: accessing authentication through sign-up or login pop-ups and viewing content pages directly.

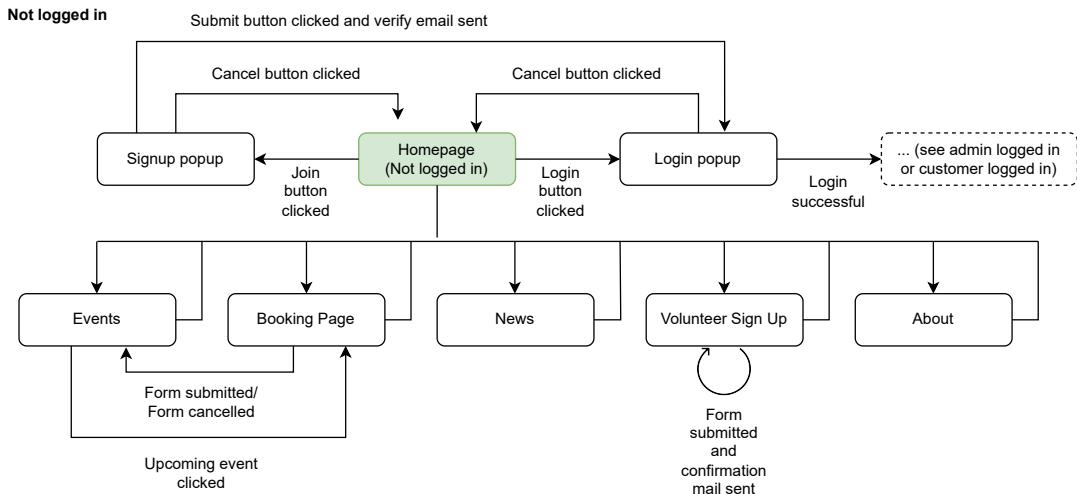


Figure 4.4: Navigation diagram for guests

4.4.2 Customer Navigation

This navigation diagram, in figure E.1, for logged-in users closely resembles the guest diagram but includes additional actions available to customers. The main difference between guests and logged-in users is that the Events page allows viewing the user's reservations. The page also supports editing or cancelling of reservations through respective pop-ups. This diagram mirrors the guest flow while extending functionality to include reservation management, reflecting the added capabilities available to logged-in users.

4.4.3 Admin Navigation

This navigation diagram for admins, in figure E.2, centres around the Dashboard, where admins manage events and locations. From the dashboard, admins can open Active Event or Upcoming Event pop-ups to make changes or cancel them. Clicking the 'Create new event' button triggers a Choose Location pop-up, leading to either the Create New Event or Create New Location pop-up based on location choice. In the Locations section, admins can create new locations or edit existing ones using respective pop-ups, where changes can be applied or cancelled. Logging out or deleting the account redirects the admin to the Homepage in the

state of 'Not logged in'. The diagram simplifies event and location management for administrators.

4.5 UI design

Since there is no concept art or any concrete design ideas to go along, it is the development team's assignment to create flexible ideas that can then be iterated upon in collaboration with the product owner. To achieve this, the team first starts with low-fidelity mockups and makes it clear to the product owner that nothing is set in stone. Subsequently, after processing Zlate's feedback, some more concrete and higher-fidelity mockups are made to serve as guidelines for the frontend implementation.

4.5.1 UI mock-up

The first design mock-up was created using paper prototypes, these prototypes were shown to the product owner and were iterated upon so they could be used to create high-fidelity mock-ups in Figma.

The Figma mock-up, shown in 4.5, provided us with a detailed visual representation of the layout, navigation flow, and core functionalities for the different user roles: guest, customer, and admin. These mock-ups guide the design and highlight how each role accesses specific functions based on its needs, defining the user experience and ensuring that the interface meets the functional needs of each role. By outlining how specific features would be accessed and utilised, the mock-ups served as a foundational guide for the frontend implementation process.

Continuing the iterative process, the design was presented to the product owner for review and approval and was adjusted before implementation. This step ensured alignment with the project's goals and ensured that the product owner's feedback was incorporated early in the process, minimising potential rework during later stages of implementation.

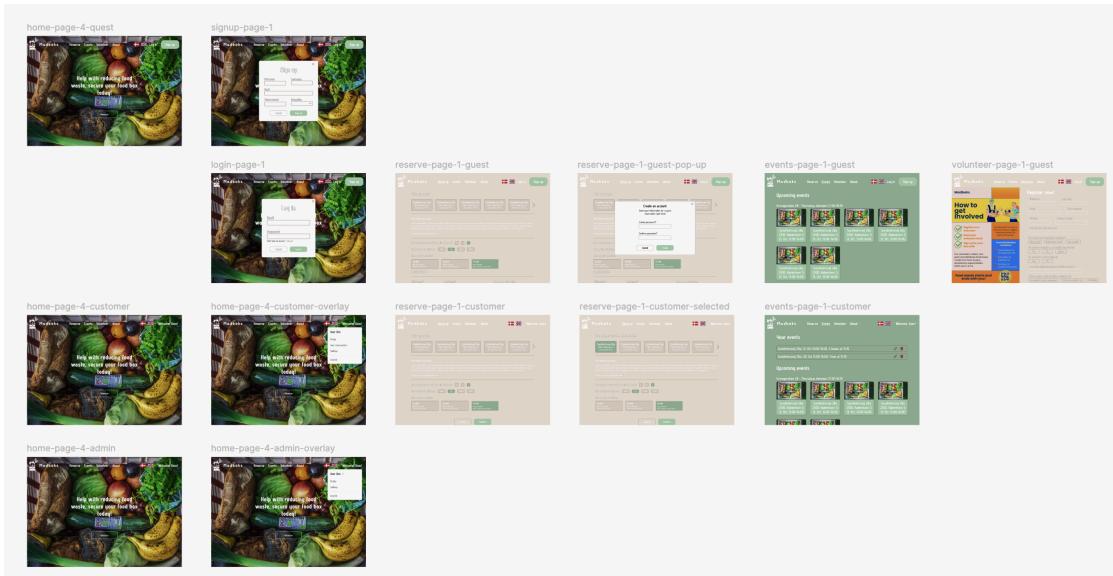


Figure 4.5: Figma

4.5.2 Summary

With the initial design complete, Madboks' new system focuses on making event management and food box booking more centralised, simpler, more effective, and more automated. The design process started by identifying key requirements inspired by modern solutions like Too-GoodToGo and Facebook Events. These requirements were prioritised in collaboration with the product owner using the MoSCoW method to ensure the system delivers the most value to users.

The new system follows a user-centred approach, putting the needs of admins and customers first. This is reflected in detailed user stories and UI mock-ups created through an iterative process incorporating product owner feedback, which ensures that the interface is intuitive and easy to use for everyone. This can be verified through user tests with admins and customers that used Madboks before.

To support Madboks' mission and future growth, the system design also takes scalability and automation into consideration. Processes that were previously manual and located on different platforms should now be centralised and largely automated, saving time and reducing the chance of human error.

Chapter 5

Implementation

In the previous chapters, the problem and application domain were analysed, general requirements for a potential solution were defined, and those requirements were mapped to specific technologies and realised into tangible mock-ups. With a clear plan and a solid understanding of the project's main objectives, the product's development can begin.

5.1 Technology Stack

With a clear common understanding of the problem at hand and the essential requirements for Madboks' new system, the next step is to explore the specifics of how these components will be built. The following section details the technology stack chosen to implement the proposed system.

5.1.1 Development environment

GitHub will be the main platform for managing version control, tracking issues, organizing the product backlog, and handling continuous integration for the project.

During development, feature-branching will be implemented, where each feature or bug fix is developed in a separate branch. Branches are then merged into the main branch only after successful code review and testing, ensuring a stable codebase. For each pull request, a template should be filled out containing a description, screenshots (if relevant), corresponding backend or frontend branch, changes, related issues, and a checklist.

GitHub's projects feature will be used to organize Product Backlog Items (PBIs) by size and priority. PBIs are managed with columns including backlog, ready, in-progress, in-review, and

done. This setup allows for a clear overview of the project's progress and aids in prioritizing tasks based on project needs.

GitHub actions are implemented for running automated testing on all pull requests. Each new change undergoes automatic tests before merging into the main branch, ensuring that code remains clean and functional throughout development. Any dependencies and bugs are documented as issues.

The quality control checklist is the following:

- Code passes all unit tests both locally and using GitHub actions.
- Code follows the established style and architecture agreed upon and passes the linting test.
- Pull request title and description are clear and descriptive.
- Pull request includes any changes and additions and other relevant information and the product backlog is updated to reflect the changes, if applicable.
- The code has passed a manual review

5.1.2 Application Technology

To build a scalable system that efficiently manages Madboks' reservation and event management, the following technology stack was used.

Frontend

The frontend will be developed using React, which is a popular JavaScript library for web user interfaces. React allows the development team to create a responsive interface for Madboks' users, including customers and admins, allowing them to easily navigate the system. React is fast, well-documented, and maintained, and has lots of good libraries, making it an optimal choice for future scalability and ease of maintenance. Also, numerous team members are already familiar with this technology, which means that the development process can start more swiftly.

Backend

The backend is to be implemented with Node.js using TypeScript. Fastify will be used as the API framework, as it is simple, fast, and well-suited for building responsive systems that handle requests quickly.

Notifying customers through emails is deemed to be a high-priority feature of the product. An SMTP (Simple Mail Transfer Protocol) server is essential for sending emails to customers because it is the backbone - standard protocol - of email delivery. When you send an email, the SMTP server processes it, determines the recipient's email server, and routes the email to its destination. Using an off-the-shelf solution ensures that the SMTP server is configured

with proper email protocols (SPF, DKIM, DMARC) to ensure that emails are recognized as legitimate by recipient email providers, and allow bulk emailing. There are many functionally equivalent services, and for this project, Brevo was selected because of its relatively high daily limit of 300 emails with bulk sends allowed [2]. To ensure scalability, simply changing to a paid plan can ensure that the system can handle lots of requests.

5.1.2.1 Database

Supabase will serve as the database for cloud data storage, offering a relational structure and real-time data synchronization, and its free tier will not propose any severe limitations for the developers. Supabase ensures robust data handling for Madboks' requirements, such as managing event reservations, user accounts, and real-time inventory updates. This database choice ensures that relevant data can be accessed and modified seamlessly across different parts of the application.

5.1.3 Ensuring security with Cloudflare Turnstile Captcha

The development team's choice of implementing some basic security measures on the website is with Cloudflare Turnstile Captcha. Running in the background, this third-party tool trusted by numerous massive service providers is designed to differentiate between human users and automated bots visiting a website and sending requests [3].

Other than checking the legitimacy of the user in general, Turnstile should be used with forms, sign-ups, logins and against API abuse.

Compared to other captchas, such as Google's ReCaptcha, Cloudflare's invisibility (no need to solve puzzles, it performs behavioural analysis in the background) makes it a solid choice for not interfering with the user experience and the website's interaction flow. Also, Turnstile does not harvest user data [3].

5.1.4 DevOps and CI/CD

To streamline the development and deployment processes for Madboks, a DevOps strategy was implemented using GitHub Actions. Continuous Integration (CI) and Continuous Deployment (CD) pipelines ensure the system remains reliable, scalable, and easy to maintain throughout its lifecycle.

5.1.4.1 Development lifecycle

The development lifecycle is the same for both the backend and frontend. The developers create feature branches for their PBIs. When the PBI is ready, a pull request (PR) is created from each feature branch to the development branch. All PRs automatically run through the CI pipeline, and another developer conducts a code review before it can be merged into the development branch DEV. When the team is ready to release to production, a PR is created from DEV to the pre-production branch PREPROD. Here it goes through the CI pipeline again and there

manual code review is conducted to ensure code quality and that potential merge conflicts are resolved. If the CI pipeline runs without errors and the PR is approved by another developer, it can be merged into PREPROD. When changes are pushed into PREPROD, it goes through the CI/CD pipeline, where it deployed to the PREPROD server. Now, the developers can test that it runs smoothly on the server and not only locally. When it has been successfully tested on the PREPROD server, a PR to the production branch MAIN can be created. Again, the PR goes through the CI pipeline and manual code review. When approved, PREPROD can be merged to MAIN. When there are pushes to MAIN, the CI/CD pipeline is running, and this time it deploys to the production server.

5.1.4.2 Frontend pipelines

Continuous Integration (CI)

GitHub Actions is configured to run tests and checks on all code changes automatically. This ensures that any new code introduced does not break existing functionality. The full code for the CI pipeline can be seen in Appendix B.1, and figure 5.1, shows how the pipeline looks like on GitHub Actions. The pipeline performs the following tasks:

- **Code Linting and Formatting:** ESLint is run on every pull request to ensure the code adheres to the project's coding standards, improving readability and reducing errors.
- **Automated Testing:** Jest tests are executed to verify the functionality of both new and existing code.
- **Build:** Ensure it is possible to build.

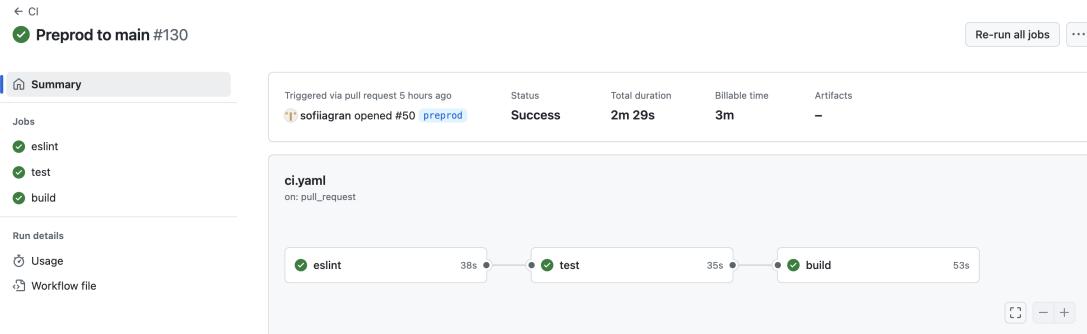


Figure 5.1: CI pipeline web

Continuous Deployment (CD)

The system uses an automated CD pipeline configured in GitHub Actions for deployment. The full code for the CI pipeline can be seen in Appendix B.2, and figure 5.2, shows how the pipeline looks like on GitHub Actions. The pipeline performs the following tasks:

1. **Build Artefact:**

- The frontend application is built using React's production build tools.
- Build artefacts are uploaded for deployment.

2. Artefact Download and Deployment:

- Artefacts generated during the build step are downloaded in the deployment job.
- Using SSH keys securely stored in GitHub Secrets, the pipeline connects to the Ubuntu server.
- The server's target directory is cleaned, and the new build artefacts are uploaded.

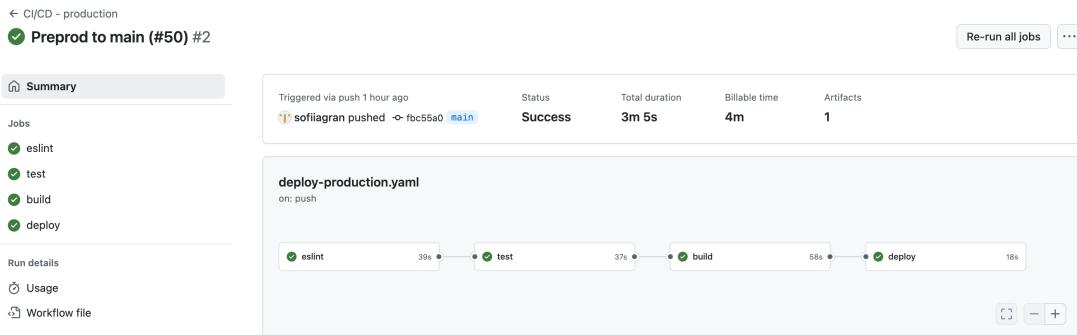


Figure 5.2: CI/CD pipeline web

5.1.4.3 Backend pipelines

Continuous Integration (CI)

GitHub Actions is configured to run tests and checks on all code changes automatically. This ensures that any new code introduced does not break existing functionality. The full code for the CI pipeline can be seen in Appendix B.3, and figure 5.3, shows how the pipeline looks like on GitHub Actions. The pipeline performs the following tasks:

- **Code Linting and Formatting:** ESLint is run on every pull request to ensure the code adheres to the project's coding standards, improving readability and reducing errors.
- **Automated Testing:** Jest tests are executed to verify the functionality of both new and existing code.
- **Build:** Ensure it is possible to build.

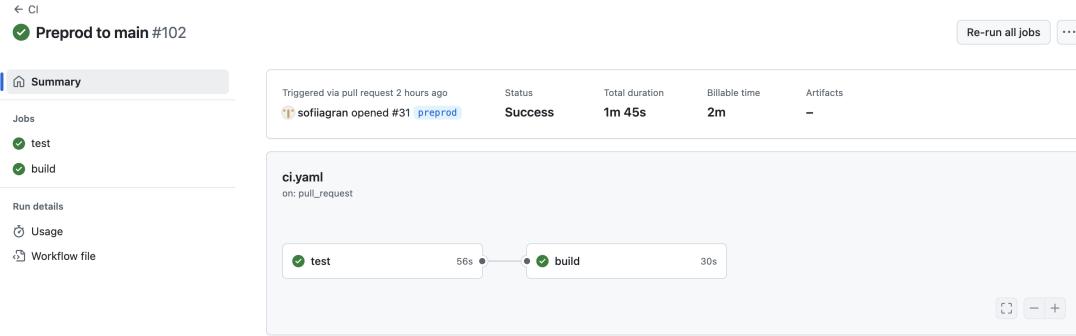


Figure 5.3: CI pipeline backend, the step ‘test’ performs both linting and automated unit/integration tests

Continuous Deployment (CD)

The system uses an automated CD pipeline configured in GitHub Actions for deployment. The full code for the CI pipeline can be seen in Appendix B.2, and figure 5.4, shows how the pipeline looks like on GitHub Actions. The pipeline performs the following tasks:

- ESLint is run on every pull request to ensure the code adheres to the project’s coding standards, improving readability and reducing errors.
- Jest tests are executed to verify the functionality of both new and existing code.
- The backend builds using xx
- Using SSH keys securely stored in GitHub Secrets, the pipeline connects to the Ubuntu server.
- The server’s target directory is cleaned, and the new build are uploaded.

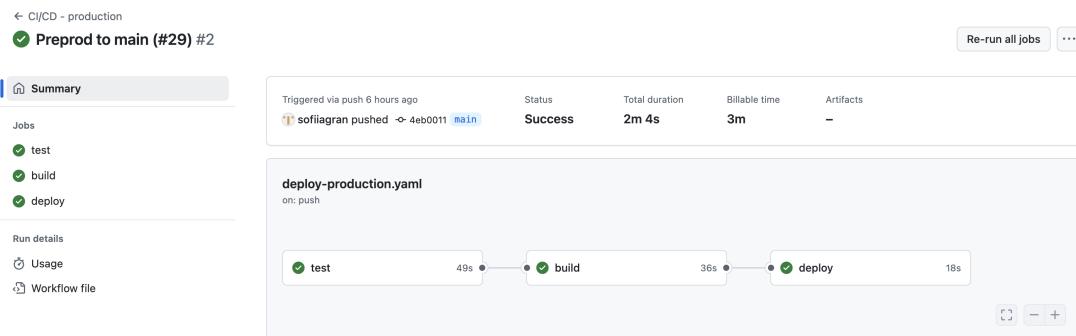


Figure 5.4: CI/CD pipeline backend

5.1.5 Hosting on server

Both the frontend and backend of Madboks are hosted on a dedicated Ubuntu server to ensure reliability and scalability. The server was created using OpenStack, accessed through the platform <https://strato-new.claaudia.aau.dk>. The backend is deployed as a Fastify application running locally on port 4000. To make the backend accessible externally and to improve security and scalability, Nginx works as a reverse proxy. Nginx forwards incoming requests to the backend server and ensures that it is properly configured for handling HTTP traffic, enhancing both performance and security. For managing the server, Terminus is used. Terminus simplifies server administration tasks, allowing the team to monitor and manage the server efficiently. To ensure the security of the Madboks platform, the Nginx configuration is designed to restrict access to the /api/ endpoints. These endpoints are configured to accept requests only from the official Madboks website, and not directly via browsers or external clients.

5.1.6 Docker

Using docker containers and images could be another solid choice to improve the deployment pipeline. Docker simplifies application deployment by packaging the frontend, backend, and dependencies into lightweight, portable containers.

Docker containers encapsulate the entire runtime environment, including application code, libraries, dependencies, and system tools. This ensures that all developers, testers, and production environments use the same configurations, reducing the risk of discrepancies between environments.

Containers allow for streamlined deployment processes. Instead of setting up dependencies manually on servers, the Docker images can be pulled and run on any host with Docker installed, reducing deployment complexity and time.

Docker's lightweight nature makes it easy to scale the application horizontally. An example of scaling horizontally is running multiple instances of the backend container in case of increased API traffic. Kubernetes could be another helpful tool for orchestrating horizontal scaling when working with Docker containers.

5.2 Application of Agile Principles

This project's software development workflow follows a fundamentally Agile approach. This approach was chosen because the events support an iterative process of continuous value delivery and evaluation in the context of external collaboration.

A key figure in this project is Roxana Zlate taking on the product owner role. Zlate is the founder and project manager at Madboks and a logistics consultant at the United Nations Populations Fund [8]. As product owner, their primary task is to represent and voice cus-

tomer and stakeholder needs while maintaining both a clear long-term product vision and meaningful short-term goals. It will be crucial to continuously collect feedback from the product owner throughout the software development process, and keep them in the loop while performing product backlog refinement and task prioritisation.

The development team responsible for designing and implementing the software solution consists of the authors of this paper. Their focus should be on constantly providing valuable increments to the product owner while maintaining transparency and facilitating open communication about the product translating stakeholder feedback into implementable refinements. There is no dedicated Scrum Master in the team because every member is familiar with Agile, and everyone actively participates in all Agile events. Responsibilities are handled as a unit and management duties are implicitly shared. Any team member can facilitate meetings and events.

The development team has no dedicated management support - however, the team's supervisor, Tung Kieu, could be considered to fulfil a similar role. Kieu's support in providing the necessary resources and tools, as well as supporting team autonomy while aligning the work with the study goals is closely adjacent to what a manager would do.

The development process consists of 2-week sprints, each sprint starting with a sprint planning session where the most important goals of the upcoming sprint are solidified and translated into product backlog items. Product backlog items (PBIs) are estimated in size and impact, and broken up into frontend and backend subparts or smaller tasks if needed. During a sprint, developers take as many items as they realistically can, favouring the ones with smaller sizes and larger impact.

During development, daily stand-ups are not strictly necessary but developers should make sure that transparency is well maintained and dependencies are recognised and tackled smoothly. At the end of the sprint, the developments are presented to the product owner and their input is recorded and formulated into manageable tickets. When considering product backlog items/tickets done, there is no "Definition of Done" list to look at, because having a static list was often overlooked in earlier semester projects and was deemed redundant. Instead, frontend items need acceptance from the product owner, and backend items need to be both manually and automatically tested, reviewed, and considered stable.

The team is already familiar with Agile software development, so following a less-rigid, somewhat less structured version of Agile development that better suits the nature of a university project did not mean that sprint planning or sprint reviews received less attention despite the lack of regular daily stand-ups or sprint retrospectives.

GitHub's projects and issues features are used throughout the project to support Agile processes during development [6]. Projects are a good tool for keeping track of a virtual product backlog, and issues are useful for dependency tracking and any potential tickets that may not be directly relevant to the product owner but are more DevOps-focused.

To keep the process hands-on and dynamic, sprint planning sessions and PBI drafting are primarily done on a blackboard, and projects and issues are used for documentation and for tracking updates and progress afterward. In general, more physical and hands-on tools have the advantage of preserving momentum and train of thought during discussions and brainstorms.

With the workflow in place, the team can begin implementing the software solution. For readability, product backlog items are labelled with sprint and frontend/backend item numbers, for example, S1F4 is the 4th item of the first sprint, and S3B2 is the second backend item of the third sprint. If a product backlog item has undergone multiple iterations, this will also be indicated in the name. Only the most relevant PBIs will be included, otherwise, they can be found in the Appendix H.

5.3 Sprint 1

5.3.1 Sprint Planning

Following the meeting with the product owner (see Appendix C.1), it was clarified that the primary goal of the initial sprint(s) was to provide users with all of the same functionalities as the ones currently available. A prerequisite to achieving this was setting up the central components of the system — namely, a React web template, Fastify API framework for the backend, and a Supabase database. Fine-tuning of the UI and smaller details of forms and such were not that important yet.

Each page of the website should be built up using reusable components, whose business logic is, if complex enough further separated into its own service file(s). Overall, a main focus on the frontend should be a separation of concerns and reusability. Working on separate components should also help with the development team working on the same page in parallel.

Given that the team gets their Figma mock-ups accepted or rejected before implementation, the acceptance on the frontend side is highly dependent on recreating the styling from Figma using Tailwind.

On the backend side, acceptance is a bit more problematic since the product owner is not a domain expert. Here, the development team is highly dependent on the quality of its internal review process. To ensure some sort of acceptance, the feature should still be shown to the product owner during sprint meetings.

5.3.2 Frontend items

- **S1F1:** Homepage displaying the most important information about Madboks (first iteration) H.0.0.1

- **S1F2:** Navigation bar with different items depending on admin/customer login and non-logged-in user (hardcoded first iteration) H.0.0.2
- **S1F3:** Upcoming events component displaying all of the upcoming events where users can book boxes (only mock data) H.0.0.3
- **S1F4:** Your events component that displays the events that a user has booked (only mock data) H.0.0.4
- **S1F5:** Location creation component for admins H.0.0.5
- **S1F6:** Event creation component for admins enabling the creation of new events H.0.0.6
- **S1F7:** Reservation form enabling customers to reserve boxes at events (only mock data) H.0.0.7

5.3.3 Backend items

- S1B1: CRUD for events enabling admins the creation, update and deletion of events
- S1B2: CRUD for location templates enabling admins the create, update and deletion of location templates
- S1B3: CRUD for reservation enabling customers to create, modify and cancel a reservation

5.3.3.1 S1B1, S1B2, S1B3: CRUD for events, locations and reservation

Create, Read, Update and Delete functionality was created for events, locations and reservations. There were also implemented unit tests for all methods. It followed best practices by being divided into routes, controller, service and model. Later on in the project there were made changes to the the specific implementation of events, locations and reservations, since the system had to handle time and timeslots differently. All classes extend the 'Base' class and they all use the general CRUD implementation defined in baseService.ts.

Read methods: The following methods are performing read operations to the database. For instance, the method 'findByfield' returns all rows in a table where a field equals a specific value (this could be id for example) as seen in code snippet below. The function inputs a field name and a value, this will only work with string data type fields, and then it makes a call to Supabase to fetch the data that matches the value.

```

1 // Find by field
2     protected async findByField(
3         field: string,
4         value: string,
5     ): Promise<{
6         data: T[] | null;
7         error: PostgrestError | null;

```

```

8 }> {
9   const { data, error }: PostgrestResponse<any> = await this.supabase
10  .from(this.tableName)
11  .select('*')
12  .eq(field, value);
13
14  if (error) return { data: null, error };
15
16  return { data: data!, error: null };
17 }

```

Create methods: The following method is performing create operations to the database. The method 'createMany' inserts a list of rows formatted as json objects into a class specific table, such as Event.

```

1 // Create many
2   protected async createMany(
3     records: Jsonable[],
4   ): Promise<{ data: Jsonable[] | null; error: PostgrestError | null }> {
5     const { data, error } = await this.supabase
6       .from(this.tableName)
7       .insert(records)
8       .select('*');
9
10    if (error) return { data: null, error };
11
12    return { data: data!, error: null };
13 }

```

Update methods: The following method is performing update operations to the database. The method 'updateField' updates a specific field by finding the record with the id matching the id parameter.

```

1 // Update field
2   protected async updateField(
3     id: string,
4     field: string,
5     value: any,
6   ): Promise<{ data: T | null; error: PostgrestError | null }> {
7     const updateData = { [field]: value };
8     const { data, error } = await this.supabase
9       .from(this.tableName)
10      .update(updateData)
11      .eq(BaseFields.ID, id)

```

```

12     .select('*')
13     .single();
14
15     if (error) return { data: null, error };
16
17     return { data: data!, error: null };
18   }

```

Delete method: The following method perform a delete operations to the database. The method 'deleteByField', deletes rows from the database based on a specific field value like 'updateByField'.

```

1 // Delete by field
2   protected async deleteByField(
3     field: string,
4     value: string,
5   ): Promise<{ error: PostgresError | null }> {
6     const { error } = await this.supabase
7       .from(this.tableName)
8       .delete()
9       .eq(field, value);
10
11     return { error: error || null };
12   }
13 }

```

5.3.4 Sprint Review

Based on the product owner's feedback, the homepage needs to get a makeover, where the view snaps to the sections of the page. Other than that, some minor frontend updates are also requested. The more noteworthy adjustments are described as "second iterations".

5.4 Sprint 2

5.4.1 Sprint Planning

The focus of Sprint 2 was to build upon the functionalities established in Sprint 1. This included enhancing the user interface, integrating email service functionalities, creating the admin dashboard, and incorporating feedback and requirements from the product owner. The overall sprint goal is to hook up all of the frontend components to the backend and thereby replace mock data with actual data fetched from the database.

5.4.2 Frontend items

- S2F1: Homepage (second iteration)
- S2F2: Volunteer page H.0.0.8
- S2F3: Email service frontend and corresponding forms (book a box and volunteer signup, emails that do not land in spam)
- S2F4: Admin dashboard - new UI and edit location functionality H.0.0.10
- S2F5: Login and signup UI connected with supabase auth
- S2F6: Connect upcoming events, your events and reservation with backend and authentication
- S2F7: Event popup, showing information and description of the event. H.0.0.9
- S2F8: Connect topbar with auth. H.0.1

5.4.2.1 S2F1: Homepage (second iteration)

Based on the product owner's feedback, the homepage was updated to snap between sections, and the call-to-action buttons now stand out more clearly. Furthermore, the call-to-action buttons' stylings are changed to be more uniform and to stand out better. Some animations are also added to the location cards and pictures to make the pages more engaging. Lastly, the page is made more responsive and buttons are connected to other pages.

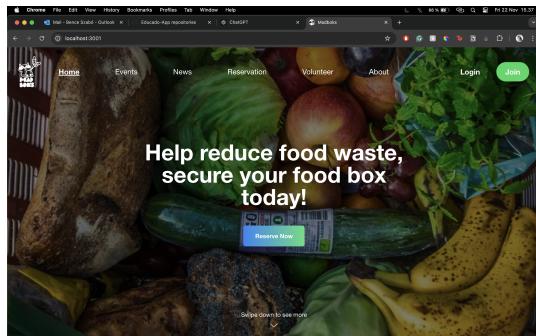


Figure 5.5: Second iteration of the homepage 1

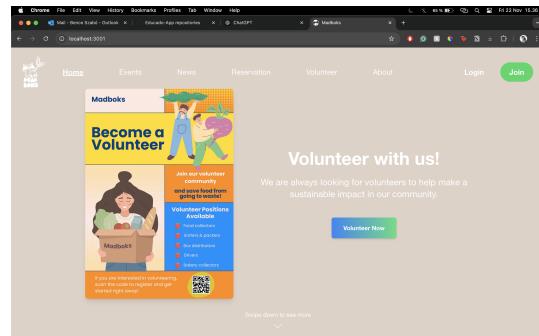


Figure 5.6: Second iteration of the homepage 2

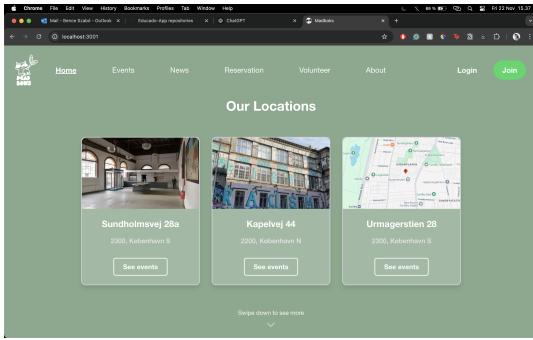


Figure 5.7: Second iteration of the homepage 3

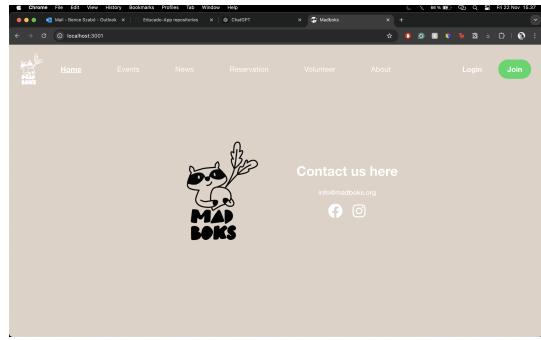


Figure 5.8: Second iteration of the homepage 4

5.4.2.2 S2F3: Email service frontend and corresponding forms

Sending confirmation emails out for box reservation confirmation and volunteer signups were highly prioritised features by the product owner. Emails were selected because users cannot be expected to check notifications on-site.

For the frontend side, forms for both box reservation and volunteer signup were connected to the newly defined email service file containing all email-related frontend functionalities.

```

1  export const sendVolunteerSignup = async (volunteerInfo: Record<string, string>) => {
2    try {
3      const response = await axios.post(`$url}/send-volunteer-signup`, volunteerInfo);
4
5      if (response.status === 200) {
6        console.log('Volunteer signup email sent successfully!');
7      } else {
8        console.error('Failed to send volunteer signup email:', response);
9      }
10    } catch (error) {
11      console.error('Error sending volunteer signup email:', error);
12    }
13  };

```

This function, `sendVolunteerSignup`, sends volunteer sign-up data (`volunteerInfo`) as a POST request to a server endpoint (`$url/send-volunteer-signup`) using Axios. It logs a success message if the server responds with status 200, otherwise logs an error message. Any network or server errors during the request are caught and logged. The reservation confirmation function works identically but receives its own function.

5.4.2.3 S2F5: Login and signup UI connected with Supabase auth

Supabase is the system's chosen database, which comes with built-in features for authentication. The auth-flow implemented starts by signing up. The flow supports automatically sending emails from a custom SMTP server to confirm the sign-up and a link to the system's confirmation page. The link URL has an access and refresh token embedded for the user, which gets saved in the Supabase auth session and in "HTTPS only" cookies. When the user uses the site after confirming the sign-up, their access is used to retrieve their session, keeping them logged in. The session gets terminated if the user logs in on another device, logs out or deletes their account.

Furthermore, there is functionality to change passwords, where the user gets an email to another confirmation page to confirm their new password. Lastly, the user can only access certain parts of the application when acting as a guest, logged-in customer or admin. For example, only the admin can access the event and location management pages.

5.4.2.4 S2F6: Connect upcoming events, your events and reservation with backend/database

The components now dynamically update based on real-time data from the backend, ensuring accurate and up-to-date information. E.g. when a user reserves a box, the reservation is registered in the database and can be seen in the 'your events', and when an admin creates an event, it can be seen in 'upcoming events' and as an option in the reservation page, the components can be seen in figure 5.9.

To display a list of event cards, a sub-component for a single event card was first created and then mapped within the upcoming events component. The number of visible items adjusts dynamically based on the window size, tracked via an event listener. A Madboks logo appears at the end of the list when no more items are available, and a fade animation ensures smooth transitions in the carousel.

```
1 // React Component for Event Display
2 <div className={"flex justify-start overflow-hidden whitespace nowrap flex-grow
3 items-center transition-opacity duration-300
4 ${isAnimating ? 'opacity-0' : 'opacity-100'}>
5   {events
6     .sort((a, b) => new Date(a.date).getTime() - new Date(b.date).getTime())
7     .slice(currentIndex, currentIndex + visibleEvents)
8     .map(event => (
9       <Event
10         key={event.id}
11         title={event.title}
12         date={new Date(event.date).toDateString()}
13         opening_hours={formatTime(event.start_time, event.end_time)}
14         id={event.id}
15         onClick={onClick}
16       />
17     )}
```

```

17     ))}
18     {currentIndex + visibleEvents >= events.length && (
19       <div className="flex flex-col items-center">
20         <img
21           src={madboksLogo} alt="Madboks Logo"
22           className="mb-2" style={{ maxWidth: '10rem' }}
23         />
24       </div>
25     )}
26   </div>

```

The transition-opacity and duration-300 classes enable smooth visibility transitions, toggling between opacity-0 (invisible) and opacity-100 (fully visible) based on the isAnimating state.

The events array is first sorted by date in ascending order using the Array.prototype.sort method. Each event's date is converted to a Date object and compared using getTime() to ensure accurate sorting. After sorting, the Array.prototype.slice method selects a subset of events to display, starting at currentIndex and including up to visibleEvents items.

The selected events are then mapped into Event components, with properties such as title, a formatted date (using toDateString()), opening_hours (calculated with a helper function formatTime), and an id. Each Event component also includes an onClick handler for interactivity.

At the end of the event list, if the currentIndex plus the number of visibleEvents exceeds the total number of events, an additional div is rendered. This contains a centered logo. This ensures that a fallback element is displayed when all events have been shown, providing a visual cue to the user.

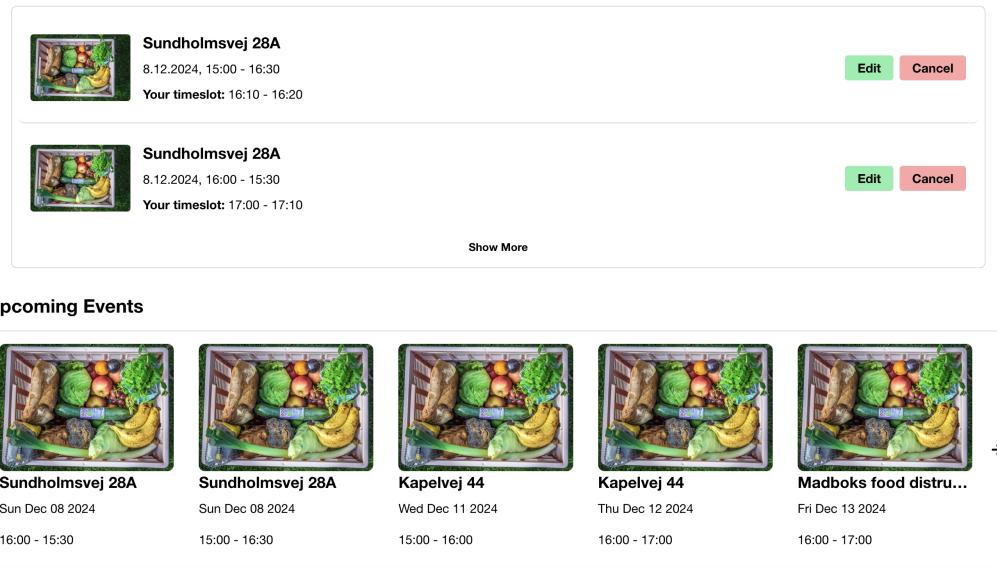


Figure 5.9: Upcoming events

5.4.3 Backend items

- S2B1: Setting up the server and hosting + pipelines H.0.2
- S2B2: Email service backend and email templates (SMTP server)

5.4.4 S2B2: Email service backend and email templates

The email service backend uses Brevo's SMTP service to send emails out to users, potentially in bulk.

The service follows a singleton architecture operating from a single instance of a class. Its functionalities are broken down into two files, mailRoutes and mailService, both contained in a single folder also containing their templates.

Using Brevo is very straightforward with Fastify. The function in mailService for sending a confirmation mail simply needs the correct arguments and a template.

```

1  public async sendConfirmationEmail(to: string): Promise<void> {
2    const subject = 'Thank you for your order!';
3    const text = 'Remember to pick up your order at the specified time.
4    For any questions, please contact us at help@madboks.com.';
5    const html = this.loadTemplate('bookingtemplate.html');
6

```

```
7     await this.sendMail(to, subject, text, html);
8 }
```

The code defining the POST endpoint at /api/send-confirmation-email using Fastify is seen in Appendix H.0.3. It extracts the to field from the request body, validates it, and returns a 400 error if it's missing. If valid, it uses the MailService to send a confirmation email. On success, it responds with a 200 status and a success message; on failure, it logs the error and returns a 500 status with an error message.

Email templates consist of straightforward HTML to adhere to the standards providers. For each different email, their template is imported and used inside the corresponding email service function.

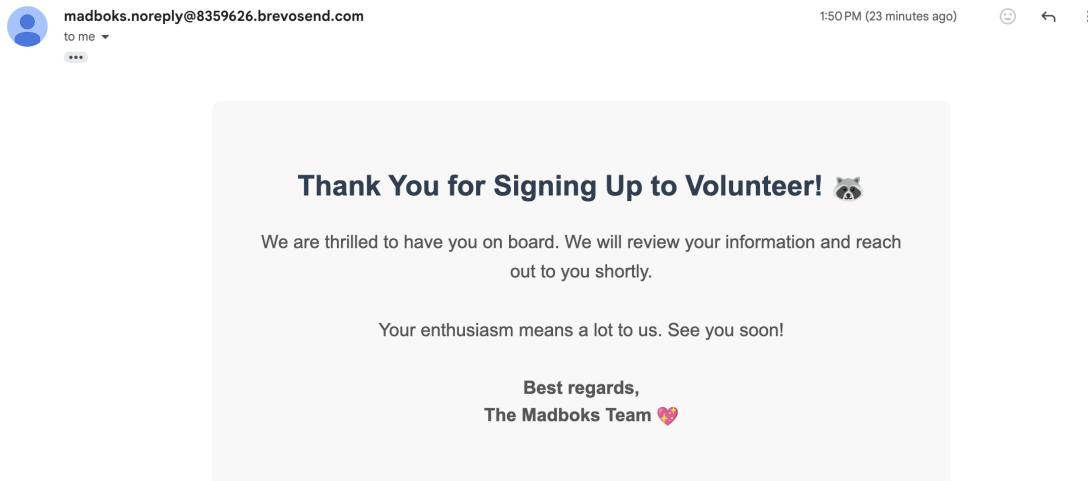


Figure 5.10: Email example

For now, emails do not include dynamic data. The main focus during this sprint was to set the technology up and make sure it worked.

5.4.5 Sprint Review

The email templates need to receive a minor makeover and include more personalised information. Also, some more frontend changes are requested, most importantly mobile compatibility for at least the customer side of the application. Everything else is accepted and the website is now in a nearly presentable shape. Therefore, user tests are scheduled for the next sprint.

5.5 Sprint 3

5.5.1 Sprint Planning

The main focus of sprint 3 is to finish up the website's core functionalities, improve security, and make the system stable for production. The website needs to be deployed and stable enough to perform remote tests with both admins and customers.

5.5.2 Frontend items

- S3F1: Mobile Compatibility
- S3F2: Cloudflare Turnstile (Captcha) and integrate with login, signup and reservations
- S3F3: About us page H.0.4
- S3F4: Admin dashboard makeover, timeslot fix and edit active and upcoming events
- S3F5: 'Event' page updates - UI fix and pop-up to edit and cancel booked events.
- S3F6: Reservation with timeslots
- S3F7: Location page for admin

5.5.3 S3F1: Mobile Compatibility

Based on the product owner's feedback, being a relatively high-priority requirement and general accessibility reasons, it was chosen to spend time assuring that the entire website is mobile-compatible. This meant reworking on the frontend code, focusing on smaller screen sizes and responsiveness during any resizing.

On the positive side, several pages were already highly responsive before starting on this PBI, but in hindsight, this should have been considered acceptance criteria for all pages from the start of the project.

Some of the most crucial reworks were the top bar turning into a burger menu when the screen got to an approximate tablet/phone size. Furthermore, it had to be ensured that images and components scaled properly and the resulting layout would still look acceptable.

5.5.4 S3F2: Cloudflare Turnstile

Cloudflare Turnstile is used for the captcha challenges, ensuring that user traffic is not by bots. The invisible captcha is used so that the user does not have to interact with a UI component, but the process gets executed in the background. The captcha is a form that can be added on specific pages where the captcha must be issued to verify the user. Supabase provides support for custom captcha when using auth functionality, which means that Turnstile has to validate the user to be able to log in and sign up.

In the code snippet, the Turnstile uses a sitekey provided by Cloudflare which is not a secret key and the name of the action for the form where it is implemented at. The function verifyToken sends the challenge token to the backend and if the captcha fails, an error message is shown.

```

1  const TurnstileComponent = ({ formName }: TurnstileComponentProps) => {
2    const { setErrorMessage, verifyToken } = useTurnstile();
3
4    const sitekey = process.env.REACT_APP_TURNSTILE_SITE_KEY || "";
5
6    if (!sitekey) {
7      console.error("Turnstile site key is not set.");
8      setErrorMessage("Turnstile site key is not set.");
9    }
10
11   return (
12     <Turnstile
13       sitekey={sitekey}
14       action={formName}
15       refreshExpired="auto"
16       onVerify={(token) => verifyToken(token)}
17       onError={(err) => {
18         console.error('Turnstile error', err);
19         setErrorMessage(err);
20       }}
21       onExpire={() => {
22         console.error('CAPTCHA expired');
23         setErrorMessage('CAPTCHA expired');
24       }}
25     />
26   );
27 }
```

5.5.5 S3F4: Admin dashboard makeover, timeslot fix, and edit active and upcoming events

The Admin dashboard now differentiates between 'active' and 'upcoming' events. The active events are published and can be seen by the customer, but the upcoming events are not yet published, and can only be seen by admin. When an admin creates an event and fills out the start/end time of the event, the system automatically creates timeslots with 10-minute intervals between them. As wanted by the PO, the total number of boxes for the event is equally distributed between the timeslots.

In the dashboard, it is possible to click on upcoming events to edit them, this form is the same as when you create a new event. The active events have a different edit page, as it is only possible to add/remove boxes. When the number of boxes is updated the timeslots are updated accordingly. If the admin tries to change the total number of boxes to a number that is less than what is already reserved, the admin gets an error message, and cannot proceed with the changes.

A Edit Event: Sundholmsvej 28A, 2300 København S
Address: Sundholmsvej 28A, 2300 København S Date: Sun Dec 08, 16:00 - 16:30
Description:
Reservations are considered automatically confirmed once you receive an email with a copy of your responses once you complete filling in this form.
You will get a box of 7-10 kg. All of fruits, veggies, bread and other good stuff. The boxes are made with food surplus & food waste donated by supermarkets, so some of it might be overripe, weird or a bit bruised. We encourage you to give it some care and a light trimming & save what is still edible. No food should go to waste unless it is rotten.
PLEASE ONLY BOOK IF YOU KNOW YOU CAN COME! IF YOU DON'T COME, YOUR BOX GOES TO WASTE. In order to save the waste, we will keep your booking for 10-15 minutes, after that the food will be given to other people. If you know you would be late for picking up your box, please contact us in advance via Facebook or Instagram (Methodbox) or WhatsApp (methodbox_jo).
Number of boxes: 100
Original Number of Boxes: 100 Reserved Boxes: 3
Timeslots:
16:00 - 16:10 Spots: 12 16:10 - 16:20 Spots: 11 16:20 - 16:30 Spots: 11 16:30 - 17:00 Spots: 11 17:00 - 17:30 Spots: 11 17:30 - 18:00 Spots: 10
17:00 - 17:10 Spots: 10 16:40 - 16:50 Spots: 10
Apply changes Cancel Event

Figure 5.11: Edit active event

S Update event
Title: Kapelvej 44 Address: Kapelvej 44, 2200 København N
From: 16:00 To: 17:00
Date: 11.12.2024 Number of boxes: 20
Description:
Bring your own bags - one for fruit & veggies and one for bread.
PLEASE ONLY BOOK IF YOU KNOW YOU CAN COME! IF YOU DON'T COME, YOUR BOX GOES TO WASTE.
You will get a box of 7-10 kg. full of fruits, veggies, bread and other good stuff. The boxes are made with food surplus & food waste donated by supermarkets, so some of it might be overripe, weird or a bit bruised. We encourage you to give it some care and a light trimming & save what is still edible. No food should go to waste unless it is rotten.
We will keep your booking for 10-15 minutes, after that the food will be given to other people.
If you know you would be late for picking up your box, please contact us in advance via Facebook (Methodbox) or Instagram (methodbox_jo).
Thank you!
Please do not attend our events if you expect our fruits and veggies to be in perfect condition - they are not. We are here to save edible food from going to waste.
Timeslots:
16:00 - 16:10 Spots: 4 16:10 - 16:20 Spots: 4 16:20 - 16:30 Spots: 1
16:30 - 16:40 Spots: 3 16:40 - 16:50 Spots: 3 16:50 - 17:00 Spots: 1
Apply changes Publish Delete Event

Figure 5.12: Edit upcoming event

Dashboard Locations Volunteer Events
+ Create new event
Active events
Sundholmsvej 28A Sun Dec 08, 16:00 - 16:30 Available Boxes: 97 Total Boxes: 100 Edit Cancel
Sundholmsvej 28A Sun Dec 08, 16:00 - 16:30 Available Boxes: 100 Total Boxes: 100 Edit Cancel
Show More

Figure 5.13: Upper part of admin dashboard

Scheduled Events
København, Wed Dec 11 2024 16:00 - 17:00 Publish
København, Sat Dec 14 2024 16:00 - 17:00 Publish
All Locations
København, Urmøsterstien 28, Sundholmsvej 28A, dsf

Figure 5.14: Lower part of admin dashboard

Method: Update timeslots

When the number of boxes in the UI is changed, the handleChange method is called. The handleChange method dynamically updates the form data when the total number of boxes is modified. It prevents overbooking by comparing the new number of boxes with already reserved boxes and disables submission if the new value is invalid.

```
1 const handleChange = (e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement |  
2   HTMLSelectElement>) => {  
3   const { name, value } = e.target;  
4   setFormData((prev) => {  
5     const updatedFormData = { ...prev, [name]: value };  
6     if (name === 'number_of_boxes') {  
       const newNumberOfBoxes = Number(value);
```

```

7     if (newNumberOfBoxes - reservedBoxes < 0) {
8         setIsOverbooked(true);
9         setDisabled(true);
10        return updatedFormData;
11    }
12    setIsOverbooked(false);
13    setDisabled(false);
14    updateTimeslots(newNumberOfBoxes - originalNumberOfBoxes);
15}
16    return updatedFormData;
17});
18}

```

This calls the method `updateTimeslots` seen in Appendix H.0.5. The `updateTimeslots` method recalculates and redistributes the number of available spots across timeslots when the total number of boxes changes. It first calculates the difference between the target and current total spots, distributing the adjustment equally across all timeslots. Any remainder from the division is distributed incrementally to individual timeslots. If any timeslot ends up with negative spots, these are corrected by redistributing the deficit among other timeslots with available spots. This ensures that all adjustments maintain a valid state with no negative values while keeping the distribution as balanced as possible. The backend code for updating is presented in subsection 5.5.12.

5.5.6 S3F5: 'Event' page updates - UI fix and pop-up to edit/cancel booked events.

In the 'Events' page, the customers can see what timeslot they have booked for, and click on 'edit' to change the timeslot, or 'cancel' to cancel the reservation. In addition, the name 'your events' is changed to 'your bookings' based on feedback from the PO. Also, the default image is changed to the 'Madboks' image used on their current Facebook events. This can be seen in figure 5.15.

Your Bookings

	Sundholmsvej 28A 8.12.2024, 16:00 - 15:30 Your timeslot: 16:40 - 16:50	Edit	Cancel
	Kapelvej 44 11.12.2024, 15:00 - 16:00 Your timeslot: 15:50 - 16:00	Edit	Cancel

[Show More](#)

Upcoming Events

	Sundholmsvej 28A Sun Dec 08 2024 16:00 - 15:30		Sundholmsvej 28A Sun Dec 08 2024 15:00 - 16:30		Kapelvej 44 Wed Dec 11 2024 15:00 - 16:00		Kapelvej 44 Thu Dec 12 2024 16:00 - 17:00		Madboks food distr... Fri Dec 13 2024 16:00 - 17:00
--	--	--	--	--	---	--	---	--	---

Figure 5.15: Event page

The backend code for updating timeslots is presented in subsection 5.5.12.

5.5.7 S3F6: Reservation with timeslots

The reservation page is updated so it works with timeslots. The page only shows available slots, ensuring that customers can only book within the event's capacity to prevent overbooking. When booked, it updates the available spots for the timeslot booked in the database and the admin dashboard.

Only the available timeslots will be displayed, and if an event is fully booked (no available timeslots), the message 'this event is fully booked' will be displayed. The timeslots and 'info before you book' (event description) are only displayed when you select an event to reserve, see figure 5.17.

Validation is added so that the customer has to fill out all fields and 'confirm' terms and conditions (info before you book) before they can book. Custom error messages and success pop-up are implemented. In addition, if the user is logged in, their contact info is already added, to effectively proceed with their booking. If not logged in, the 'contact' fields have to be manually filled out. The contact information part is seen in figure 5.18.

Book your Madboks

Select an event

Sundhedsborg 28A, 2300 København S Sun Dec 09, 16:00 - 16:30	Sundhedsborg 28A, 2300 København S Sun Dec 09, 15:00 - 16:30	Kapelvej 44, 2300 København N Wed Dec 11, 15:00 - 16:00	Kapelvej 44, 2300 København N Thu Dec 12, 16:00 - 17:00	Kapelvej 44, 2300 København N Fri Dec 13, 16:00 - 17:00
--	--	---	---	---

Show More

Info before you book

I understand and I will try to save as much food as possible from going to waste!

How many boxes would you like to book? 1

Choose time for pickup:

Choose your donation:

25 DKK Minimum donation Covers food transport	50 DKK Medium donation Covers all event logistics	75 DKK Large donation Helps organise future events
---	---	--

Figure 5.16: Reservation page without a selected event

Book your Madboks

Kapelvej 44, 2300 København N, Fri Dec 13, 16:00 - 17:00

Sundhedsborg 28A, 2300 København S Sun Dec 09, 16:00 - 16:30	Sundhedsborg 28A, 2300 København S Sun Dec 09, 15:00 - 16:30	Kapelvej 44, 2300 København N Wed Dec 11, 15:00 - 16:00	Kapelvej 44, 2300 København N Thu Dec 12, 16:00 - 17:00	Kapelvej 44, 2300 København N Fri Dec 13, 16:00 - 17:00
--	--	---	---	---

Show More

Info before you book

Bring your own bags - one for fruit & veggie and one for bread.

PLEASE ONLY BOOK IF YOU KNOW YOU CAN COME! IF YOU DON'T COME, YOUR BOX GOES TO WASTE!

You will get a box of 7-10kg. of fruits, veggies, bread and other good stuff. The boxes are not sealed, so we include a fruit juice, cleaned by supermarkets, so some of it might be overripe, wilted or a bit stale. In order to save the waste, we will keep your booking for 10-15 minutes, after that the food will be given to other people.

If you know you will be late for picking up your box, please contact us in advance via Facebook (Madboks) or Instagram (@madboks_kbh).

Please do not attend our events if you expect no fruits and veggies to be in perfect condition - they are not. We are here to save edible food from going to waste!

I understand and I will try to save as much food as possible from going to waste!

How many boxes would you like to book? 1

Choose time for pickup:

16:00 - 16:10 16:10 - 16:20 16:30 - 16:40 16:40 - 16:50 16:50 - 17:00

Figure 5.17: Reservation page with selected event

Contact info

First Name <input type="text" value="Sofia"/>	Last Name <input type="text" value="Gran"/>	Background: <input type="checkbox"/> Student <input checked="" type="checkbox"/> Employed <input type="checkbox"/> Unemployed <input type="checkbox"/> Refugee background
Email <input type="text" value="developer.sofiagran@gmail.com"/>	Phone Number <input type="text" value="Phone number"/>	<input type="checkbox"/> Single parent <input type="checkbox"/> Pensionist <input type="checkbox"/> Low income family <input type="checkbox"/> Other
Comments <input type="text" value="Write your comments here"/>		

Are you interested in volunteering with Madboks in the future?

Figure 5.18: Reservation page

The backend code for reserving is presented in subsection 5.5.12.

5.5.8 S3F7: Location page for admin

There is now a location page for admin, where they can create new locations and see/edit/delete current locations. It is still possible to do this from the dashboard as well, however, using the locations page can be more effective. The location page can be seen in figure 5.19.

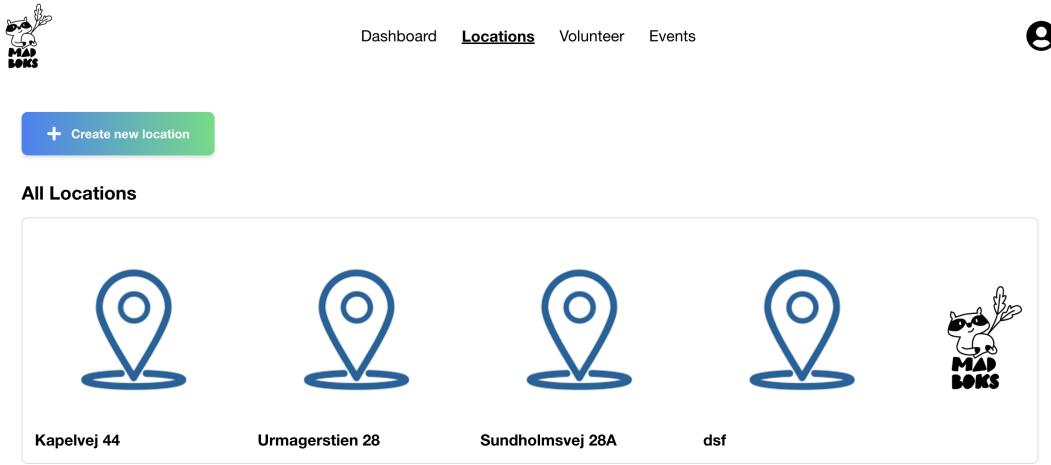


Figure 5.19: Location page

5.5.9 Backend items

- S3B1: Cloudflare Turnstile Captcha
- S3B2: Setup of pre-production server
- S3B3: Timeslots/Locations/Events/Reservations makeover and fix
- S3B4: Email updates (send multiple emails at once, add personal information to formatting using Handlebars)
- S3B5: Docker Setup

5.5.10 S3B1: Cloudflare Turnstile Captcha

From the frontend, the Turnstile issues a challenge, that must be passed, to receive a token. The challenge token is sent to the backend to verify the user with an API call to Cloudflare which sends a response of either success or failure back to the frontend.

5.5.11 S3B3: Setup of pre-production server

A pre-production server is set up so that changes can be tested on a server before being merged into production. This is created to mitigate hotfixes on the production server, as the website and backend may act differently when deployed to the server, contra being hosted/tested locally.

5.5.12 S3B4: Timeslots/Locations/Events/Reservations makeover

To make a reservation of timeslot work as the PO wanted, with a maximum number of spots available per timeslot, there has to be quite a makeover in the backend and the database. To ensure that timeslots were updated correctly, they became their own table in the database, instead of being a string array in the event table. All classes still use the CRUD methods presented in sprint 1 5.3.3.1, but the business logic has been modified to work with the newly created timeslot table.

Timeslots

Instead of timeslots being an property of events, it became its own table. In the timeslot table, there is a foreign key 'event_id', which refers to the event table, and if an event is deleted, all timeslots that refer to that event are deleted as well. In addition, the table has a column 'slot' which is a string that represents the timeslot (this is auto-generated in the frontend, based on the event start/end time), and 'available_spots' which is a number that represents how many available spots is left for that timeslot.

The constructor for timeslots is as follows:

```
1 id?: string | null;
2 created_at?: Date | null;
3 modified_at?: Date | null;
4 event_id: string;
5 slot: string;
6 available_spots: number;
```

In the database, the fields 'id' and 'event_id' is converted to type GUID.

Events

The event table needed an update as well. To make the generation of timeslots easier, the event had to have columns start_time and end_time of datatype 'time', instead of just a column 'time' of datatype string. Then the array called 'Timeslots' is removed, since it is no longer needed, when timeslot has a foreign key reference to the event table.

The constructor for events is as follows:

```
1 id?: string | null;
2 created_at?: Date | null;
3 modified_at?: Date | null;
4 title: string;
5 description: string;
6 location_id: string;
7 address: string;
8 type: string;
9 number_of_boxes: number;
10 date: Date;
```

```
11 start_time: string;
12 end_time: string;
13 isPublished?: boolean;
14 isCancelled?: boolean;
15 scheduled_time?: Date;
```

In the database, the fields 'start_time' and 'end_time' are converted to type 'Time', and 'id' and 'location_id' are converted to type GUID.

Location

Since the location table works as a template for event creation, the location table has to have the same structure as events. Therefore, the column 'default_opening_hours' of the datatype string, was replaced by columns 'default_start_time' and 'default_end_time' of datatype 'time'. Also, the 'default_timeslots' column was no longer needed, as these are generated automatically based on start/end time in event creation.

The constructor for locations is as follows:

```
1 id?: string | null;
2 created_at?: Date | null;
3 modified_at?: Date | null;
4 city: string;
5 zip_code: string;
6 country: string;
7 street_address: string;
8 apartment_etc?: string;
9 default_number_of_boxes?: number;
10 default_description?: string;
11 default_start_time?: string;
12 default_end_time?: string;
```

In the database, the fields 'default_start_time' and 'default_end_time' are converted to type 'Time', and 'id' is converted to type GUID.

Reservations

When making a reservation, the customer reserves a specific timeslot, therefore, a foreign key 'timeslot_id' that refers to the timeslot table, had to be added.

The constructor for locations is as follows:

```
1 id?: string | null;
2 created_at?: Date | null;
3 modified_at?: Date | null;
4 user_id: string;
5 event_id: string;
```

```
6 timeslot_id: string;
7 donation_amount: number;
8 number_of_boxes: number;
9 comment?: string;
```

In the database, the fields 'id', 'event_id', 'location_id' and 'user_id' are converted to type GUID.

Method: Book box

In reservationRoutes.ts: A POST request to the reservation endpoint calls the reservationController.createReservation method seen in Appendix H.0.6.1.

The method 'createReservation' in reservationController.ts, seen in Appendix H.0.6.2, calls the reservationService.createReservation to handle business logic.

Method 'createReservation' in reservationService.ts, seen in Appendix H.0.6.3, validate the reservation data. It attempts to decrement the available_spots for the specified timeslot using the bookTimeslot method. If timeslot booking succeeds, it creates the reservation in the database. If it fails, it rolls back the timeslot booking.

The method 'bookTimeslot' in timeslotService.ts, calls the stored procedure 'decrement_available_spots' to decrease the available spots for a timeslot in the database. If no spots are available, an exception is raised.

```
1 public async bookTimeslot(id: string): Promise<Result<Timeslot, BaseError>> {
2     const { error: rpcError } = await this.supabase.rpc(
3         'decrement_available_spots',
4         { p_id: id },
5     );
6
7     if (rpcError) {
8         return ResultFactory.Err(
9             this.handleSupabaseError(rpcError, 'decrementTimeslot', { id }),
10        );
11    }
12
13    const { data, error } = await this.findById(id);
14    if (error)
15        return ResultFactory.Err(
16            this.handleSupabaseError(error, 'findTimeslotById', {
17                id,
18            }),
19        );
20    return this.validateAndMapRecord(data!);
21 }
```

The stored procedure 'decrement_available_spots' updates the database, ensuring the available spots count reflects the booking.

```
1 CREATE OR REPLACE FUNCTION decrement_available_spots(p_id UUID)
2 RETURNS void AS $$ 
3 BEGIN
4     UPDATE timeslots
5     SET available_spots = available_spots - 1
6     WHERE timeslots.id = p_id AND available_spots > 0;
7
8     IF NOT FOUND THEN
9         RAISE EXCEPTION 'Timeslot not available or invalid ID';
10    END IF;
11
12 END;
13 $$ LANGUAGE plpgsql;
```

Method: Cancel reservation

The method 'cancelTimeslot' in timeslotService.ts, calls the stored procedure increment_available_spots to increase the available spots for a timeslot when a reservation is cancelled.

```
1 public async cancelTimeslot(
2     id: string,
3 ): Promise<Result<Timeslot, BaseError>> {
4     const { error } = await this.supabase.rpc('increment_available_spots', {
5         p_id: id,
6     });
7
8     if (error) {
9         return ResultFactory.Err(
10             this.handleSupabaseError(error, 'incrementTimeslot', { id }),
11         );
12     }
13     return ResultFactory.Ok(undefined);
14 }
```

The stored procedure 'increment_available_spots' updates the database, ensuring the available spots count reflects the cancellation.

```
1 CREATE OR REPLACE FUNCTION increment_available_spots(p_id UUID)
2 RETURNS void AS $$ 
3 BEGIN
4     UPDATE timeslots
5     SET available_spots = available_spots + 1
```

```

6   WHERE timeslots.id = p_id;
7
8   IF NOT FOUND THEN
9     RAISE EXCEPTION 'Timeslot not found for ID %', p_id;
10  END IF;
11
12 END;
$$ LANGUAGE plpgsql;

```

Method: Edit timeslot for reservation

The method 'updateReservationTimeslot' in reservationService.ts, seen in Appendix H.0.6.4, fetches the current reservation details. It calls the method 'cancelTimeslot' to increment available_spots for the former timeslot. It then calls the 'bookTimeslot' to decrement available_spots for the new timeslot. Then it updates the reservation record with the new timeslot's id. If updating the new timeslot or reservation fails, it rolls back changes to the previous timeslot, ensuring database consistency.

Method: Update active event

The method 'updateActiveEvent' in eventService.ts, fetches the original event details for roll-back if needed seen in Appendix H.0.6.5. Then it updates event details (description and number_of_boxes) in the database. It calls updateTimeslotsAvailability to adjust the available_spots for associated timeslots. If timeslot update fails, it rolls back, to maintain consistency.

The method 'updateTimeslotsAvailability', in timeslotService.ts, seen in Appendix H.0.6.6, first validates each timeslot record. Then it updates the available_spots field for each timeslot in the database. It returns the updated records if all operations succeed or an error if any update fails.

5.5.13 S3B5: Email updates (send multiple emails at once, add personal information to formatting using Handlebars)

The backend of the email service was updated to be able to send emails in bulk and to send multiple different emails with the same request. In its current implementation, an email is sent to the admins with all of the relevant information about a volunteer signup, and a new and unique email is sent to the volunteer confirming their sign-up.

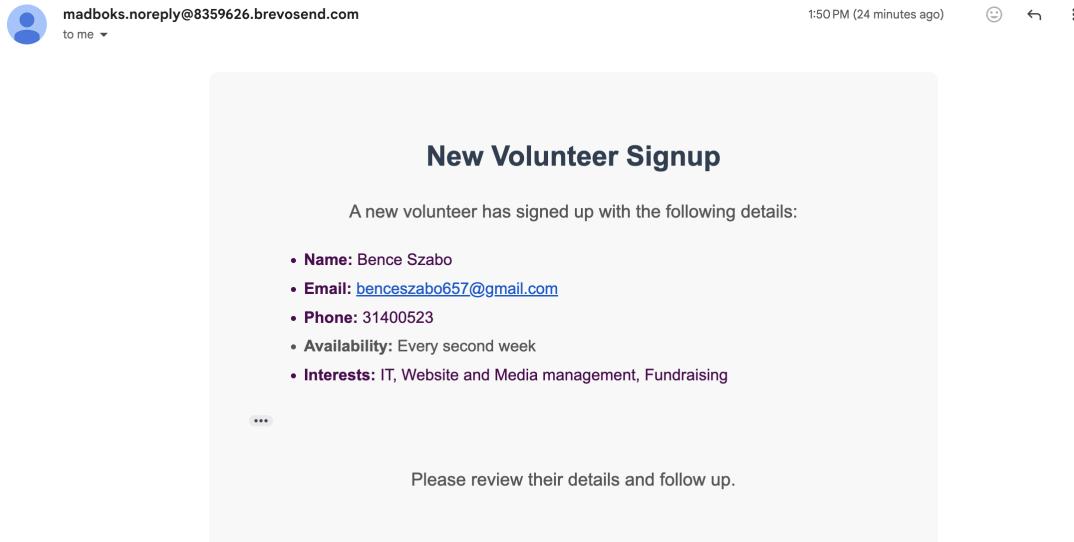


Figure 5.20: Variables in emails

To enable the input of variables into an email rather than sending a static message, the Handlebars library was used, and the following helper function was written for it.

```
1 private compileTemplate(
2     templatePath: string,
3     variables: Record<string, string | number | boolean>,
4 ): string {
5     try {
6         const templateContent = fs.readFileSync(
7             path.resolve(__dirname, templatePath),
8             'utf8',
9         );
10        const compiledTemplate = handlebars.compile(templateContent);
11        return compiledTemplate(variables);
12    } catch (error) {
13        console.error('Error compiling email template:', error);
14        return '';
15    }
16 }
```

This `compileTemplate` function reads a Handlebars template file from the given `templatePath`, compiles it using the Handlebars templating engine, and returns the rendered string with the

provided variables. If an error occurs during file reading or template compilation, it logs the error and returns an empty string.

5.5.14 S3B6: Docker Setup

The docker setup is introduced to create images of the application, so it can be run in a container for the developers or sent to the server to be deployed. This will remove complexity from the GitHub deploy file and make sure the application is stable regardless of the local machine.

5.5.15 Sprint Review

The website was at the end of the sprint in an acceptable state to conduct admin and user tests. The structure of the user tests is described in the chapter 'Quality Assurance' 6.2. The results from these arguably provide the most meaningful feedback up until now.

5.5.15.1 User acceptance test

Results and Insights from Admin tests:

The overall feedback on the product was very positive. Both the product owner and the other Madboks admins were impressed with how far the project had come within the limited time frame. They expressed overall satisfaction with the product and its design. They emphasised its efficiency and an intuitive, easy-to-use and easy-to-learn interface. With that being said, it was discovered some bugs and some business acceptance criteria that were not fully met, as well as ideas for future work on the product.

Discovered bugs:

- The create event form disappears on larger screens.
- The source code is revealed using the debug inspector.
- You are automatically logged in as the last person that logged in.
- 'See events' button on the home page does not work.

Business acceptance criteria not met:

- The 'default' information in the event description should be modified.
- Create event/location should have better validation and error handling.
- There should be an info message when trying to book more than one box.
- Sign-up as a volunteer should open in a new tab
- Links to socials should be open in a new tab.
- Should be able to book for others - by changing the 'receiver' of the reservation in the contact form.

- When booking an event, it should be removed from 'upcoming' and only display in 'your bookings'
- Add validation to avoid that two events with the same address can be created on the same day and time.

Future work:

- The volunteer sign-up should be linked to the already existing 'welcome' mail.
- Implement waiting lists on fully booked events.
- Make social media references in the description work as links.
- Implement the possibility to schedule an event to be published on a specific date and time.
- Implement the possibility for reoccurring food distribution events to be auto-created one month prior.

Results and Insights from Volunteer/Customer test:

The tester expressed overall satisfaction with the product and its design. He explained that the system was easy and efficient to use, and expressed enthusiasm related to some of the features, such as the ability to edit/cancel your booking. He believed that this would be very helpful for both the customers, but also the volunteers. The test also discovered some new bugs and notes for future work.

Discovered bugs:

- Wrong error message: 'unexpected error' if you try to log in without confirming your email first.
- Does not save phone number when signing up

Future work:

- Having 'volunteer pages', the same way as there are 'admin pages'. Could be used by volunteers and admins to handle scheduling etc.
- Sending notification email on the day or the day before a customer has a booked event.
- Better formatting of the event description, so it is more readable.
- The Reservation page could benefit from being step-by-step with auto navigate to the next step (displays less information at the same time).

5.6 Sprint 4

5.6.1 Sprint Planning

The main goal of this final sprint is to implement some of the easier low-risk items based on user feedback gathered during user tests and to perform an overall clean-up of the product.

5.6.2 Frontend items

- S4F1: User test fixes and small text and layout updates H.0.7
- S4F2: Configuration fix to not expose source code in the browser H.0.8
- S4F3: More mobile compatibility
- S4F4: Different receiver of email than logged-in user

5.6.3 S4F3: More mobile compatibility

When testing the app on actual phones, it was discovered that not all screens were not scaling properly on phones. The reason for this is that while testing mobile compatibility, the screen it was tested on was 500px wide, while phones can be down to 320px wide. Therefore, there were made some changes to ensure the website is accessible for different mobile screens.

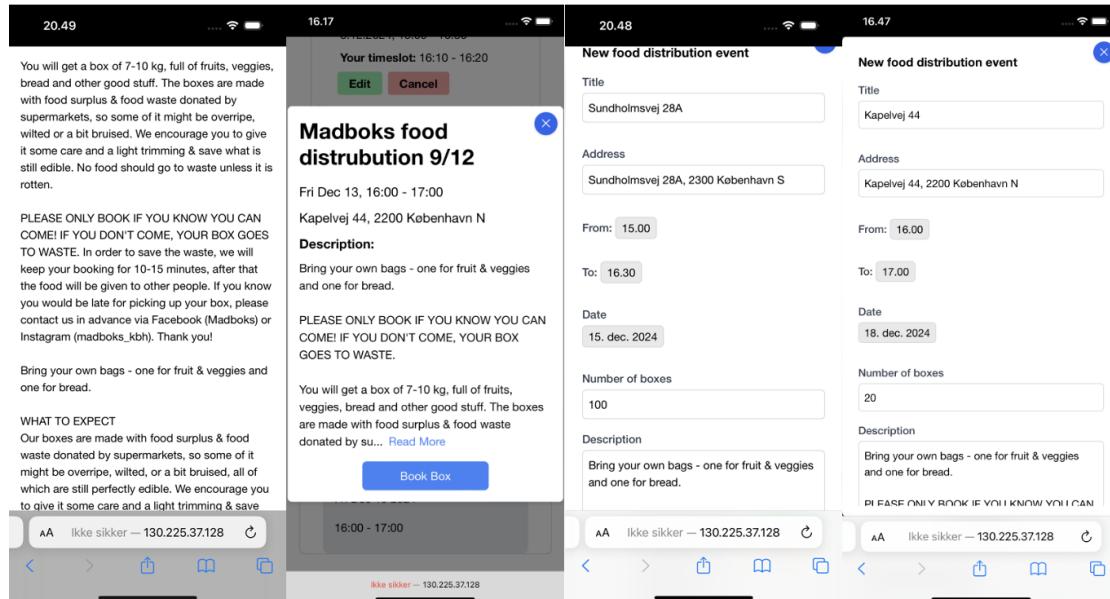


Figure 5.21: Screens explained from left to right: 1) Event info pop-up before changes. 2) Event info pop-up after changes. 3) Create event pop-up before changes. 4) Create event after changes.

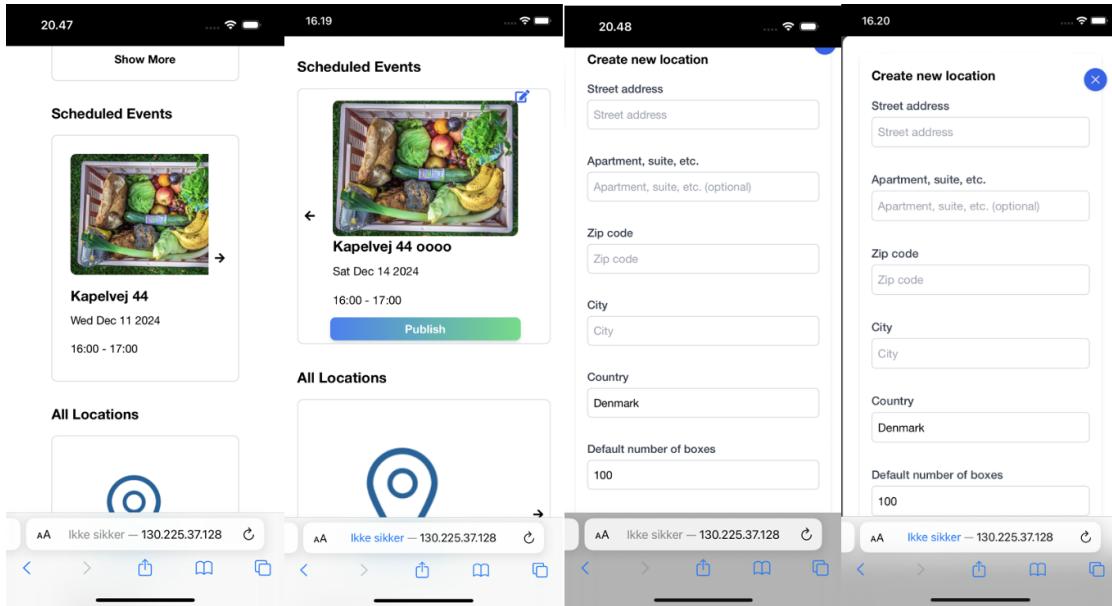


Figure 5.22: Screens explained from left to right: 1) Admin upcoming event display before changes. 2) Admin upcoming event display after changes. 3) Create location pop-up before changes. 4) Create location after changes.

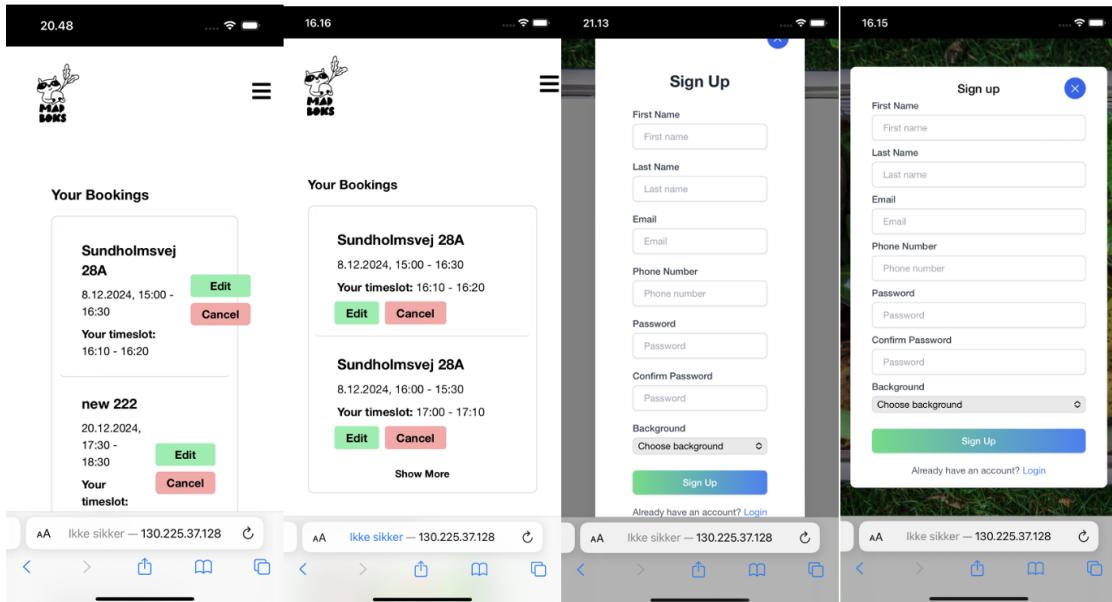


Figure 5.23: Screens explained from left to right: 1) Your bookings display before changes. 2) Your bookings display after changes. 3) Sign-up pop-up before changes. 4) Sign-up location after changes.

5.6.4 S4F4: Email updates

The email template was updated so that it also takes the name of the customer and the booked timeslot as a parameter. These parameters are then used in the email template to create personalised email confirmations see figure 5.24.

In addition, as wanted by the product owner in the user test feedback, the email is sent out to the email that is used in the contact form (also if it differs from the email of the logged-in user), so it is possible to book for other people (e.g. if someone is booking for their parents that are not very technical).

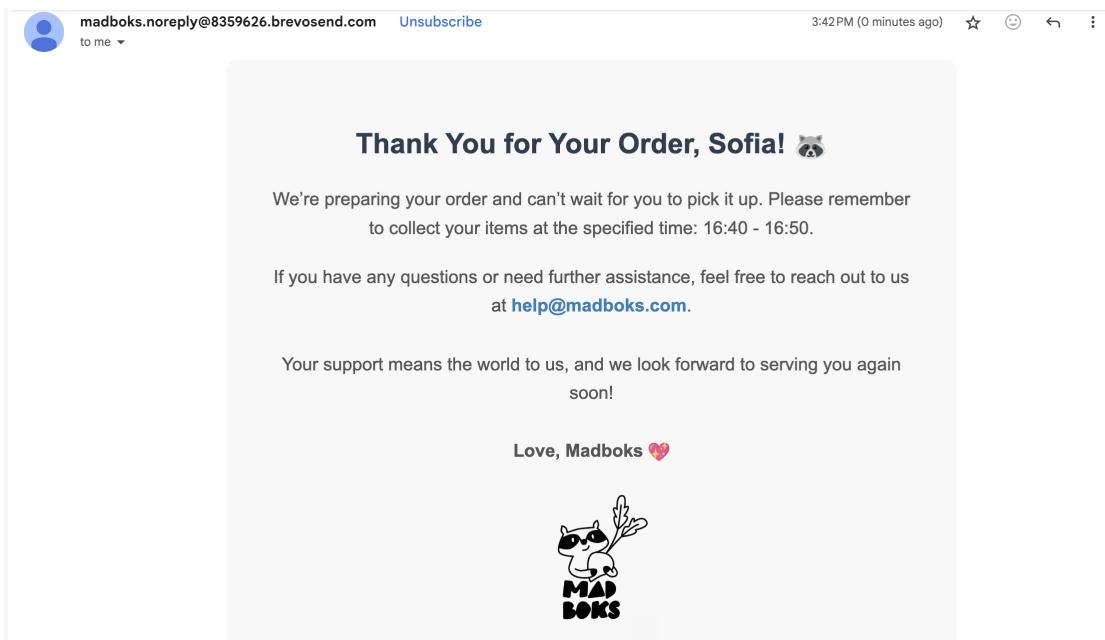


Figure 5.24: New confirmation email

5.6.5 Sprint Review

The primary goal of this sprint was to implement some low-risk items mostly based on feedback from user tests. This was successfully done, and a final meeting was held with the product owner to give a rundown on the final iteration of the product. Other than that, a general clean-up of the code was done.

Chapter 6

Quality Assurance

To ensure acceptable code quality, a pre-planned quality assurance process was followed. Unit tests were used to validate individual components, structured pull requests and a review process had to be followed upon code merges, and user testing was applied to gather feedback on the overall user experience. The main goal was to identify potential issues early in the development cycle and to avoid pushing faulty code by running tests and linting in GitHub actions.

6.1 Unit tests

Unit testing is a fundamental part of the quality assurance process, focusing on the verification of individual components or functions of the application. The tests are designed to ensure that each unit of code performs as expected in isolation, without dependencies on other parts of the system. Unit tests were implemented in this project to validate the core functionalities of key modules, ensuring the robustness and reliability of the underlying logic. The tests were automated in a CI pipeline using GitHub Actions.

In the backend, unit tests were created for all endpoints related to the database tables: events, locations, timeslots, reservations and auth.

6.2 User Tests

A key focus of the project was enhancing the user experience. To achieve this, regular acceptance testing with the product owner and user testing with individuals unfamiliar with the product were essential steps.

Three semi-structured user tests with people from the Madboks organisation were conducted. The user tests were conducted to identify the usability of the website and identify bugs and

missing functionality. It is important to note that usability is not a single property of a user interface but has many different components [10].

Jakob Nielsen, Danish computer scientist and UX expert, describes in his book "Usability Engineering", that he associates the definition of usability with these five usability attributes [10].

- **Learnability:** The system should be easy to learn so that the user can rapidly start getting some work done with the system.
- **Efficiency:** The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.
- **Memorability:** The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
- **Errors:** The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.
- **Satisfaction:** The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.

The usability attributes defined by Nielsen [10], were used in the user tests for evaluation.

On the 4th of December, the Product Owner, a Madboks admin and a Madboks volunteer conducted the user test. This type of user test was a great tool for discovering bugs, missing functionality and ideas for future work. Involving the product owner in the testing is essential, as the product owner is the one who has the final say in how the platform should look and what functionalities it should provide, from the business perspective. This criteria is defined in this project as business acceptance criteria. Therefore, the tests also ensured that the product meets the business acceptance criteria.

6.2.0.1 Product owner and admin tests

The tests by the Product owner and Madboks admin focused on evaluating the website's ability to meet the organisation's needs. The objective was to ensure the system effectively supports efficient event management and reservation tracking. In addition, it ensured that the customer side of the website met all business acceptance criteria.

Test methodology:

1. **Scenario Setup:** The admin was asked to perform tasks related to Madboks' operations:
 - Log in
 - Creating, modifying, and deleting events.
 - Creating, modifying, and deleting locations.

- Managing active events
 - Exploring the rest of the website to ensure it meets the business acceptance criteria
2. **Observation:** During the test, the present developers observed user interactions to identify potential challenges, confusion, or areas requiring improvement. The testers also commented while performing the test.
 3. **Feedback Collection:** After the tasks, the tester participated in a semi-structured interview to provide feedback on the system's utility, efficiency, and areas for potential enhancement. The testers got the opportunity to voice their thoughts and findings, followed by questions and clarifications by the developers.

6.2.0.2 Volunteer/customer tests

The tests by the Madboks volunteer focused on evaluating the website's ability to meet the needs from both a customer's perspective and from the perspective of someone who is contributing to the management of the events (as a volunteer). The objective was to ensure the system effectively supports the booking of boxes and cancellation/editing of their booking. In addition, the user interface was evaluated, such as the log-in/sign-in process, the home page and the events page.

Test methodology:

1. **Scenario Setup:** The volunteer was asked to conduct tasks related to the customer's behaviour:
 - Inspect the home page
 - Create a reservation without a user
 - Sign up
 - Create a reservation with a user
 - Edit your reservation
 - Sign up as a volunteer
 - Explore the website
2. **Observation:** During the test, the present developers observed user interactions to identify potential challenges, confusion, or areas requiring improvement. The testers also commented while performing the test.
3. **Feedback Collection:** After the tasks, the tester participated in a semi-structured interview to provide feedback on the system's utility, efficiency, and areas for potential enhancement. The testers got the opportunity to voice their thoughts and findings, followed by questions and clarifications by the developers.

The user tests conducted with the Product Owner, the Madboks admin, and the volunteer proved invaluable in refining the platform and aligning it with both business and user expectations. These tests identified critical usability issues, bugs, and missing functionalities early, allowing the team to address them before the final deployment. Moreover, the active involvement of the product owner ensured the platform met the business acceptance criteria, while the volunteer tests provided insights into real-world user scenarios and highlighted areas for improving the customer experience.

Chapter 7

Overview of the final product

The end product of this project is essentially a website for creating and managing events on the admin side, and booking boxes and managing bookings on the customer side - supported by features such as live previews, template creation, and keeping track of relevant information in real-time. The website fetches event and customer data from a Supabase database sends emails using a third-party SMTP server, and ensures security with Cloudflare's Captcha turnstile. The following is a brief breakdown of the implemented features of the end product. Below this is a walk-through with more elaboration.

- Homepage redirecting customers to reservation, volunteer signup and communicating Madboks' main mission goals
- Volunteer signup page hooked up to email service
- Reservation page hooked up to a database and email service
- About us page
- Personal profiles for easier booking and overview for bookings
- Cancellation and booking edits
- Location template management for admins
- Event creation and management for admins
- Event overview and edits for admins
- Frictionless CAPTCHA (Cloudflare Turnstile; no need for manual puzzle solving as it runs in the background) to avoid form/login spam and bots
- Mobile compatibility (responsiveness)
- Supabase database storing all user and event data

The website's key functionalities on the customer side include booking food boxes, creating accounts to manage or cancel said bookings, discovering future events and their details, and having the opportunity to participate in volunteering and learn about Madboks and its mission goals. The website is usable with or without customer logins.

On the admin side, events can be created/edited/deleted with the help of a live preview and template selector, said location templates can be created/edited/deleted to streamline the event creation process, and relevant information regarding events such as number of boxes left etc. is available as a part on an admin dashboard. During the event creation and editing process, several Madboks-specific solutions support their unique processes, such as the automatic distribution of boxes across collection timeslots and a donation selector to provide an estimate of potential earnings etc.

On the frontend side, the website is highly responsive, being fully compatible with both computer and phone screens. Implementing unique Tailwind styling for diverse screen sizes, the development team ensured that all components and pages were rendered correctly on all platforms. For instance, the top bar becomes a burger menu, and the home screen displays images and text vertically while keeping its snappy behaviour between sections.

To account for scalability in the website's email service, Brevo's SMTP server was selected. Its flexible pricing allows Madboks to only pay for what the organization needs, and its quality service ensures that emails consistently reach recipients' inboxes on time without being flagged as spam.

Cloudflare Captcha further supports the potential scalability of the product - Turnstile is engineered to be lightweight, ensuring minimal impact on page load times. It typically adds less than 100 milliseconds to loading, significantly less than Google reCAPTCHA, which can add 200-500 milliseconds due to its heavier scripts [1]. Even more importantly, Turnstile does not require solving puzzles - it analyses behavioural signals instead, ensuring zero user friction. The quality and reliability of Cloudflare's Turnstile Captcha in the context of websites with massive user bases is further confirmed by its popularity, for instance, OpenAI, Shopify, and GitLab [18].

The database selected for this project, Supabase, is a scalable database solution because it's built on PostgreSQL, a database solution that supports large datasets and concurrent queries. It offers serverless APIs to handle high traffic efficiently. Supabase also includes real-time data updates, storage for structured data, and access control with Row-Level Security. Its global deployment options ensure low latency, while its pay-as-you-go model makes it cost-effective for scaling. Additionally, it supports extensibility with PostgreSQL functions and integrations for continuous improvement [14].

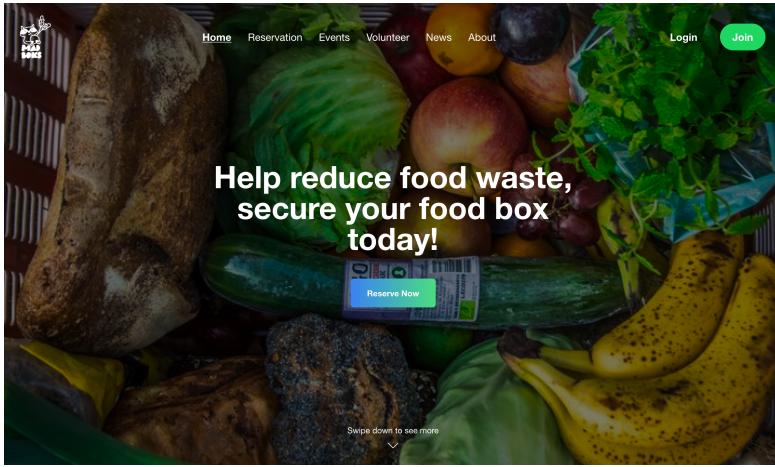


Figure 7.1: Homepage 1

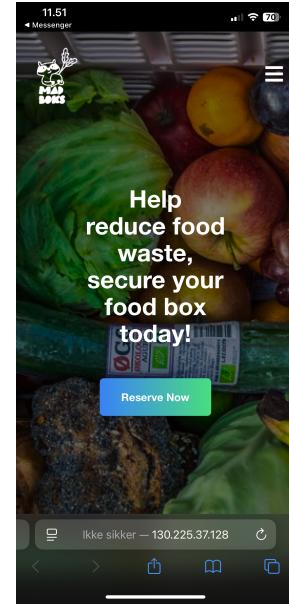


Figure 7.2: Homepage 1
mobile

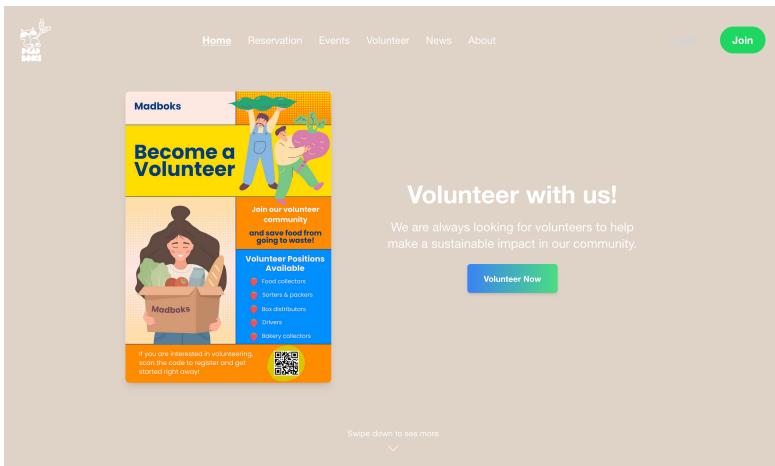


Figure 7.3: Homepage 1



Figure 7.4: Homepage 2
mobile

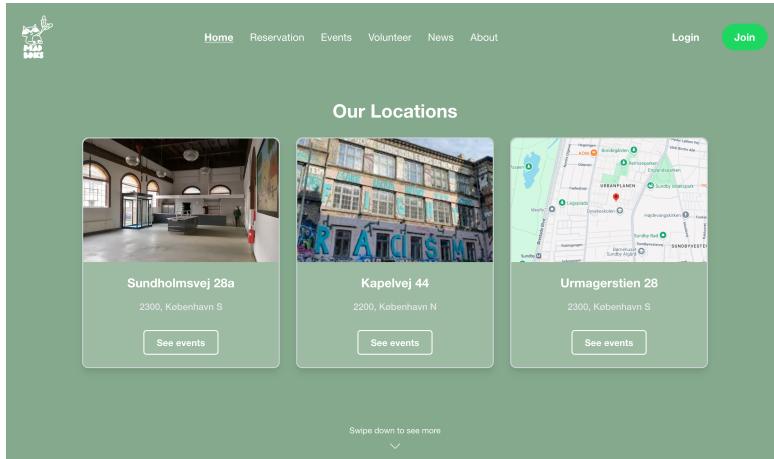


Figure 7.5: Homepage 3

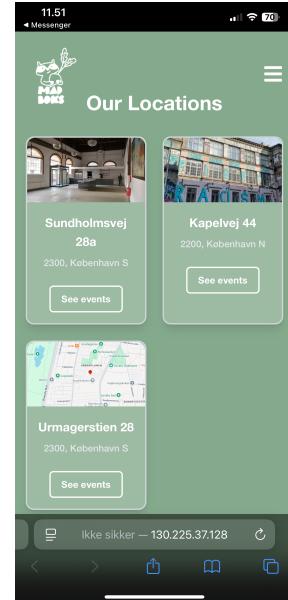


Figure 7.6: Homepage 3
mobile

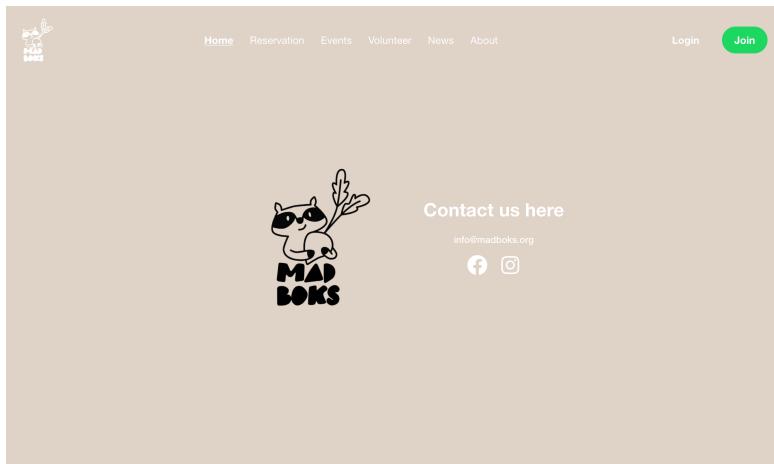


Figure 7.7: Homepage 4

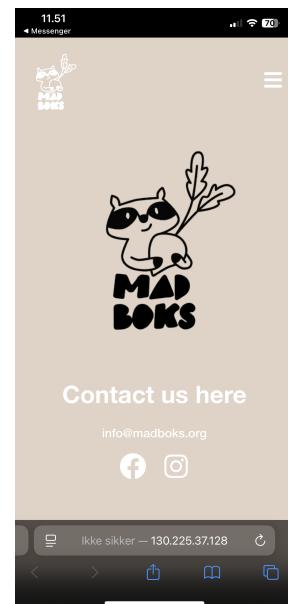


Figure 7.8: Homepage 4
mobile

Figure 7.9: Reservation page

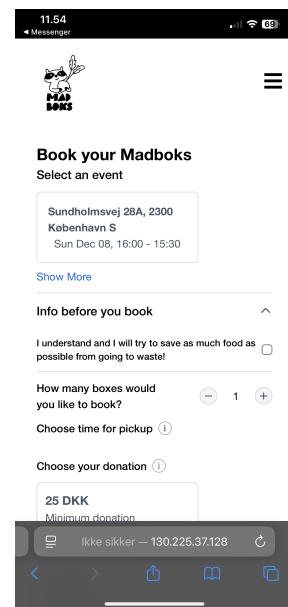


Figure 7.10: Reservation page mobile

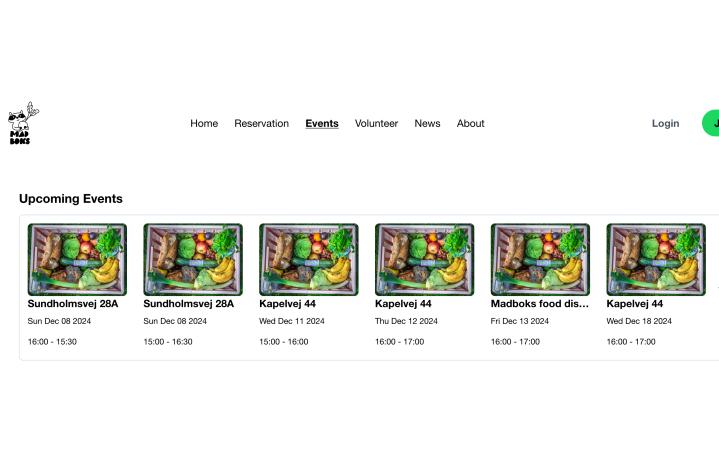


Figure 7.11: Events page

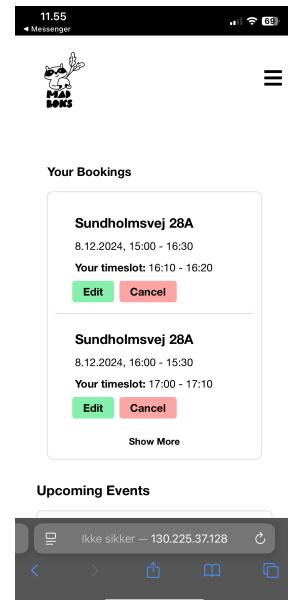


Figure 7.12: Events page mobile

Figure 7.13: Events admin page

Figure 7.14: Events admin page mobile

Chapter 8

Discussion

With the completion of the Madboks system, there is still room to refine and expand the system's functionalities to further support the organisation's goals. This discussion reflects on the development process, and areas for future improvements, including pending PBIs and security considerations and discusses the system's scalability and sustainability by reducing resource waste.

8.1 Process

8.1.1 Reflections on the development process

The Agile-adjacent workflow of this project worked well for supporting transparency and continuous value delivery. The bi-weekly sprint review meetings with the product owner served as a good place to discuss where the project was headed at that specific point in time and to ensure a good common understanding across the roles. Other than that, having to present an increment at the end of each sprint naturally applied some pressure on the developers to keep working to avoid disappointment. Working face-to-face with the product owner also put more emphasis on the importance of taking feedback, translating feedback into usable user stories or PBIs, and implementing the resulting items.

The product owner of this project, Roxana Zlate, showed a lot of engagement throughout the entire project period and did a great job of voicing stakeholder concerns and representing user needs. The latter, for instance, is confirmed by the user test conducted at the end of sprint 3.5.5.15.1. Feedback was clear and concise, and transparency was kept from both the developer and product owner side with the help of frequent communication.

The development team's workflow and the generated value were acceptable, but in hindsight, there are a couple of things the team would have done differently. First, it would have made

sense to give everyone time to familiarise themselves with the setup of different components rather than a single member taking responsibility for it. The lack of transparency regarding this part of the development operations resulted in items being "delivered" but not guaranteed to be working in production and also made it difficult for other team members to help with the production issues.

Continuing on internal processes, laying plans out for solving challenges and/or engaging in pair programming when addressing larger items could have aided in a more well-rounded team with a general understanding of a larger chunk of the product rather than every member specialising in their specific product backlog items. On the other hand, this could have slowed down the delivery process.

Github's Projects and Issues tools were utilised according to plans - although spending some more time documenting issues could have helped in the documentation and report-writing following sprints. Overall, Github as a version-control tool and its extra features were good at aiding the team's feature-branch structure.

8.2 The product owner's reflections on the development process

The feedback from the product owner, Roxana Zlate, has been overwhelmingly positive, reflecting satisfaction with the project's progress and outcomes. Overall, they expressed that they were very happy with the work done and the direction of the project. The frequency of meetings was appreciated, with the product owner noting that the current schedule was effective. However, they suggested that increased online communication and collaboration between meetings could further optimise the process by addressing issues as they arise.

The meeting format was praised as being well-structured and productive. Zlate also highlighted that her input was consistently considered and effectively incorporated into the project. This level of involvement contributed to her satisfaction with the outcome, which she described as excellent. The full written feedback can be found in Appendix F.

8.3 How QA could have been improved

A major point of concern for the developers at the end of the project was the lack of sufficient testing required for a scalable web application. The most crucial improvements were recognised to be the lack of integration testing, frontend unit tests, load tests and static code analysis.

8.3.1 Integration testing

Arguably the most important missing element of this project is integration testing. Integration testing evaluates the interactions between multiple components to verify that they work together as intended. Unlike unit tests, which assess isolated functions, integration tests examine how well different modules or systems communicate and operate as a cohesive whole. This is particularly important for identifying issues related to data flow, API calls, or dependencies between modules. In this project, integration tests could have been conducted to, for instance, validate communication between the frontend, backend, and database, ensuring that all layers of the system work as intended. These tests would have been important to confirm the stability and reliability of the application in its entirety.

8.3.2 Load tests

Load tests are a type of performance testing used to evaluate how a system behaves under a specific workload. They simulate a variety of user activities, such as making requests, submitting forms, or performing database queries, to measure the system's responsiveness, stability, and scalability. The goal is to determine how well the system handles expected and peak traffic levels, identifying bottlenecks and potential failures before they impact real users [11].

Although Madboks is not expected to undergo a massive increase in the number of users anytime soon, a couple of hundred visits a day is still something that the current system may not be able to handle gracefully.

Although there is no data that supports this, it is safe to assume that Madboks' number of daily visitors is not evenly distributed - high traffic is a lot more likely after publishing events. If registered users are notified about new events, it is almost guaranteed.

Submitting forms, sending emails, and API requests are central to the system, so conducting load tests would make perfect sense to find out how many concurrent visitors or requests the website can handle before crashing under the load. Also, a load test would be useful to uncover bottlenecks.

One such tool that seems like a reasonable choice for this purpose is k6, a load-testing tool designed for APIs and microservices [7]. According to its documentation, its integration in a CI/CD pipeline is also supported, and it integrates well with Fastify RESTful endpoints.

8.3.3 Unit tests on the frontend side

As for unit tests, although not including a lot of complicated logic and error-prone components, it would have been good practice to write tests for the frontend as well. Specifically, testing Axios would ensure that the frontend is correctly communicating with your backend or third-party APIs.

8.3.4 Static code analysis (CodeScene)

For improved quality assurance and evaluation, the team could have utilised the CodeScene static code analysis tool throughout the project to continuously evaluate code health and recognise problematic areas [4]. Although CodeScene is very simple, checking for overly nested code, dead code and repetition, it would have been good practice to ensure an extra check for code health, and to be able to put a metric on code quality. CodeScene also has an extension to provide warnings during the development process which could have been useful.

8.4 Future work

As of the writing of this report, there are still some known issues that should receive first priority to fix if the team had another sprint to work on the code. Namely, these are resolving issues with the authentication service, and deploying the already partially implemented form validation.

The authentication issue arises because the application saves a session when a user logs in and retrieves the same session upon revisiting the site, without verifying that the session belongs to the user. This happens due to a lack of authentication using site cookies with access and refresh tokens. To resolve this, the application should use the access tokens stored in these cookies to verify the user's identity before retrieving the session, enabling automatic and secure login.

Proper form validation for admins, should be in place to minimise the risk of unwanted behaviour. In addition, implementing input validation of all fields accessible on the website is an important measure to protect against input that could overload the server, or send bogus requests and malicious code. Also, having a Privacy Policy that the users have to agree on, before creating an account and making a booking, is important to implement before the website is ready to be used by real customers.

With the approximate 3-month window of the project, the development team needed to prioritise features to be implemented. The following list encapsulates features and enhancements that could be interesting for future work.

- Allow users without accounts to cancel or edit bookings through a link
- Implement a waiting list for sold-out events
- Enable scheduling of events and auto-create events one month in advance
- Connect the volunteer sign-up page with the current email service for volunteers
- Add validation to prevent creating two events with the same location, date, and time
- Email service enhancement - Send email notifications the day before or on the day of an event

- Email service enhancement - Send cancellation emails if an admin cancels an event
- Fully implement the ability to create "other events"
- News forum where admins can post announcements

Cancellation is, in general, a very important thing for the product owner, because if a customer does not cancel, then a foodbox goes to waste, literally counteracting Madboks' entire mission goal. Currently, if a customer is not registered, then can place orders, but have to reach out to Madboks through Facebook to cancel their booking. This is not an ideal solution. A way to tackle this could be to generate a specific link where non-registered users can manage their bookings without explicitly logging in. This could be done with a secret link - generally a pretty popular solution. Links are shared and it could pose potential vulnerabilities, but the overall risk is still pretty low.

The email service enhancements would require some work because they require exploring new tools and architectural patterns. For instance, when having to send emails out a day or two before an event, an event-driven architecture and a scheduler-worker pattern would be something the development team would have to look into. Systems such as this are built around events like "event created," "event updated," or "time to send reminder", triggering messaging systems or other actions. The database for event data is already in place, and Supabase does have a built-in scheduler, CRON job, periodically checking for events happening in the next 24 hours. With the help of a queue managed by said scheduler, the worker should then be able to dequeue and use the already implemented SMTP server to send emails out. Enabling the scheduling of events and auto-create events one month in advance would require very similar architecture and design patterns.

The waiting list is a useful addition that would allow users to sign up for events that are already at full capacity. By creating a waiting list, users have a way of registering their interest for a specific event. If someone cancels their reservation, the next person on the waiting list will be notified and offered the opportunity to reserve the box. This would help ensure that every available spot is filled, preventing food from going to waste. In terms of implementing this, it would require integrating the registration system with a dynamic notification system, that would automatically track cancelled events and notify waiting list users. Arguably, it is also somewhat similar to the previously described scheduler-worker pattern, because it would also have to work with triggers.

8.4.1 Security

To ensure the Madboks platform is ready for full deployment to all users, several additional security measures should be implemented as part of future work:

- **Secure Domain and HTTPS:** Host the website on a secure domain using HTTPS and TLS certificates to encrypt all data in transit, ensuring secure communication between users and the platform.

- **Endpoint Authorisation:** Introduce authorisation mechanisms for all API endpoints to prevent unauthorised access to user data. For instance, only authenticated users should be able to access or modify their own bookings.
- **Input Validation:** Implement input validation across all form fields on the website to protect against malicious inputs, such as SQL injections or payloads designed to overload the server like it is done with email and phone number fields.
- **Role-Based Access Control (RBAC):** Expand the RBAC system to ensure that only authorised roles (e.g., admins) can access sensitive features like event creation, modification, or deletion.
- **Privacy Policy Agreement:** Require users to agree to a Privacy Policy before creating an account or making a booking, ensuring compliance with data protection regulations and increasing user trust.

These measures will significantly enhance the security of the platform, protecting sensitive user information and preventing malicious attacks.

8.4.2 Expanded User Testing

While the system meets the core requirements defined during development, the system could greatly benefit from more extensive user testing. The user tests conducted so far have provided valuable feedback but have been limited in scope, primarily focusing on core functionalities and involving people from the Madboks organisation. The system would benefit from having user tests that involve both recurring and new customers. Recurring customers could offer more insights into improving features they frequently use, and ensuring that no prior functionality is lost in the new system. Involving new customers would give insight into the usability of the website when the user has no prior knowledge of Madboks. It could identify challenges and highlight areas where the platform can be more intuitive for first-time users. Working with more user tests could give more insight into user behaviour and potential bugs, enhancing both usability and user satisfaction and allowing the team to refine features based on detailed user feedback.

In addition, these user tests could benefit from being more structured than the one conducted with the Madboks employees. One way to structure this is to ask the participants the same set of questions after the test, to consistently gain insight into their perception and experience. The questions could be based on the usability attributes defined by Nielsen [10], presented in the Quality Assurance chapter 6.2. The participants could answer each question using a Likert scale ranging from 1 to 5, with 1 being the worst rating (Strongly disagree), 5 being the best rating (Strongly agree), and 3 being the neutral [10]. Using this scale helps to understand the users' perceptions and experiences, and makes it easier to compare and analyse the data collected from the conducted user study. Using questions related to the usability attributes ensures that the application is intuitive and effective and provides a satisfying user experience.

8.5 Scalability

During the design phase of this project, most third-party tools and software architecture/design patterns implemented were selected with scalability in mind. The layered and tiered design pattern chosen promotes the separation of concerns inside the system and enables horizontal scaling independently for each layer when addressing potential bottlenecks — adding more machines/nodes to the system.

One major point of improvement in the system in terms of scalability would be to restructure the backend services so they do not operate using a singleton pattern. A singleton instance means there's only one service handling all requests. If that instance fails, the entire functionality becomes unavailable. Also, a singleton instance has limited capacity to handle concurrent requests. As the load increases, it can quickly become a bottleneck.

As Madboks evolves, its architecture could transition to a micro-services-based approach, where each service (e.g. user authentication, event reservation, notification) runs in its own container. Docker makes it easy to adopt such a structure.

8.6 Sustainability

By reducing the manual administrative tasks, the volunteers can focus their efforts on scaling operations and improving event quality, ultimately amplifying the organisation's impact.

Concerning the SDGs, the project aligns in the following ways:

- SDG 2: Zero Hunger

The project tackles food insecurity by redistributing surplus food efficiently to individuals who need it. By enhancing the logistics of food donation events, we ensure that surplus food reaches people who need it rather than being wasted. With the improved accessibility and user-friendly interface it is easier for individuals to access food resources.

- SDG 11: Sustainable Cities and Communities

The project contributes to building sustainable communities by improving the logistics and accessibility of food donation events. By creating a more efficient digital solution, we can reduce the burden of manual coordination on administrators and volunteers, fostering a more sustainable operation.

The project fosters community cohesion by bringing together donors, volunteers, and people through a centralised platform. It promotes the distribution of food resources to groups who need it, contributing to more sustainable communities.

- SDG 12.3: Global Food Loss and Waste

The project promotes sustainable consumption and production by reducing food waste through redistribution. The system ensures that surplus food isn't wasted, but rather distributed efficiently, reducing unnecessary food waste.

By implementing features such as better management, a more centralized booking system and automated notifications, the platform ensures that food is distributed to those in need without excess wastage.

By addressing these SDGs through an optimised digital solution, the Madboks platform supports its organisational mission and also shows how localised efforts can contribute to achieving global sustainability goals by reducing environmental waste, fostering stronger communities, and ensuring fair access to resources.

Chapter 9

Conclusion

This project addresses a critical social and environmental issue—food waste through the development of a web application for the non-profit organisation Madboks. The collaboration aimed to replace their fragmented, manual processes with a unified booking and event management platform to enhance efficiency, scalability, and user experience.

This report documented every phase of the project, including analysis, design, implementation, testing, evaluation, and the authors' reflection. The project successfully delivered a functional web application that centralises Madboks' operations, focusing on reservations, event management, and communication.

Key developments included replacing manual tools like Google Forms and Excel with automated reservation and event management systems to reduce administrative overhead and inefficiencies, incorporating user-centered design principles inspired by platforms such as Too Good To Go and Facebook to create intuitive interfaces for customers, admins, and volunteers, and implementing somewhat scalable software architecture and DevOps practices—including CI/CD pipelines and scalable hosting solutions—to support Madboks' growing user base and feature set.

Additionally, the project integrated features like Cloudflare Turnstile CAPTCHA and Supabase authentication to ensure data security and accessibility, while aligning the platform's goals with the UN's Sustainable Development Goals (SDGs), particularly those related to zero hunger, responsible consumption, and sustainable communities.

Reflecting on the problem statement — "How can food waste among Danish retail stores be minimised by developing a new localised, web-based platform for Madboks that enables them to create and manage events for distributing near-expiry food items and allows users to conveniently book them?" — it is clear that the project addressed many of its key components. The developed platform provides a solution for event creation and management,

offering users a convenient way to book near-expiry food items. Operational bottlenecks were reduced through automation, and the focus on user-centered design enhanced the platform's accessibility and usability for all stakeholders.

However, the fulfilment of the problem statement's ultimate goal—minimising food waste—is inherently challenging to measure accurately in the short term. While the platform introduces mechanisms to support food distribution, its success in reducing food waste depends on adoption rates, user engagement, and the extent to which Madboks and their partners can consistently prevent surplus food from being discarded. Quantitative evaluation metrics, such as tracking the volume of food saved from waste and the number of users served, require long-term data collection and analysis. Furthermore, external factors such as partnerships with retailers, user behaviour, and logistical constraints also influence the platform's impact, making it difficult to isolate and attribute success solely to the new system.

While the project met many of its core objectives, there is still room for improvement. The Product Owner expressed interest in features such as waiting lists for sold-out events, notifications on the day of events, cancellations without login, support for another event types, news posts, and scheduling or auto-creation of food distribution events. Future work could also include integrating payment systems, advanced volunteer management tools, and mobile application support to further streamline operations and improve user convenience. Implementing analytics to track the platform's impact on food waste reduction and user engagement over time would provide more concrete insights into its effectiveness.

By enhancing operational efficiency with a web application and supporting Madboks' broader vision of sustainability, this solution lays a potential foundation for reducing food waste and promoting responsible consumption. With further development and sustained effort, the platform could evolve into a powerful tool for advancing both organisational goals and societal change.

Appendix A

Screenshots

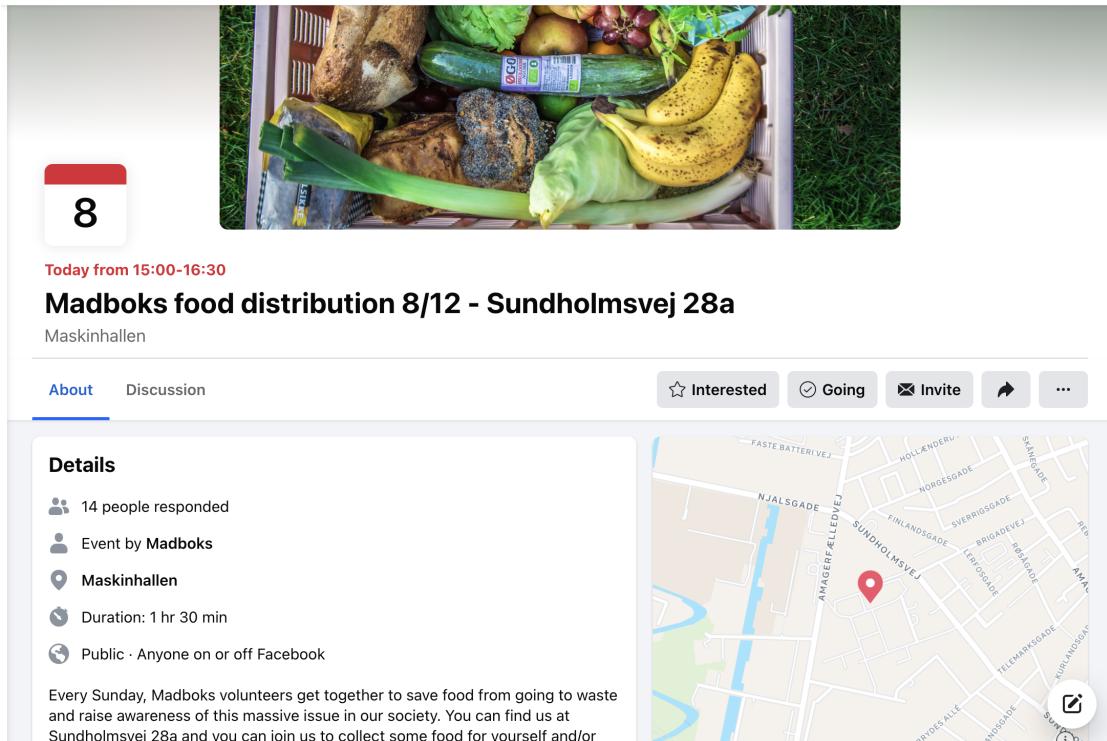


Figure A.1: Layout of how it looks when you click on an Madboks event on Facebook from the computer



Box Reservations for Madboks Sundholmsvej
28a - Food Distribution 8/12

How to reserve a box of food:

1. Fill in this form & choose a time slot for your box pick-up.
2. Check your email for a copy of your responses.

Reservations are considered automatically confirmed once you receive an email with a copy of your responses once you complete filling in this form.

You will get a box of 7-10 kg, full of fruits, veggies, bread and other good stuff. The boxes are made with food surplus & food waste donated by supermarkets, so some of it might be a overripe, wilted or a bit bruised. We encourage you to give it some care and a light trimming & save what is still edible. **No food should go to waste unless is rotten.**

PLEASE ONLY BOOK IF YOU KNOW YOU CAN COME! IF YOU DON'T COME, YOUR BOX GOES TO WASTE. In order to save the waste, we will keep your booking for **10-15 minutes**, after that the food will be given to other people. If you know you would be late for picking up your box, please contact us in advance via Facebook (Madboks) or Instagram (@madboks_kbh). Thank you!

Bring your own bags - one for fruit & veggies and one for bread.

Time & place:

Figure A.2: Layout of how the google forms layout look like when you open from a computer

Appendix B

Pipelines

B.1 CI pipeline web

```
1 // CI pipeline for web
2 name: CI
3
4 # Trigger the workflow on push or pull requests to the main or dev branches
5 on:
6   pull_request:
7     branches:
8       - main
9       - preprod
10      - dev
11
12 jobs:
13
14   eslint:
15     runs-on: ubuntu-latest
16
17 steps:
18   # Check out the code from the repository
19   - name: Checkout code
20     uses: actions/checkout@v3
21
22   # Set up Node.js
23   - name: Set up Node.js
24     uses: actions/setup-node@v3
25     with:
26       node-version: '22'
```

```

27
28     # Install dependencies
29     - name: Install dependencies
30         run: npm install
31
32     # Run ESLint to check for linting errors
33     - name: Run ESLint
34         run: npm run lint
35
36
37 test:
38     runs-on: ubuntu-latest
39     needs: eslint
40     steps:
41
42         # Check out the code from the repository
43         - name: Checkout code
44             uses: actions/checkout@v3
45
46         # Set up Node.js
47         - name: Set up Node.js
48             uses: actions/setup-node@v3
49             with:
50                 node-version: '22'
51
52         # Install dependencies
53         - name: Install dependencies
54             run: npm install
55
56         # Run Jest tests
57         - name: Run Jest tests
58             run: npm test
59
60 build:
61     runs-on: ubuntu-latest
62     needs: test
63     steps:
64         - name: Checkout Code
65             uses: actions/checkout@v2
66
67         - name: Set up Node.js
68             uses: actions/setup-node@v2
69             with:
70                 node-version: '22'
71
72         - name: Install Dependencies
73             run: npm install

```

```

74
75      - name: Build Project
76      env:
77        REACT_APP_BACKEND_URL: ${{ secrets.BACKEND_URL }}
78        CI: false # Disable treating warnings as errors
79      run: npm run build

```

B.2 CI/CD pipeline web -preprod

```

1 // CI/CD pipeline for web - preprod
2 name: CI/CD - production
3
4 on:
5   push:
6     branches:
7       - main
8
9 jobs:
10   eslint:
11     runs-on: ubuntu-latest
12
13 steps:
14   # Check out the code from the repository
15   - name: Checkout code
16     uses: actions/checkout@v3
17
18   # Set up Node.js
19   - name: Set up Node.js
20     uses: actions/setup-node@v3
21     with:
22       node-version: '22'
23
24   # Install dependencies
25   - name: Install dependencies
26     run: npm install
27
28   # Run ESLint to check for linting errors
29   - name: Run ESLint
30     run: npm run lint
31
32 test:
33   runs-on: ubuntu-latest
34   needs: eslint

```

```

35
36   steps:
37     # Check out the code from the repository
38     - name: Checkout code
39       uses: actions/checkout@v3
40
41     # Set up Node.js
42     - name: Set up Node.js
43       uses: actions/setup-node@v3
44       with:
45         node-version: '22'
46
47     # Install dependencies
48     - name: Install dependencies
49       run: npm install
50
51     # Run Jest tests
52     - name: Run Jest tests
53       run: npm test
54
55   build:
56     runs-on: ubuntu-latest
57     needs: test
58
59   steps:
60     # Check out the code from the repository
61     - name: Checkout code
62       uses: actions/checkout@v3
63
64     # Set up Node.js
65     - name: Set up Node.js
66       uses: actions/setup-node@v3
67       with:
68         node-version: '22'
69
70     # Install dependencies
71     - name: Install dependencies
72       run: npm install
73
74     # Build the project
75     - name: Build Project
76       env:
77         REACT_APP_BACKEND_URL: ${{ secrets.BACKEND_URL_PROD }}
78         CI: false  # Disable treating warnings as errors
79       run: npm run build
80
81   # Upload the build artifact

```

```

82      - name: Upload Build Artifact
83        uses: actions/upload-artifact@v3
84        with:
85          name: build
86          path: build/
87
88    deploy:
89      runs-on: ubuntu-latest
90      needs: build
91
92    steps:
93
94      # Download the build artifact
95      - name: Download Build Artifact
96        uses: actions/download-artifact@v3
97        with:
98          name: build
99          path: ./ # Download files into the current directory
100
101     # Clean the target directory on the server
102     - name: Clean Target Directory on Server
103       uses: appleboy/ssh-action@v0.1.6
104       with:
105         host: ${secrets.SERVER_HOST_PROD}
106         username: ${secrets.SERVER_USER_PROD}
107         key: ${secrets.SERVER_SSH_KEY_PROD}
108         script: |
109           rm -rf /var/www/madboks-web/*
110
111     # Upload the build to the server
112     - name: Upload Build to Server
113       uses: appleboy/scp-action@v0.1.4
114       with:
115         host: ${secrets.SERVER_HOST_PROD}
116         username: ${secrets.SERVER_USER_PROD}
117         key: ${secrets.SERVER_SSH_KEY_PROD}
118         source: "./*"
119         target: "/var/www/madboks-web"

```

We had another CI/CD pipeline for pre-production. It has exactly the same steps, except it triggers on pushes to the preprod branch, and has different github secret variables so it deploys to the preprod server.

B.3 CI pipeline backend

```
1 // CI pipeline for backend
2 name: CI
3
4 # Trigger the workflow on pull requests to the main or dev branches
5 on:
6   pull_request:
7     branches:
8       - main
9       - dev
10      - preprod
11
12 jobs:
13
14   test:
15     runs-on: ubuntu-latest
16
17   steps:
18     # Check out the code from the repository
19     - name: Checkout code
20       uses: actions/checkout@v3
21
22     # Set up Node.js
23     - name: Set up Node.js
24       uses: actions/setup-node@v3
25     with:
26       node-version: '22'
27
28     # Install dependencies
29     - name: Install dependencies
30       run: npm install
31
32     # Run Jest tests
33     - name: Run Jest
34       run: npm test
35
36 build:
37   needs: test
38   runs-on: ubuntu-latest
39
40   steps:
41     # Step 1: Checkout the code
42     - name: Checkout Code
43       uses: actions/checkout@v2
44
```

```

45      # Step 2: Install Dependencies
46      - name: Install Dependencies
47          run: npm install
48
49      # Step 3: Build the Project
50      - name: Build Project
51          env:
52              SUPABASE_URL: ${{ secrets.SUPABASE_URL }}
53              SUPABASE_KEY: ${{ secrets.SUPABASE_KEY }}
54              SUPABASE_SERVICE_ROLE: ${{ secrets.SUPABASE_SERVICE_ROLE }}
55              SMTP_USER: ${{ secrets.SMTP_USER }}
56              SMTP_HOST: ${{ secrets.SMTP_HOST }}
57              SMTP_PASS: ${{ secrets.SMTP_PASS }}
58              SMTP_PORT: ${{ secrets.SMTP_PORT }}
59              FROM: ${{ secrets.FROM }}
60              FRONTEND_URL: ${{ secrets.FRONTEND_URL }}
61          run: npm run build

```

B.4 CI/CD pipeline backend

```

1 // CI/CD pipeline for backend
2 name: CI/CD Pipeline - Production
3
4 on:
5     push:
6         branches:
7             - main
8
9 jobs:
10    test:
11        runs-on: ubuntu-latest
12
13 steps:
14     # Check out the code from the repository
15     - name: Checkout code
16         uses: actions/checkout@v3
17
18     # Set up Node.js
19     - name: Set up Node.js
20         uses: actions/setup-node@v3
21         with:
22             node-version: '22'
23

```

```

24      # Install dependencies
25      - name: Install dependencies
26          run: npm install
27
28      # Run Jest tests
29      - name: Run Jest
30          run: npm test
31
32  deploy:
33      needs: test
34      runs-on: ubuntu-latest
35
36  steps:
37      # Step 1: Checkout the code
38      - name: Checkout Code
39          uses: actions/checkout@v2
40
41      # Step 2: Install Dependencies
42      - name: Install Dependencies
43          run: npm install
44
45      # Step 3: Build the Project
46      - name: Build Project
47          env:
48              SUPABASE_URL: ${{ secrets.SUPABASE_URL_PROD }}
49              SUPABASE_KEY: ${{ secrets.SUPABASE_KEY_PROD }}
50              SUPABASE_SERVICE_ROLE: ${{ secrets.SUPABASE_SERVICE_ROLE_PROD }}
51              SMTP_USER: ${{ secrets.SMTP_USER }}
52              SMTP_HOST: ${{ secrets.SMTP_HOST }}
53              SMTP_PASS: ${{ secrets.SMTP_PASS }}
54              SMTP_PORT: ${{ secrets.SMTP_PORT }}
55              FROM: ${secrets.FROM}
56              FRONTEND_URL: ${{ secrets.FRONTEND_URL_PROD }}
57          run: npm run build
58
59      # Step 4: Upload Files to the Server
60      - name: Upload Files to Server
61          uses: appleboy/scp-action@v0.1.4
62          with:
63              host: ${secrets.SERVER_HOST_PROD}
64              username: ${secrets.SERVER_USER_PROD}
65              key: ${secrets.SERVER_SSH_KEY_PROD}
66              source: "."
67              target: "/var/www/madboks-backend"
68
69      # Step 5: Restart Fastify Service with Environment Variables
70      - name: Restart Fastify Service

```

```

71      uses: appleboy/ssh-action@v0.1.6
72      with:
73        host: ${ secrets.SERVER_HOST_PROD }
74        username: ${ secrets.SERVER_USER_PROD }
75        key: ${ secrets.SERVER_SSH_KEY_PROD }
76        script: |
77          # Export all environment variables
78          echo "Setting environment variables..."
79          export SUPABASE_URL=${ secrets.SUPABASE_URL_PROD }
80          export SUPABASE_KEY=${ secrets.SUPABASE_KEY_PROD }
81          export SUPABASE_SERVICE_ROLE=${ secrets.SUPABASE_SERVICE_ROLE_PROD }
82          export SMTP_USER=${ secrets.SMTP_USER }
83          export SMTP_HOST=${ secrets.SMTP_HOST }
84          export SMTP_PASS=${ secrets.SMTP_PASS }
85          export SMTP_PORT=${ secrets.SMTP_PORT }
86          export FROM=${ secrets.FROM }
87          export FRONTEND_URL=${ secrets.FRONTEND_URL_PROD }

88
89      # Create an .env file for the app
90      echo "Creating .env file..."
91      cat <<EOF > /var/www/madboks-backend/.env
92      SUPABASE_URL=${ secrets.SUPABASE_URL_PROD }
93      SUPABASE_KEY=${ secrets.SUPABASE_KEY_PROD }
94      SUPABASE_SERVICE_ROLE=${ secrets.SUPABASE_SERVICE_ROLE_PROD }
95      SMTP_USER=${ secrets.SMTP_USER }
96      SMTP_HOST=${ secrets.SMTP_HOST }
97      SMTP_PASS=${ secrets.SMTP_PASS }
98      SMTP_PORT=${ secrets.SMTP_PORT }
99      FROM=${ secrets.FROM }
100     FRONTEND_URL=${ secrets.FRONTEND_URL_PROD }
101     EOF

102
103      # Install dependencies and restart the app
104      echo "Restarting application..."
105      cd /var/www/madboks-backend
106      npm install --omit=dev
107      pm2 restart all || pm2 start dist/index.js --name "madboks-backend" --env
→       production

```

We had another CI/CD pipeline for pre-production. It has exactly the same steps, except it triggers on pushes to the preprod branch, and has different github secret variables so it deploys to the preprod server.

Appendix C

Transcription

C.1 First Meeting

Sofia:

So maybe just to get that out of the way, if it's okay that we record the audio so we have for notes afterwards.

Roxana:

Yeah, that's no problem at all.

Sofia:

Perfect. So, we have prepared a couple of questions, and then we will also show you some examples of how the design can look like.

Bence:

I can take the questions. So, just to catch up from last time, because some of us were not there, unfortunately.

Roxana:

Yeah, nice to meet you.

Bence:

You too. My name is Bence, by the way. I don't know if it, oh yeah, it should show my name, hopefully. It does, but

Roxana:

It's showing Sabo.

Bence:

Yeah, that's the second name.

It should be the other way. But yeah, anyways, so last time, I think you talked about some stuff that would be very nice to have documented and probably one of them was like this, overview

of how the mailbox works today. So could you maybe just quickly do like a rundown of the daily operations and big picture?

Roxana:

Do you want to hear about how the food distribution work in general or how the booking system works in particular?

Bence:

I think both, but the booking system is more important for us.

Roxana:

Okay, cool. So Madboks works on the basis of weekly distributions, which we organize three times a week: Two in Amager on Wednesdays and Sundays, and one in Nørrebro on Saturdays. For these food distributions, the volunteers collect food donations from supermarkets and bakeries. They bring it to our culture houses, where we have partnerships with Copenhagen municipality to use these spaces, and they sort the good [food] from the bad.

They categorize it and then make these food boxes with a variety of fruits, veggies, bread, pastries—whatever food donations we receive from supermarkets. In order for people to book these food boxes, currently we have Google forms that people fill in on a weekly basis. The Google forms open up the day before the food distribution.

People can choose from different time slots at 10-minute intervals. On Saturdays and Sundays, it's between 3 and 4 pm, and on Wednesdays, it's between 5:30 and 6:30 pm, because we also wanted to have a food distribution during the weekdays after work hours so that people can access those as well.

In order to get people to book these boxes, we normally just post on different local Facebook groups. The majority of people who come to our events are from the local area—neighbors, students. We also promote in a lot of refugee groups and social groups for people living on lower incomes, single-parent households, and immigrant groups, so we try to target groups that would benefit most from having access to free food, and also groups interested in this project from a sustainability perspective. And that's kind of it in a nutshell.

Bence:

Yeah, I think that's good.

Sofia:

Just a quick question since we talked about it. So if people book and do not meet up, or if they cancel last minute, do you have any record of that? Do you write it down, or do you have some sort of numbers of the spillover from people not showing up or canceling too late?

Roxana:

Yes. So, in order for people to cancel, the booking form also instructs people that if they need to cancel their booking, they should write to our Instagram or Facebook page, where our communications volunteers can see the messages and delete their name from the attendance list, so that we're not going to prepare boxes for them.

On the day, we have a bit of a margin of error that we've gotten used to. So we always expect to have five to ten no-shows, and we usually take that into account when making the boxes. For instance, if we collected a hundred boxes, and we know that because we collected a hundred,

we can give a hundred boxes out, right?

To ensure that we don't have food waste at the end of the event, we usually open up about 110 bookings. So then we have 10 extra bookings. We normally get three or four cancellations, and the rest—six or seven—we expect won't show up. If we see during the day that everyone is showing up, then we might split some of the boxes and make slightly smaller ones. But normally, it works out. Around 100-102 people show up out of the 100 we expected. Sometimes even one or two fewer.

Sofia:

Okay. Thank you.

Bence:

Yeah, and also, for the process, what part do you find most frustrating?

Roxana:

All of it.

No, I think the part of the process I find a bit annoying, from our perspective, is that people can't easily edit and adjust their booking. People can, but they don't know that they can edit their responses in the Google forms, because it's a pretty basic tool. You can edit your responses, but not a lot of people know that.

So, because they don't do it, we end up playing a kind of mind game with ourselves: we have this many bookings, this many boxes, expect this many no-shows, and expect this many cancellations. And it also means having a volunteer who is actively checking Facebook and Instagram throughout the day on the days of the food distributions. This can be annoying, especially on Wednesdays, as many of our volunteers are either students or work, so it's not always easy to have someone checking and responding immediately to cancellations or changes.

The fact that you can't easily edit or change bookings is a problem. I also think it's an issue from the attendees' perspective, because filling out the booking form doesn't take long. A lot of the people who come to our events are regulars. I think at this point they don't even read the information anymore—they just click through and show up. And because they don't use Google forms often, they don't know they can edit their booking.

They don't know the reason we ask people to log in with their Google accounts is so they can receive a copy of their responses, which functions as a confirmation. I'm not going to wait until every box is booked to manually send emails to confirm bookings, right? So that manual aspect is a little frustrating.

Lastly, Google add-ons aren't the best, so we can't easily automate the process of limiting bookings per time slot. What happens is that the add-ons don't work well or might stop working altogether, as has happened recently. We end up with 40 bookings at 3 o'clock, and then none until 3:30. It kind of works out because people aren't always on time, but it's a problem in winter. It's not a big deal to queue for 10 minutes in summer, but in winter, when it's cold and rainy, it's an issue when people have to queue outside, as there's no indoor space. Having the ability to limit responses by time frame would help spread out the attendees and avoid people standing in the cold.

Appendix D

Analysis

Class	Description	Selected
Admin	Manages events, reservations, and logistics, with operations like creating, modifying, or cancelling events.	✓
Volunteer	Handles logistics and food distribution tasks, including collection, sorting, and preparation.	✓
Customer	Central to reservations, interacting with bookings and cancellations.	✓
Event	Organises mainly food distribution.	✓
Reservation	Tracks individual reservations, including customer details, associated events, and status.	✓
Organisation Box	Manages food during preparation.	✓
Customer Box	Represents food boxes reserved or picked up by customers.	✓
Transaction	Manages payment details.	✓
Location	Represents venues for events.	✓
Food Retailer	Supplies surplus food, coordinates donations, and schedules logistics.	X
Food Item	Represents individual food items.	X
Rental Car	Used for food transport logistics.	X
Device	Represents computers and phones for when using the system.	X
Google Forms	Third party tool for reservations.	X
Excel Sheet	Third party tool for tracking logistics.	X
Social Media Post	Used for updates and announcements.	X
Social Media Account	Used for managing posts.	X

Class	Description	Selected
-------	-------------	----------

Table D.1: Class description and selection

Events	Description	Selected
Reservation Created	A reservation is made for a food box by a customer.	✓
Reservation Modified	Details of an existing reservation are updated.	✓
Reservation Cancelled	A reservation is cancelled by the customer or admin.	✓
Event Created	An event is added to the system by an admin.	✓
Event Modified	An existing event is updated with new information.	✓
Event Cancelled	An event is removed or cancelled.	✓
Location Created	A location is added to the system by an admin.	✓
Location Modified	Updates are made to a venue's details.	✓
Location Deleted	A venue is permanently removed from the system.	✓
Bought	A payment is made for a reserved food box, completing the transaction.	✓
Prepared	A food box is prepared and made available for reservation or distribution.	✓
Signed Up/In	A user (customer, volunteer, or admin) registers or signs in to the system.	✓
Signed Out	A user (customer, volunteer, or admin) signs out of the system.	✓
Account Deleted	A user account is permanently removed from the system.	✓
Assigned	A volunteer is assigned to assist with an event.	X
Shipped	Food items are transported from retailers to sorting or distribution locations.	X
Distributed	Prepared food boxes are handed out to customers during a distribution event.	X
Processed	Payments made manually, like cash, outside the system are handled.	X

Table D.2: Event description and selection

Appendix E

Navigation

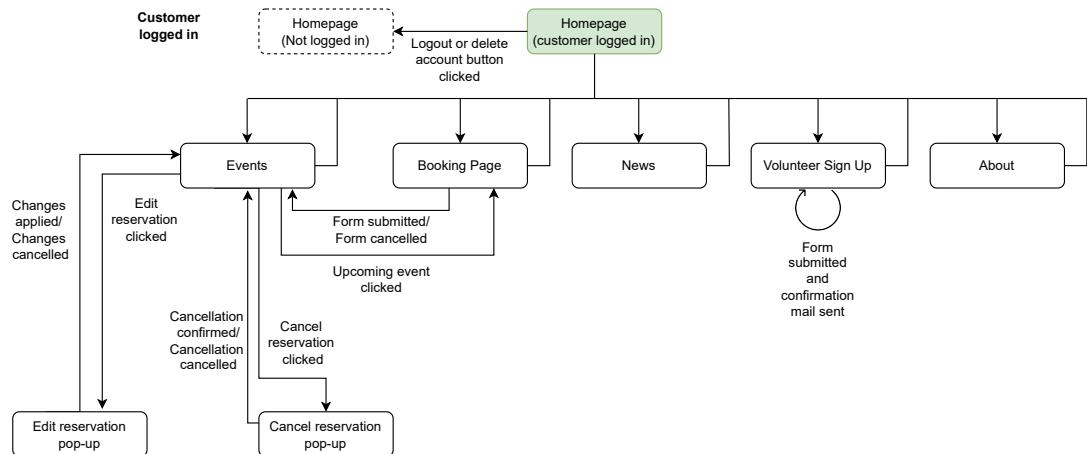


Figure E.1: Navigation diagram for logged-in users

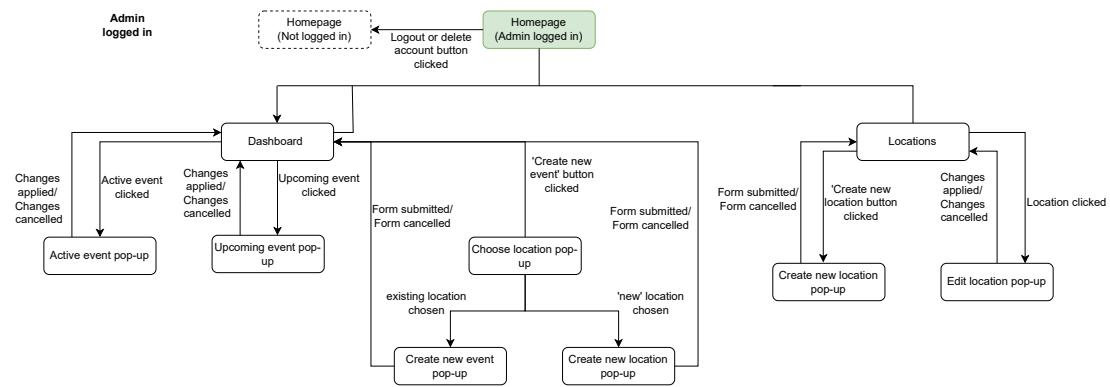


Figure E.2: Navigation diagram for admins

Appendix F

Product owner final evaluation

Questions asked:

*"To finalise our project, we wanted to hear your thoughts/feedback on the collaboration.
How satisfied are you with the communication and feedback during this project?
Were you satisfied with the frequency and format of the meetings?
Did you feel that your input was listened to and incorporated into the development process?
Thank you so much!"*

Roxana Zalto written response:

"Feedback:

- Very happy with the work in general and with the project
- Communication and frequency of meetings: very happy with the frequency of meetings and would maybe encourage a bit more communication and collaboration online between meetings on an as-needed basis to ensure things are tackled as they happen
- Meeting format great
- I definitely felt my input was considered and incorporated in the project and I'm super happy with the end result."

Appendix G

User Stories

Customer User Stories:

- As a customer, I want to view available food box events so that I can select a distribution event that aligns with my schedule.
- As a customer, I want to reserve a food box by selecting from available time slots to ensure convenient access at my preferred time.
- As a customer, I want to receive a booking confirmation with reservation details so that I have a clear record of my booking.
- As a customer, I want the ability to modify or cancel my reservation if I cannot attend, ensuring that my slot can be reassigned effectively.
- As a customer, I want to receive notifications if an event or my reservation is modified or cancelled, allowing me to adjust my plans accordingly.
- As a customer, I want to contact Madboks support easily if I have any questions.

Admin User Stories:

- As an admin, I want to create, update, or delete food distribution events to effectively manage distributions based on available resources and demand.
- As an admin, I want efficient reservation management, including booking limitations and cancellation tracking, to avoid overbooking and optimise user experience.
- As an admin, I want notifications of new reservations, modifications, and cancellations to remain informed and make necessary adjustments.
- As an admin, I want access to templates for recurring events because it makes event setup easier and more efficient.

Appendix H

Sprints

H.0.0.1 S1F1: Homepage (first iteration)

The homepage serves as the entry point for the web platform. It encapsulates Madboks' main mission goals and prompts new users to give website functionalities a go.

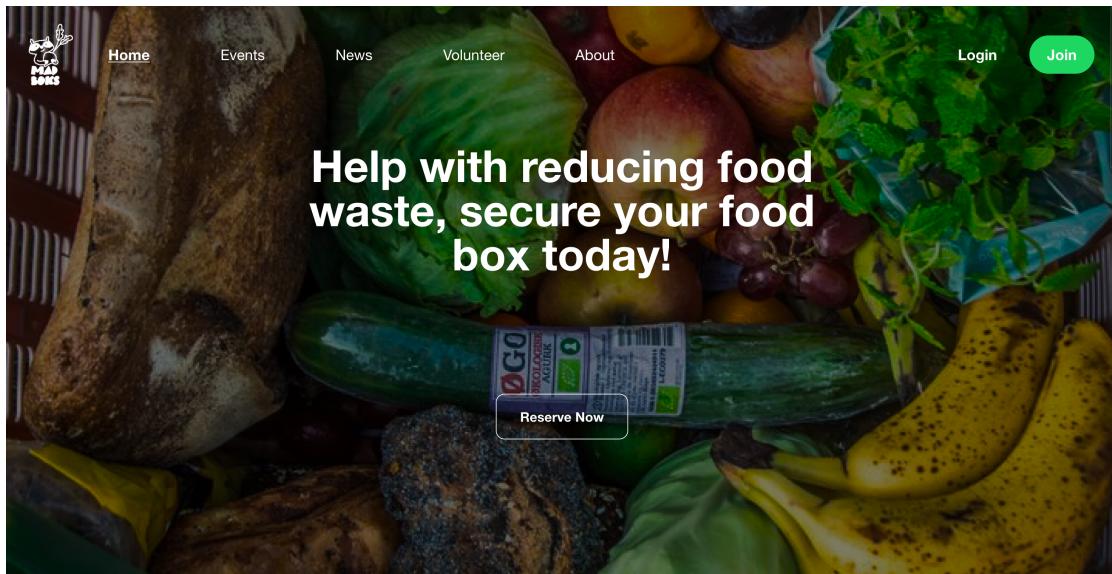


Figure H.1: First iteration of the homepage

The information is broken down into sections encapsulating one core functionality or message each, such as a call-to-action for becoming a volunteer, which encourages visitors to actively contribute to Madboks' mission. Other highlights include information about upcoming events

and locations.

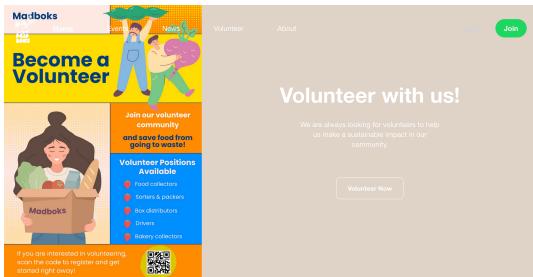


Figure H.2: First iteration of the homepage 2

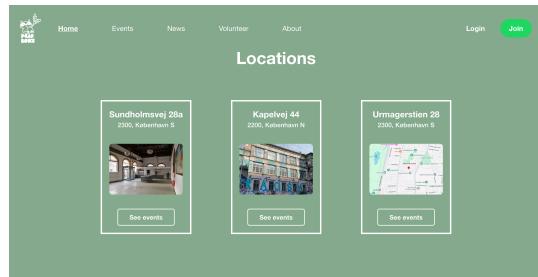


Figure H.3: First iteration of the homepage 3

H.0.0.2 S1F2: Navigation bar (hardcoded first iteration)

The navigation bar enables quick navigation across the website for different user roles (guests, customers, and admins). A notable focus of the development team is to minimise the number of clicks it takes to get from A to B while avoiding clutter.

For guests, the bar displays options like Home, Events, News, Volunteer, About and Login/Register.

Additionally, the Madboks logo, displayed on the left side of the bar, functions as a home button, allowing users to quickly return to the main landing page.

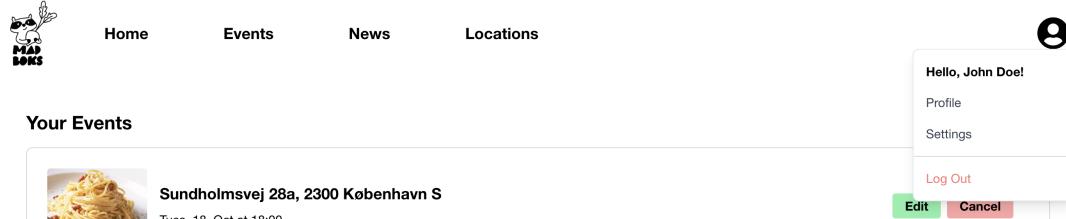


Figure H.4: Navigation bar

H.0.0.3 S1F3: Upcoming events component (mock data)

The upcoming events component displays a list of the upcoming food distribution events. Each card includes the event name (which is the location), date and time. The design of this component closely resembles Facebook's design 2.

Upcoming Events

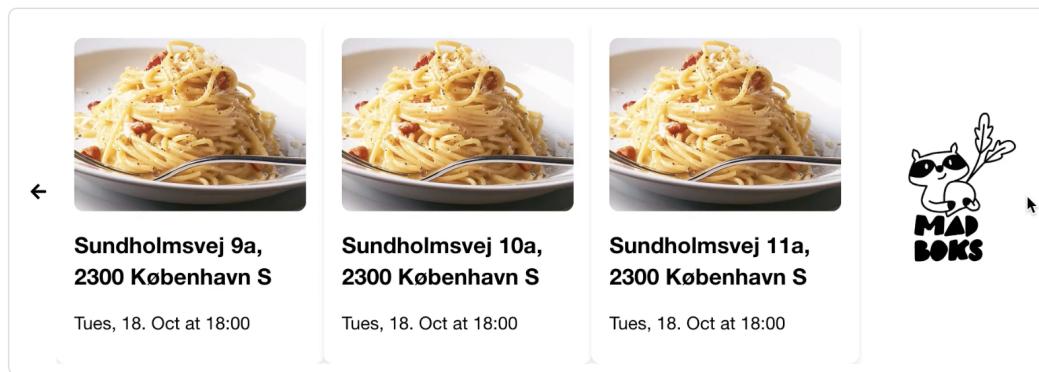


Figure H.5: Upcoming events component

H.0.0.4 S1F4: Your events component (mock data)

The your events component provides logged-in customers with an overview of the events they have booked boxes for. Since this component was developed in parallel with the login and signup functionalities, it used mock data for now. The reservation is displayed, showing details like event name, date and time.

From a technical standpoint, it works almost identically to the upcoming events but uses a horizontal display component, and is currently missing the edit a cancel buttons - these are added briefly following the sprint review. The design of this component is also inspired by Facebook's "my events" page.

The image shows a user interface component titled "Your Events". At the top, there is a navigation bar with icons for "Book", "Events", "News", "Login", and a green "Join" button. Below the navigation bar, the title "Your Events" is displayed. Underneath, there are two cards, each showing a plate of spaghetti. The first card has the address "Sundholmsvej 28a, 2300 København S" and the date "Tues, 18. Oct at 18:00". The second card has the same information. At the bottom of the list is a "Show More" button.

Figure H.6: Your events component

H.0.0.5 S1F5 Location creation

The location creation is a feature for admins, that enables the creation and management of locations templates. When creating a location, the admin-user type in the 'default' values for that location. These default values are then used as 'templates' when creating a food distribution event for that location. New locations could be created by selecting 'new' in the drop-down menu for event creation.

H.0.0.6 S1F6: Event creation (first iteration)

The events creation is a feature for admins, that enables the creation and management of food distribution events. When creating a new event, it is based on the default values from the location. The event creation is inspired by Facebook Marketplace, where the left side is where you write the values, and on the right side, you see at preview.

H.0.0.7 S1F7: Reservation page

The reservation page allows customers to book food boxes for specific events. Users select a desired time slot and the desired number of boxes, fill out their contact info and confirm their reservation by agreeing to the terms and conditions. The page is separated into sections, some collapsible, to make it less overwhelming to use. The idea is to only show the contact info section as the default for non-logged-in users since the system does not have the required information to auto-fill those fields. This would make the reservation flow more effective for logged-in users, and give the non-logged-in user an incentive to create an account.



Figure H.7: Reservation page

How many boxes would you like to reserve? - 1 +

Choose time for pick-up

Choose your donation i

25 DKK Minimum donation Covers food transport	50 DKK Medium donation Covers all event logistics	75 DKK Large donation Helps organise future events
--	--	---

Contact info

First Name <input style="width: 100%; height: 25px;" type="text"/>	Last Name <input style="width: 100%; height: 25px;" type="text"/>	Occupation Student Employed Unemployed Refugee background
Email <input style="width: 100%; height: 25px;" type="text"/>	Phone Number <input style="width: 100%; height: 25px;" type="text"/>	Single parent Pensionist Low income family Other

Comments

Cancel Reserve

Figure H.8: Reservation page 2

An important design decision, inspired by TooGoodToGo and Facebook Marketplace, was the use of visually distinct modals to break up the lengthy form, reducing repetition and confusion while differentiating sections. The design prioritised maintaining a visual hierarchy and ensuring clear separation between sections to enhance usability and prevent users from feeling overwhelmed.

H.0.0.8 S2F2: Volunteer page

The volunteer page is fairly straightforward containing some clarification about the volunteering process and a form to submit a request to become a Madboks volunteer. Since the email service is still being worked on, this product backlog item only considers the creation of the form, but it is not yet hooked up to any email service.

The information needed here was decided by the product owner. To ensure that the design looks good on all screen sizes, the styling of this page considers all screen sizes from desktop through tablet to mobile. Making the entire website mobile-friendly and highly responsive should also be an important thing to keep in mind for any other page.

Volunteer Signup

First Name	Last Name
<input type="text"/>	<input type="text"/>
Email	Phone Number
<input type="text"/>	<input type="text"/>
Date of Birth	Country of Origin
<input type="text"/>	<input type="text"/>
How did you hear about us?	
<input type="text"/>	

Volunteer Preferences

How often would you like to volunteer?

Every week Every second week Once a month

Do you see yourself as a potential team leader?

Figure H.9: Volunteer page

H.0.0.9 S2F7: Event popup, showing information and description of the event

On the page 'events', users can now click on an upcoming event to view more details or proceed with a reservation. The pop-up can be seen in figure H.10.

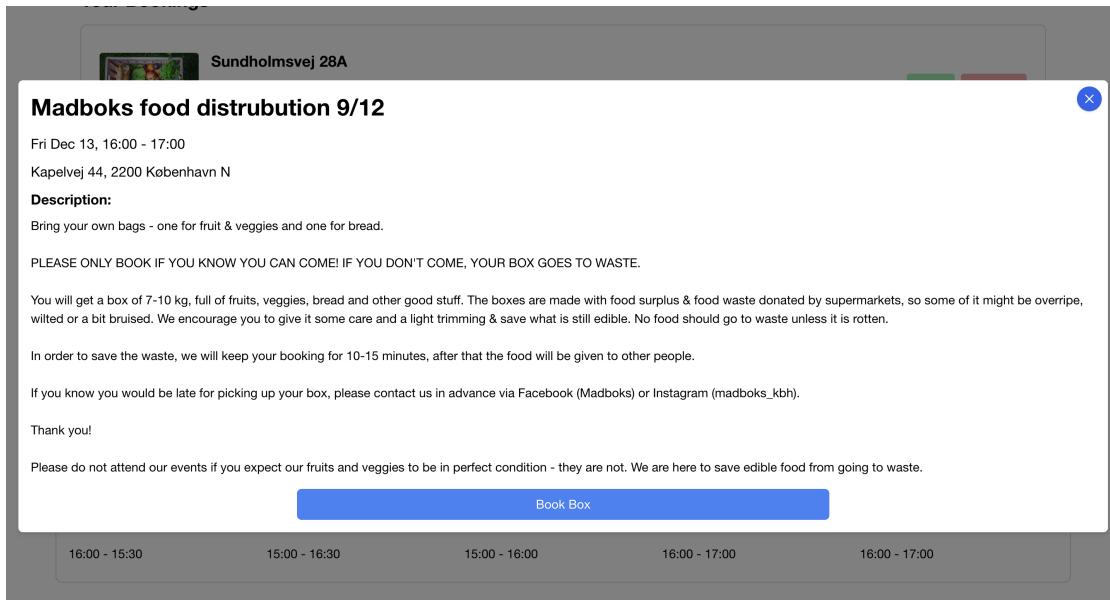


Figure H.10: Event info pop-up

H.0.0.10 S2F4: Admin dashboard - display events/locations and edit location functionality

The admin dashboard has the following components; 'create new event' button, a list of upcoming events and a list of locations. If the user clicks on a location, a pop-up for editing location shows up.

H.0.1 S2F8: Connect navigation bar with auth

Now the navigation bar works with Supabase authentication. Once logged in, the navigation should adjust to include tailored links based on the user's role. For customer, it is: 'Home', 'Reservation', 'Events', 'Volunteer', 'News' and 'About', and for admins, it is: 'Dashboard', 'Locations', 'Volunteer' and 'Events'. The code checks for a logged in user with admin role to be able see the admin workflow.

H.0.2 S2B1: Setting up the server and hosting + pipelines

The Ubuntu server for hosting the website and the backend was created, and the GitHub Actions were used to create CI/CD pipelines that both run tests, build and deploy to the server.

H.0.3 S2B2: Email service backend

```
1  this.fastify.post(
2      '/api/send-confirmation-email',
3      async (request, reply) => {
4          const { to } = request.body as { to: string };
5
6          // Validate input
7          if (!to) {
8              return reply
9                  .status(400)
10                 .send({ error: 'Missing required field: to' });
11         }
12
13         try {
14             const mailService = MailService.getInstance();
15             await mailService.sendConfirmationEmail(to);
16             return reply.status(200).send({
17                 message: 'Confirmation email sent successfully!',
18             });
19         } catch (error) {
20             request.log.error('Error sending confirmation email:', error);
21             return reply
22                 .status(500)
23                 .send({ error: 'Failed to send confirmation email.' });
24         }
25     },
26 );
```

H.0.4 S3F3: About us page

Creating the "about us" page was quite straightforward talking about its contents with the product owner and implementing a mobile-compatible, purely frontend solution. This page encapsulates the most important things to learn about Madboks as someone completely new to this non-profit organization, and the most important call to action and social-media buttons are added to ensure a smooth flow from this page to any other.

The screenshot shows two identical web pages side-by-side, likely demonstrating a comparison between different versions of the 'About Us' content. Both pages have a header with a logo, navigation links (Home, Events, News, Reservation, Volunteer, About, Login, Join), and a 'Join Our Mission' button. Below the header, there are sections for 'About Us', 'Who We Are', 'Our Impact', and 'Join Our Mission'. Each section contains a brief text description and a small note at the bottom. On the right side of each page, there is a 'Contact Us' section with an email address (info@madboks.org) and social media links (Facebook, Instagram). A small 'MADBOKS' logo is visible in the top right corner of the page content.

Figure H.11: About us 1

Figure H.12: About us 2

H.0.5 S3F4

```

1  const updateTimeslots = (boxDifference: number) => {
2      setTimeslotsFormData((prevTimeslots) => {
3          const updatedTimeslots = [...prevTimeslots];
4
5          // Calculate the original total available spots
6          const originalAvailableSpots = originalNumberOfBoxes - reservedBoxes;
7
8          // Calculate the target total spots
9          const targetTotalSpots = boxDifference + originalAvailableSpots;
10
11         // Calculate the current total available spots
12         const currentTotalSpots = updatedTimeslots.reduce((sum, timeslot) => sum +
13             timeslot.available_spots, 0);
14
15         // Calculate the difference to be adjusted to match the target
16         let adjustmentNeeded = targetTotalSpots - currentTotalSpots;
17
18         // Distribute the adjustment evenly across timeslots
19         const timeslotCount = updatedTimeslots.length;
20         const equalAdjustment = Math.floor(adjustmentNeeded / timeslotCount);
21         let remainder = adjustmentNeeded % timeslotCount;
22
23         // Apply equal adjustments to all timeslots
24         updatedTimeslots.forEach((timeslot) => {
25             timeslot.available_spots += equalAdjustment;
26         });
27
28         // Distribute the remainder
29         for (let i = 0; remainder !== 0 && i < timeslotCount; i++) {
30             if (remainder > 0) {

```

```

30         updatedTimeslots[i].available_spots += 1;
31         remainder--;
32     } else if (remainder < 0 && updatedTimeslots[i].available_spots > 0) {
33         updatedTimeslots[i].available_spots -= 1;
34         remainder++;
35     }
36 }
37
38 // Ensure no timeslot has negative spots and adjust if necessary
39 let extraNeeded = 0; // Track how much adjustment is required to fix negatives
40 updatedTimeslots.forEach((timeslot) => {
41     if (timeslot.available_spots < 0) {
42         extraNeeded += timeslot.available_spots; // Negative values decrease the
43         ↵ adjustment
44         timeslot.available_spots = 0; // Reset to 0 since spots can't be negative
45     }
46 });
47
48 // Redistribute the extra adjustment across other timeslots
49 if (extraNeeded < 0) {
50     for (let i = 0; extraNeeded !== 0 && i < timeslotCount; i++) {
51         if (updatedTimeslots[i].available_spots > 0) {
52             const reduction = Math.min(updatedTimeslots[i].available_spots,
53             ↵ -extraNeeded);
54             updatedTimeslots[i].available_spots -= reduction;
55             extraNeeded += reduction;
56         }
57     }
58
59     return updatedTimeslots;
60 });

```

H.0.6 S3B4

H.0.6.1 S3B4-1

```

1 fastify.post(
2   `${prefix}`,
3   (req: FastifyRequest<{ Body: { data: Reservation } }>, reply) =>
4     reservationController.createReservation(req, reply),
5   );

```

H.0.6.2 S3B4-2

```
1  public async createReservation(
2    req: FastifyRequest<{ Body: { data: Reservation } }>,
3    reply: FastifyReply,
4  ) {
5    const result = await this.reservationService.createReservation(
6      req.body.data,
7    );
8    return this.sendResponse(reply, result, 201);
9 }
```

H.0.6.3 S3B4-3

```
1  public async createReservation(
2    record: Reservation,
3  ): Promise<Result<Reservation, BaseError>> {
4    const validationResult = this.validate(record, false);
5    if (validationResult.success === false) return validationResult;
6
7    // Attempt to book the timeslot
8    try {
9      await TimeslotService.getInstance().bookTimeslot(record.timeslot_id);
10   } catch (error) {
11     return ResultFactory.Err(
12       this.handleSupabaseError(error, 'bookTimeslot', {
13         record: record.toJson(),
14       }),
15     );
16   }
17
18   // Attempt to create the reservation
19   const { data, error } = await this.create(record);
20
21   if (error) {
22     // Rollback the timeslot if reservation creation fails
23     try {
24       await TimeslotService.getInstance().cancelTimeslot(record.timeslot_id);
25     } catch (rollbackError) {
26       return ResultFactory.Err(
27         this.handleSupabaseError(rollbackError, 'rollbackTimeslotBooking', {
28           record: record.toJson(),
29         }),
30       );
31     }
32   }
33 }
```

```

30         );
31     }
32
33     return ResultFactory.Err(
34       this.handleSupabaseError(error, 'createReservation', {
35         record: record.toJson(),
36       }),
37     );
38   }
39
40   return this.validateAndMapRecord(data!);
41 }

```

H.0.6.4 S3B4-4

```

1
2   public async updateReservationTimeslot(
3     id: string,
4     new_timeslot_id: string,
5   ): Promise<Result<Reservation, BaseError>> {
6     // Fetch existing reservation
7     const { data: reservation, error: reservationError } =
8       await this.findById(id);
9     if (reservationError) {
10       return ResultFactory.Err(
11         this.handleSupabaseError(
12           reservationError,
13           'updateReservationTimeslot',
14           {
15             id,
16             new_timeslot_id,
17           },
18         ),
19       );
20     }
21
22     if (!reservation) {
23       return ResultFactory.Err(
24         new ValidationError('Reservation not found', { id }),
25       );
26     }
27
28     // Cancel the current timeslot
29     try {

```

```

30     await TimeslotService.getInstance().cancelTimeslot(
31       reservation.timeslot_id,
32     );
33   } catch (error) {
34     return ResultFactory.Err(
35       this.handleSupabaseError(error, 'cancelTimeslot', {
36         id,
37         timeslot_id: reservation.timeslot_id,
38       }),
39     );
40   }
41
42   // Attempt to book the new timeslot
43   try {
44     await TimeslotService.getInstance().bookTimeslot(new_timeslot_id);
45   } catch (error) {
46     // Rollback the previous timeslot booking
47     await TimeslotService.getInstance().bookTimeslot(reservation.timeslot_id);
48   }
49   return ResultFactory.Err(
50     this.handleSupabaseError(error, 'bookTimeslot', {
51       id,
52       new_timeslot_id,
53     }),
54   );
55
56   // Update the reservation with the new timeslot ID
57   const { data: updatedReservation, error: updateError } =
58     await this.updateField(
59       id,
60       ReservationFields.TIMESLOT_ID,
61       new_timeslot_id,
62     );
63
64   if (updateError) {
65     // Rollback the new timeslot and rebook the old one
66     await TimeslotService.getInstance().cancelTimeslot(new_timeslot_id);
67     await TimeslotService.getInstance().bookTimeslot(reservation.timeslot_id);
68   }
69   return ResultFactory.Err(
70     this.handleSupabaseError(updateError, 'updateReservationField', {
71       id,
72       new_timeslot_id,
73     }),
74   );
75
76   return this.validateAndMapRecord(updatedReservation!);

```

77 }

H.0.6.5 S3B4-5

```
1  public async updateActiveEvent(
2      id: string,
3      description: string,
4      number_of_boxes: number,
5      timeslots: Timeslot[],
6      ): Promise<Result<Event, BaseError>> {
7          // Fetch the original state
8          const { data: originalEvent, error: fetchError } = await this.findById(id);
9          if (fetchError) {
10              return ResultFactory.Err(
11                  this.handleSupabaseError(fetchError, 'fetchOriginalEvent', { id }),
12              );
13          }
14
15          // Update the event fields
16          const { data, error } = await this.updateTwoFields(
17              id,
18              'description',
19              description,
20              'number_of_boxes',
21              number_of_boxes,
22          );
23          if (error) {
24              return ResultFactory.Err(
25                  this.handleSupabaseError(error, 'updateEventField', {
26                      id,
27                      description,
28                      number_of_boxes,
29                  }),
30              );
31          }
32
33          try {
34              // Update timeslot availability
35              const timeslotService = TimeslotService.getInstance();
36              await timeslotService.updateTimeslotsAvailability(timeslots);
37          } catch (timeslotUpdateError) {
38              // Rollback event updates
39              await this.updateTwoFields(
40                  id,
```

```

41     'description',
42     originalEvent!.description,
43     'number_of_boxes',
44     originalEvent!.number_of_boxes,
45   );
46
47   // Return the error for timeslot update
48   return ResultFactory.Err(
49     this.handleSupabaseError(
50       timeslotUpdateError,
51       'updateTimeslotsAvailability',
52     {
53       id,
54       timeslots: timeslots.map(timeslot => timeslot.toJson()),
55     },
56   ),
57 );
58 }
59
60 // Validate and map the updated record
61 return this.validateAndMapRecord(data!);
62 }
63

```

H.0.6.6 S3B4-6

```

1  public async updateTimeslotsAvailability(
2    records: Timeslot[],
3  ): Promise<Result<Timeslot[], BaseError>> {
4    for (const record of records) {
5      const validationResult = this.validate(record, false);
6      if (validationResult.success === false) return validationResult;
7    }
8    for (const record of records) {
9      const { data, error } = await this.updateField(
10        record.id,
11        'available_spots',
12        record.available_spots,
13      );
14      if (error)
15        return ResultFactory.Err(
16          this.handleSupabaseError(error, 'updateTimeslotsAvailability', {
17            record: record.toJson(),
18          }),

```

```
19     );
20 }
21
22     return this.validateAndMapRecords(records);
23 }
```

H.0.7 S4F1: User test fixes and small text and layout updates

Fixed line-break in event description, changed the default event description based on PO feedback, fixed so links open in new tab, added info box informing that users cannot book more than one box, fixed the 'see event' button (now navigates to events), fixed navigate to home when sign-out/delete user and fixed the cancel button in reservation page, so it navigates to home.

H.0.8 S4F2: Configuration fix to not expose source code in the browser

In the user test, it was discovered that the source code for the react app is displayed in the browser when opening the debugger tool. To solve this issue, the build command was changed to not include source code files when deploying. The build command is therefore the following:

```
1 "build": "react-scripts build && find build -name '*.map' -type f -delete",
```

Appendix I

Design Criteria for the Madboks Platform

The design of the Madboks platform requires a careful balance between functionality, user experience, and domain-specific needs. Based on insights from the product owner and analysis of state-of-the-art platforms like Too Good To Go and Facebook, the design must streamline food distribution operations, enhance user accessibility, and address logistical challenges unique to Madboks. This involves creating a user-centric solution that supports both administrators managing events and customers managing bookings efficiently while reducing manual labour.

Key Design Criteria

- **User-friendly interface:** Ensure the interface is intuitive for users across all demographics, including first-time users and regular attendees. Draw inspiration from *Too Good To Go*'s explore and reservation pages, as well as *Facebook*'s events page which emphasises clean layouts and intuitive navigation.
- **Simplified booking process:** Provide a quick and straightforward booking system to replace the current reliance on Google Forms. Features like real-time booking confirmations, the ability to edit or cancel reservations, and automated reminders will reduce manual workload for volunteers and improve the user experience.
- **Scalable time slot management:** Incorporate mechanisms to limit bookings per time slot, ensuring even distribution of attendees throughout events. This would prevent overcrowding and mitigate issues like queuing in poor weather, as highlighted by the product owner.
- **Administrators workflow optimisation:** Reduce the reliance on manual processes by automating routine tasks, such as handling cancellations and managing time slots. Integration of tools to monitor attendance and cancellations dynamically would alleviate the

workload on administrators, especially during mid-week distributions.

- **Mobile and desktop compatibility:** Design the platform for cross-device usability, taking cues from *Facebook*'s dual-platform success. A mobile-first approach should prioritise accessibility for attendees, while desktop features can support administrators managing logistics.
- **Effective communication with customers:** Enhance communication with targeted user groups through email notifications, reminders, and updates. By integrating features to notify users of cancellations or booking openings, Madboks can minimise no-shows and optimise food box distribution.
- **Statistics:** Provide an interface, where administrators easily can see relevant statistics for active events. Including information about how many booked boxes there are for the events. This will make it easier for the admins to get an overview of the specific statistics for an event.

Bibliography

- [1] Uğur Aydoğdu. *CAPTCHA Solutions Comparison: Cloudflare Turnstile vs Google reCAPTCHA*. Oct. 2024. URL: <https://epigra.com/en/blog/captcha-solutions-comparison>.
- [2] Brevo. *Email & multichannel marketing software*. 2024. URL: <https://www.brevo.com/products/marketing-platform/>.
- [3] Cloudflare. *Cloudflare Turnstile*. 2024. URL: <https://www.cloudflare.com/application-services/products/turnstile/>.
- [4] CodeScene. *Where Can CodeScene Help You Improve?* 2024. URL: <https://codescene.com/>.
- [5] Jean-Paul Garin, Nisfi Mubarokah, and Arpit Bhutani. *Food Waste in the Municipality of Copenhagen: A CIRCULAR ECONOMY VISION*. Tech. rep. Copenhagen, Denmark: Circular Innovation Lab, Aug. 2022. URL: <https://www.circularinnovationlab.com/post/food-waste-in-the-municipality-of-copenhagen-a-circular-economy-vision>.
- [6] Github. *About Projects*. 2024. URL: <https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects>.
- [7] Grafana Labs. *The best developer experience for load testing*. 2024. URL: <https://k6.io/>.
- [8] LinkedIn. Roxana Zlate. 2024. URL: <https://www.linkedin.com/in/roxana-zlate>.
- [9] Lars Mathiassen et al. *OBJECT-ORIENTED ANALYSIS & DESIGN*. 2. edition. Metodica ApS, 2018. ISBN: 978-87-970693-0-1.
- [10] Jakob Nielsen. *Usability Engineering*. 24-2 8 Oval Road, London NW1 7DX: Academic Press, Limited, 1993. ISBN: 0-12-518406-9. URL: <https://dl.acm.org/doi/pdf/10.5555/2821575>.
- [11] Open Text Corporation. *What is Load Testing?* 2024. URL: <https://www.opentext.com/what-is/load-testing>.
- [12] Mehmet Ozkaya. *Layered (N-Layer) Architecture*. Sept. 2021. URL: <https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-e15ffdb7fa42>.
- [13] State of Green. *Food Waste in Denmark Down by 25 per cent*. July 2015. URL: <https://stateofgreen.com/en/news/food-waste-in-denmark-down-by-25-per-cent/>.
- [14] Supabase. *Supabase Documentation*. 2024. URL: <https://supabase.com/docs>.

- [15] Supabase. *Users*. 2024. URL: <https://supabase.com/docs/guides/auth/users>.
- [16] "The evolution of food donation with respect to waste prevention". In: 33 (). DOI: <https://doi.org/10.1016/j.wasman.2012.10.025>. URL: <https://www.sciencedirect.com/science/article/pii/S0956053X12005430>.
- [17] United States government. *A Call to Action: United States Food Loss & Waste 2030 Reduction Goal*. Nov. 2024. URL: <https://www.epa.gov/sustainable-management-food/call-action-united-states-food-loss-waste-2030-reduction-goal>.
- [18] Wappalyzer. *Cloudflare Turnstile*. 2024. URL: <https://www.wappalyzer.com/technologies/security/cloudflare-turnstile/>.

This bibliography is in alphabetical order.