



# **Internship LHS**

## Realisation document

**Internship 2023-2024**

**Tymo Verhaegen 3APP02**

Academic Year 2022-2024

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

©

# Table of Contents

<b>1 Thank you</b>	<b>5</b>
<b>2 Terminology</b>	<b>6</b>
<b>3 Assignment</b>	<b>7</b>
<b>3.1 Company</b>	<b>7</b>
<b>3.2 Team</b>	<b>7</b>
<b>3.3 Project description</b>	<b>7</b>
<b>4 Technologies and methodologies used</b>	<b>9</b>
<b>4.1 Technologies</b>	<b>9</b>
4.1.1 Django / Python	9
4.1.2 HTML & Tailwind CSS	9
4.1.3 PostgreSQL	9
4.1.4 Javascript	9
4.1.5 Docker	9
<b>4.2 Programs / Platforms</b>	<b>10</b>
4.2.1 Figma	10
4.2.2 (Star)UML	10
4.2.3 Github	10
<b>4.3 Methodologies</b>	<b>10</b>
4.3.1 SCRUM	10
<b>4.4 Pipeline</b>	<b>11</b>
4.4.1 CI/CD	11
<b>5 Realizations</b>	<b>12</b>
<b>5.1 Analysis</b>	<b>12</b>
<b>5.1.1 Use cases</b>	<b>12</b>
5.1.1.1 Use Case Descriptions (customer)	13
5.1.1.2 Use Case Descriptions (staff)	16
5.1.1.3 Use Case Descriptions (company)	18
5.1.1.4 Use Case Descriptions (tool).	20
<b>5.1.2 Database design</b>	<b>21</b>
<b>5.1.3 Sitemap</b>	<b>22</b>
<b>5.1.4 Prototypes</b>	<b>23</b>
5.1.4.1 Calendar (Week View)	23
5.1.4.2 Calendar (Month View)	23
5.1.4.3 Appointment (details)	24
5.1.4.4 Statistics & logs	24
5.1.4.5 Statistics & logs (details)	25
5.1.4.6 Contacts	25
5.1.4.7 Contacts (create)	26
5.1.4.8 Contacts (details)	26
5.1.4.9 Resources	27
5.1.4.10 Resources (details)	27
5.1.4.11 Resources (create branch)	28
5.1.4.12 Company Settings	28

	<b>A4</b>	
5.1.4.13	Company settings (details)	29
5.1.4.14	Promotional codes	29
5.1.4.15	Promotional codes (partners)	30
5.1.4.16	Promotion codes (details)	30
5.1.4.17	Booking	31
5.1.4.18	Booking (cont.).	31
<b>5.2</b>	<b>Front-end</b>	<b>32</b>
5.2.1	Navigation	32
5.2.2	Login	33
5.2.3	Agenda	33
5.2.4	Statistics & logs	34
5.2.5	Customers	35
5.2.6	Employees	36
5.2.7	Services & packages	38
5.2.8	Resources	40
5.2.9	Promotions & offers	41
5.2.10	Reviews	42
5.2.11	Company Settings	42
5.2.12	Booking (end-user)	43
<b>5.3</b>	<b>Backend</b>	<b>44</b>
5.3.1	Navigation	44
5.3.2	Login	45
5.3.3	Agenda	45
5.3.4	Statistics & logs	47
5.3.5	Customers, Employees and Partners	47
5.3.6	Reviews	50
5.3.7	Services & packages	51
5.3.8	Resources	54
5.3.9	Promotions & offers	57
5.3.10	Company settings	58
5.3.11	Booking (end-user)	59
<b>6</b>	<b>Conclusion</b>	<b>60</b>

# 1      **THANK YOU**

On this note, I would like to take a moment to thank LHS by Spot Group for giving me the opportunity to do an internship with the company. This would not have been possible without Steven. I learned a lot during this internship and enjoyed it.

I would also like to thank my colleagues and fellow students, mainly Thomas Verbruggen and Joppe Kerkhofs, they were my groupmates for the project and I enjoyed working out this project with them.

Of course, we must not forget my internship mentor Patrick Dielens. He was a very pleasant and good supervisor who was always ready when we needed him.

## 2 TERMINOLOGY

TERM	RELATED
Framework	Platform or structure for building software that provides tools and conventions to simplify development.
Formatting Language	A language that defines the structure of documents.
Responsive	A design approach that ensures proper operation and display of software on different devices.
Containerization technology	A method of packaging and executing applications using containers for consistent execution on different environments.
UML	Unified Modeling Language: A standardized modeling language for visualization, design and documentation of software projects.
CI/CD	Continuous Integration / Continuous Deployment: A practice for automating processes such as testing and deploying code.
SEA/SEO	<ul style="list-style-type: none"> <li>- Search Engine Advertisement These are paid advertisements in search engines such as Google Ads.</li> <li>- Search Engine Optimization Optimizing web content to rank higher in organic (non-paid) search results</li> </ul>
Use cases	Descriptions of how users will interact with a system/product. They outline the steps a user may take and the possible reactions to them.
Sitemap	Visual representation of the structure and hierarchy of pages within an application or Web site. It shows the relationship between pages.
Wireframes	Wireframes are schematic overviews of a Web site/application, focusing primarily on layout and functionality.
Prototypes	Prototypes are interactive representations of how the final product will look and function
MVC	<p>Model View Controller is a design pattern that divides the design of complex applications into three units with different responsibilities</p> <ul style="list-style-type: none"> <li>- Model: data model</li> <li>- View: data presentation</li> <li>- Controller: application logic</li> </ul>

## 3 ASSIGNMENT

### 3.1 Company

LHS is a small company that specializes in building websites and web applications, since 2023 LHS is part of Spot Group. Spot Group is a larger company with more capacity. As a result, LHS is now part of a company that can offer more services on a larger scale, these are services such as SEA/SEO, hosting, marketing and much more.

### 3.2 Team

The team with which I developed this project consisted of Thomas Verbruggen, Joppe Kerkhofs and myself. All three of us are application development students at Thomas More in Geel.

This is not the first project we have developed together, in fact we have already created school projects together which had advantages for the internship project. This is because we know the strengths and weaknesses of each other and can help each other where necessary.

### 3.3 Project description

A complex booking tool and modular management system with a wide range of functionalities, functionalities include:

- **Booking system**

The central aspect of the web application, it allows customers to easily make appointments, reservations or bookings based on availability.

- **Resource management**

The system will include functions for managing assets, such as rooms, equipment, personnel,..... So this includes tracking resource availability, staff allocations, ....

- **Calendar integration**

Integration of a calendar will be offered, allowing both customers and employees to easily view, change and track appointments.

- **Services and packages**

The system will include features for managing services and packages, so customers can easily choose the right option for their needs.

- **Payment system**

A secure payment system will be implemented to enable online payments for bookings. This will be implemented to save time and make prepayments.

- **Customer Management**

Some sort of customer database will be maintained, with customer information, personal information, booking history and preferences.

- **Management of offers and promotions**

The system will provide the functionality to manage and apply offers, promotions and discounts to customer bookings.

- **Communication and notifications**

Customers receive automatic email notifications when they make a booking, or when changes happen to their booking.

- **Statistics and logs**

Employees will be able to view statistics such as number of bookings, number of users and more, they will also be able to view a detailed history of all customers and their booking history

- **Cronjobs**

The system will perform automated tasks and processes, such as calendar and booking optimization.

- ...

## 4 TECHNOLOGIES USED AND METHODOLOGIES

### 4.1 Technologies

#### 4.1.1 Django / Python

Django is a high-level open-source Python-based Web framework that encourages rapid development and tight, pragmatic design. Django uses the MVC principle.

Reason: Django was LHS's preferred choice for this project.

#### 4.1.2 HTML & Tailwind CSS

HTML is a markup language used to define the structure of Web pages, Tailwind is a framework used to style Web pages to quickly and efficiently build responsive, customized user interfaces.

Reason: HTML is mandatory to create a web application, Tailwind was the group's personal preference. Thanks to its extensive use for school projects, this was an easy choice.

#### 4.1.3 PostgreSQL

PostgreSQL is an open-source relational database management system (RDBMS) used to store and manage data.

Reason: PostgreSQL was LHS's preference for this project.

#### 4.1.4 Javascript

Javascript is a programming language used to develop interactive Web pages. It allows developers to manipulate the content of Web pages.

Reason: Javascript is pretty much mandatory to make a functioning web application.

#### 4.1.5 Docker

Docker is a platform used to develop, ship and run applications. It uses containerization technology to package applications with all the necessary code.

Reason: Docker was a personal preference of the group, that way everyone would always own the same test data and everything would run a little easier.



## **4.2 Programs / Platforms**

### **4.2.1 Figma**

Figma is a design tool used to create user interfaces, wireframes, prototypes and other design elements.

Reason: Figma was a personal preference of the group for creating sitemaps, wireframes and prototypes. This is also an industry standard.

### **4.2.2 (Star)UML**

UML is a standardized modeling language used for visualizing, designing and documenting software projects.

Reason: StarUML was a personal preference of the group to design our database and use cases in, due to its extensive use during school projects.

### **4.2.3 Github**

Github is a platform used for version control and collaboration during software development. On Github, developers can store, edit and share their code with others.

Reason: Github was obvious, it was the preference of LHS and the group because of its extensive use during school projects and this is also the industry standard.

## **4.3 Methodologies**

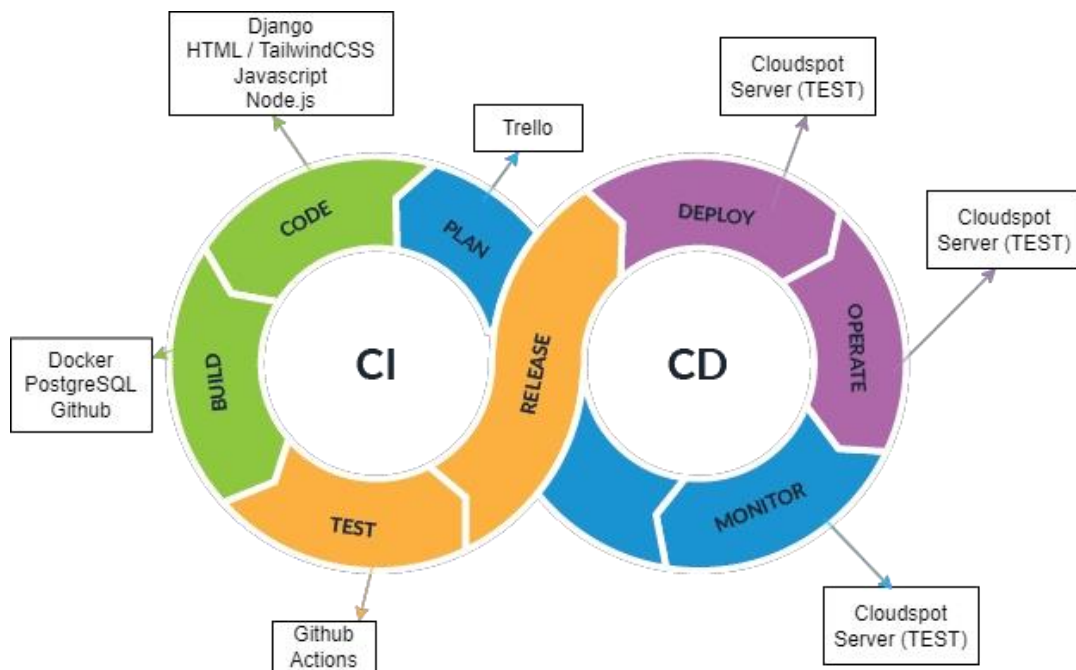
### **4.3.1 SCRUM**

An agile methodology that works on the basis of sprints, typically from two to four weeks. It also includes retrospectives in which the team looks back at the previous sprint and provides each other with feedback on the positives and negatives of that sprint.

Reason: SCRUM methodology was the preferred method of working for our group, the extensive use of this methodology for school projects made this an easy choice.

## 4.4 Pipeline

### 4.4.1 CI/CD



# 5 REALIZATIONS

## 5.1 Analysis

The analysis of the project took place in week 4, during this phase the team conducted an in-depth analysis of user needs.

### 5.1.1 Use cases

StarUML was used to create the use cases



#### 5.1.1.1 Use Case Descriptions (customer)

- Use Case: Login
  - **Description:** The customer can login to the system by entering his/her email and password
  - **Post-conditions:** The customer sees the homepage from the customer-view
  
- Use Case: Logout
  - **Description:** The customer can log out of the system by clicking the logout button.
  - **Conditions:** Customer is logged in
  - **Post-conditions:** The customer is logged out and sent to the login screen
  
- Use Case: Appointment Making
  - **Description:** The customer can make an appointment by selecting a service or package. After this select a time and then click confirm. The customer will have to pay 50% of the amount in advance (*see use case pay*)
  - **Conditions:** Customer is logged in
  - **After-conditions:** The customer is sent to the home page of the customer view. A confirmation email is sent to the customer
  - **Alternative flows:**
    - Staff is logged in
      - The staff member can make an appointment at all hours. He/she can override the system's proposal

- Use Case: Appointment editing
  - **Description:** The client can adjust the services and time of his/her appointment
  - **Conditions:** Customer is logged in
  - **Post-conditions:** The modified appointment is saved in the system and a confirmation email with the changes is sent to the client
  
- Use Case: Payment
  - **Description:** The customer is sent to the Stripe payment system where the payment will be completed
  - **Conditions:** The customer is logged in and has a valid appointment in the system
  
- Use Case: voucher use
  - **Description:** The customer can enter a code of a voucher or discount code during payment
  - **Conditions:** The customer is logged in, has a valid appointment in the system and is making payment
  - **Post-conditions:** The final amount has a reduction in price
  
- Use Case: Leaving a review
  - **Description:** The customer can leave a review after his/her appointment
  - **Conditions:** Customer is logged in and has a completed appointment
  - **Post-conditions:** The review is stored in the system and can be used as a "testimonial" on the company's website

- Use Case: Sharing medical information
  - **Description:** The client can enter his/her medical information
  - **Conditions:** Customer is logged in
  - **Post-conditions:** Client's medical information is stored in the system
  
- Use Case: Consult Price List
  - **Description:** The customer can view the company's price list
  
- Use Case: Consulting opening hours
  - **Description:** The customer can view the opening hours of the business
  
- Use Case: Leaving a review
  - **Description:** The customer can leave a review after his/her appointment
  - **Conditions:** Customer is logged in and has a completed appointment
  - **Post-conditions:** The review is stored in the system and can be used as a "testimonial" on the company's website

#### 5.1.1.2 Use Case Descriptions (staff).

Staff inherits all use cases from customer

- Use case: Manage work hours
  - **Description:** Staff can enter his/her working hours per day
  - **Prerequisites:** The staff member is logged in
  - **Post-conditions:** Working hours are stored in the system
  
- Use Case: Consult Calendar
  - **Description:** Staff can view his/her calendar
  - **Prerequisites:** The press member is logged in
  
- Use case: Managing payments
  - **Description:** Staff can log payments
  - **Prerequisites:** The staff member is logged in
  
- Use case: Managing invoices
  - **Description:** Staff can create an invoice after completing a treatment that is automatically sent to the client's mailbox
  - **Conditions:** The staff member is logged in and treatment is complete
  - **Post-conditions:** The customer will receive an invoice in his/her mailbox
  
- Use Case: Consult Reviews
  - **Description:** Staff can view reviews
  - **Prerequisites:** The staff member is logged in

- Use Case: Consulting medical information
  - **Description:** Staff can look at a client's medical information
  - **Prerequisites:** The staff member is logged in and the client has uploaded a valid medical sheet
  - **Post-conditions:** Working hours are stored in the system
  
- Use case: Managing vouchers
  - **Description:** Staff can create, edit and delete vouchers
  - **Prerequisites:** The staff member is logged in
  - **Post-conditions:** Customer receives promotions in his/her mailbox
  
- Use Case: Log consulting
  - **Description:** Staff can view logs of all bookings
  - **Prerequisites:** The staff member is logged in



### 5.1.1.3 Use Case Descriptions (company)

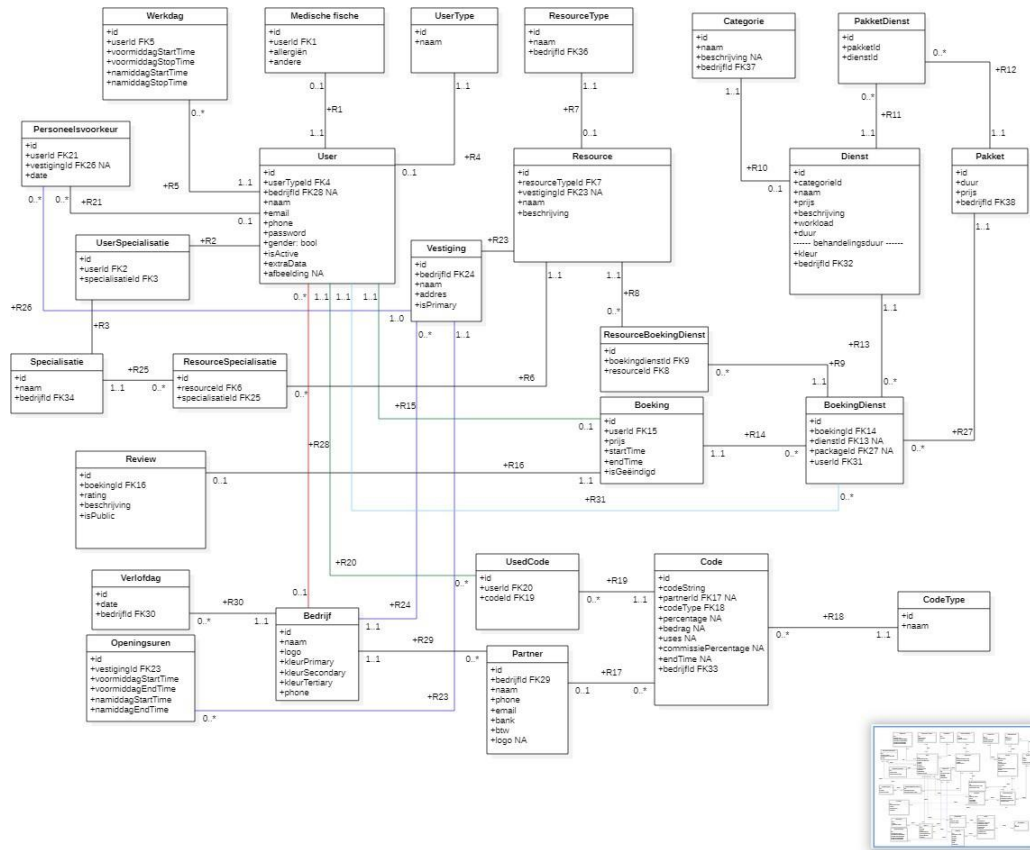
- Use Case: Managing Staff
  - **Description:** The company can create, edit and delete personnel
  - **Prerequisites:** The company is logged in
  - **Post-conditions:** Staff accounts are stored in the system.  
Afterwards, personnel on the platform can
  
- Use case: Managing resources
  - **Description:** The company can create, edit and delete resources
  - **Prerequisites:** The company is logged in
  - **Post-conditions:** Resources are stored in the system
  
- Use Case: Managing promotional codes
  - **Description:** The company can create, edit and delete promotional codes
  - **Prerequisites:** The company is logged in
  - **Post-conditions:** Promotion codes are stored in the system and can be used by the end-user
  
- Use Case: Manage Committees
  - **Description:** The company can create, edit and delete commission codes
  - **Prerequisites:** The company is logged in, and a valid partner must exist in the system to which this code can be linked
  - **Post-conditions:** Commission codes are stored in the system. If a customer uses these codes, the linked partner will receive a portion of the amount

- Use case: managing services
  - **Description:** The company can create, edit and delete services
  - **Prerequisites:** The company is logged in
  - **Post-conditions:** Services are stored in the system. Company price lists are automatically updated
  
- Use case: Managing packages
  - **Description:** The company can create, edit and delete packages.
  - **Conditions:** The company is logged in, and valid services must exist in the system
  - **Post-conditions:** Packages are stored in the system
  
- Use case: Managing opening hours
  - **Description:** The company can create or edit opening hours
  - **Prerequisites:** The company is logged in
  - **Post-conditions:** Opening hours are stored in the system
  
- Use Case: Managing a business
  - **Description:** The company can modify company data
  - **Prerequisites:** The company is logged in
  - **Post-conditions:** Tool is adapted to company's preference

#### 5.1.1.4 Use Case Descriptions (tool).

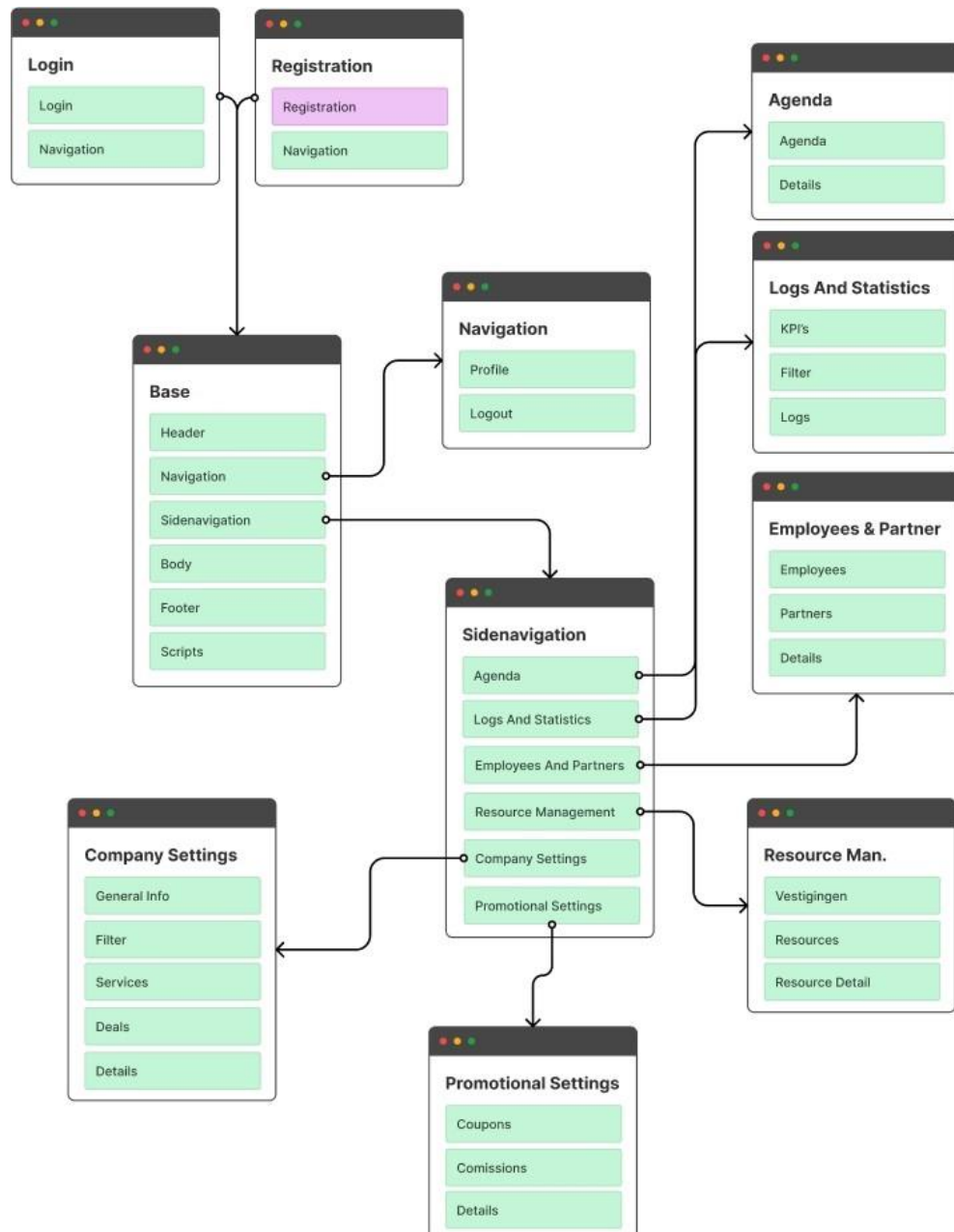
- Use Case: Calendar Optimization
  - **Description:** The system will automatically search for possible optimizations in the calendar by moving bookings. If an optimization is found then the system will automatically send an email to the respective client to see if the booking can be moved. If the customer agrees, the system will perform optimizations.
  - **Conditions:** There are entries in the system
  - **Post-conditions:** The agenda has been optimized
  - **Alternative flows:**
    - Customer does not agree to move booking
      - System will send mail to the next customer, if they agree to the move, this customer's appointment will be rescheduled. The system will continue like this indefinitely.
    - Customer agrees but does not show up
      - Staff member will enter this into the system as "no-show." An email will then be sent to the customer who did not show up. An email is also sent to the next customer to see if they would like to reschedule their appointment.
    - Staff member is absent
      - This is indicated in the system and the customers for the absent staff member are all informed via email. Optimization is still being carried out to see if another staff member can handle the workload

## 5.1.2 Database design



### 5.1.3 Sitemap

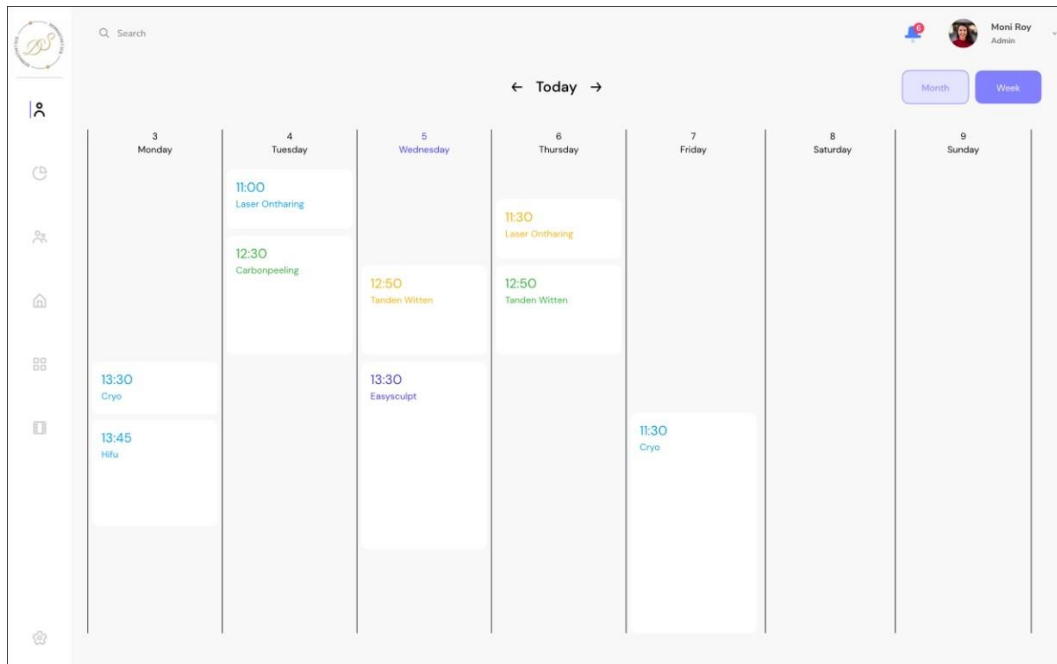
Figma was used to create the sitemap.



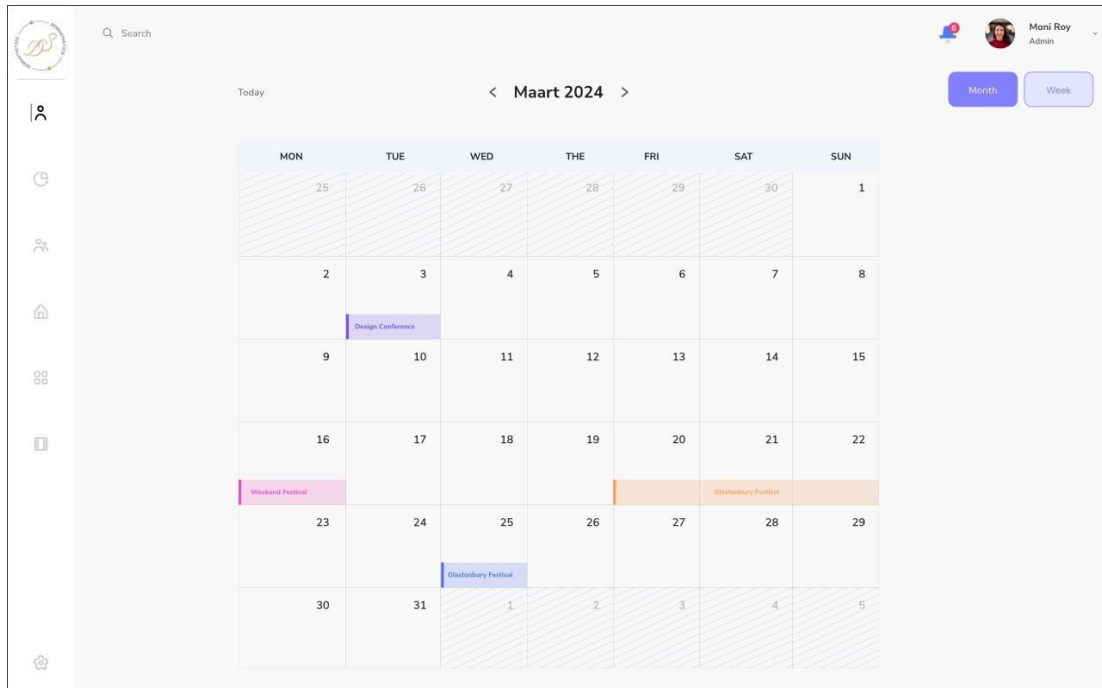
### 5.1.4 Prototypes

Due to the fact that we did not receive the project brief until the end of week 3, I decided to make prototypes right away, to make sure there would be no lack of time. To create the prototypes, Figma was used

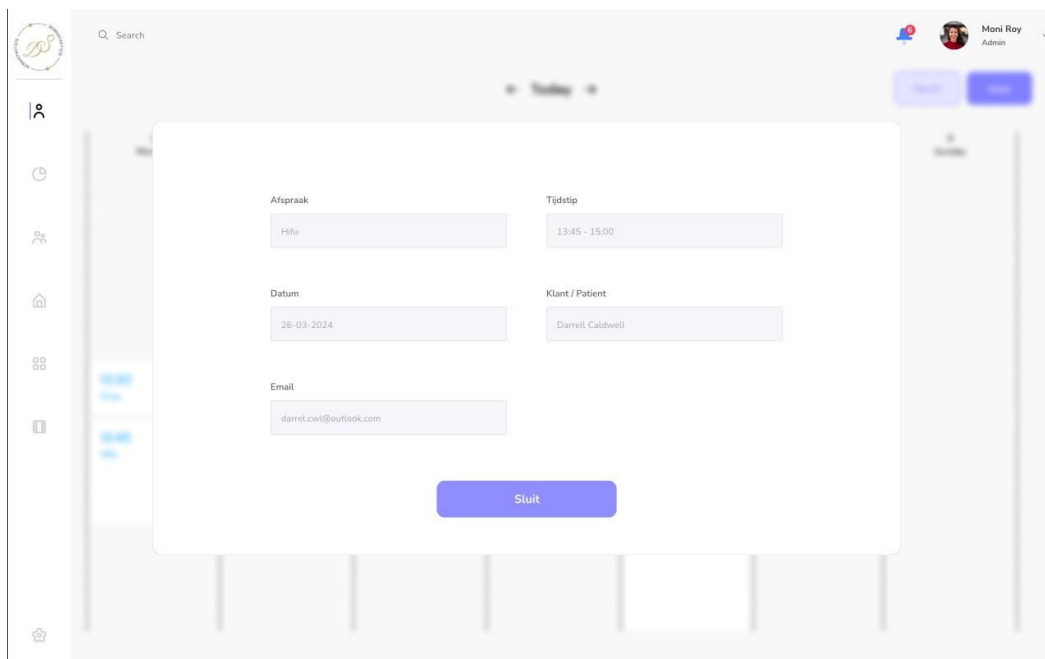
#### 5.1.4.1 Calendar (Week View)



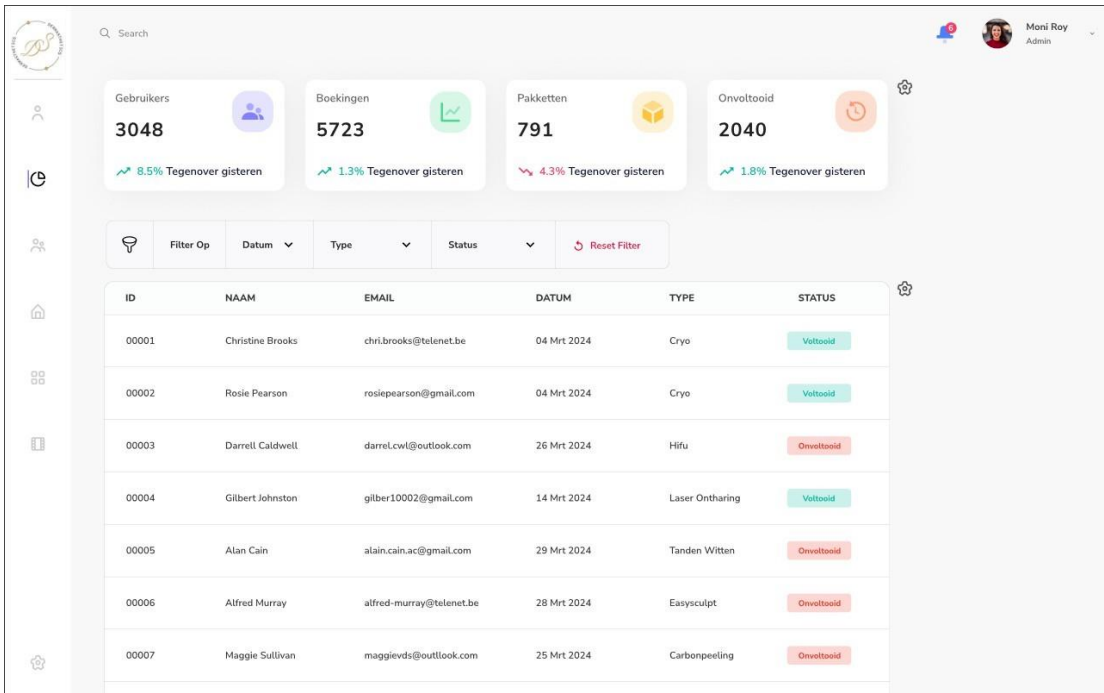
### 5.1.4.2 Calendar (Month view)



### 5.1.4.3 Appointment (details)



5.1.4.4 Statistics & logs

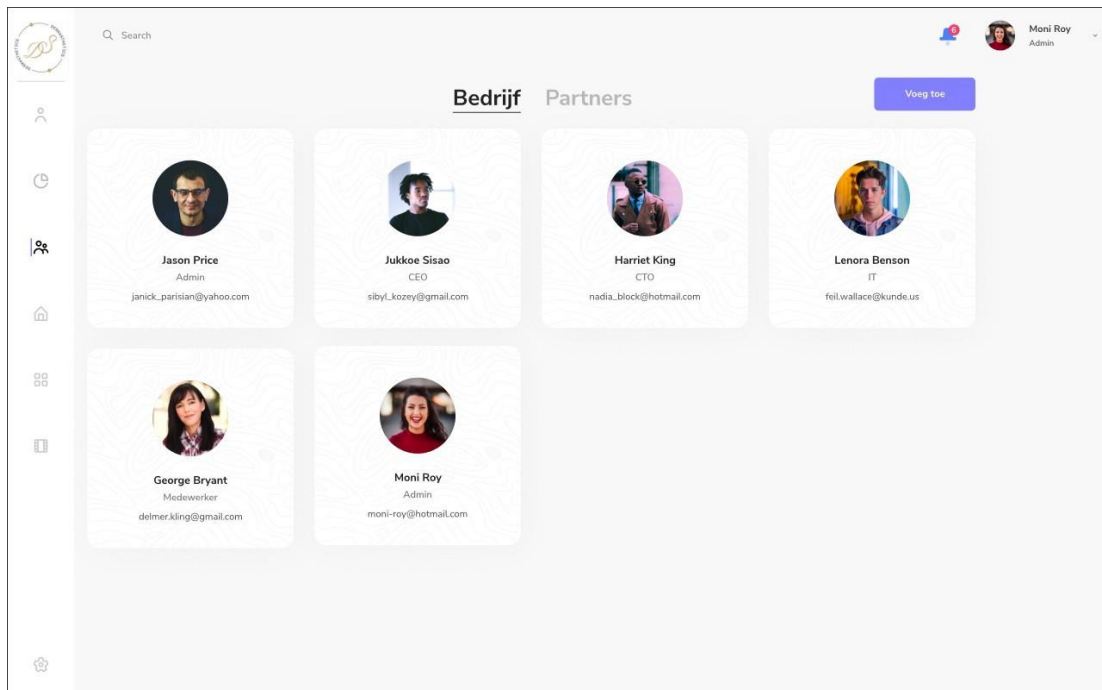


5.1.4.5 Statistics & logs (details)

A modal form is displayed over the dashboard, containing the following fields: **Afspraak** (Hifu), **Tijdstip** (13:45 - 15:00), **Datum** (26-03-2024), **Klant / Patient** (Darrell Caldwell), and **Email** (darrel.cw1@outlook.com). A blue **Sluit** button is at the bottom.



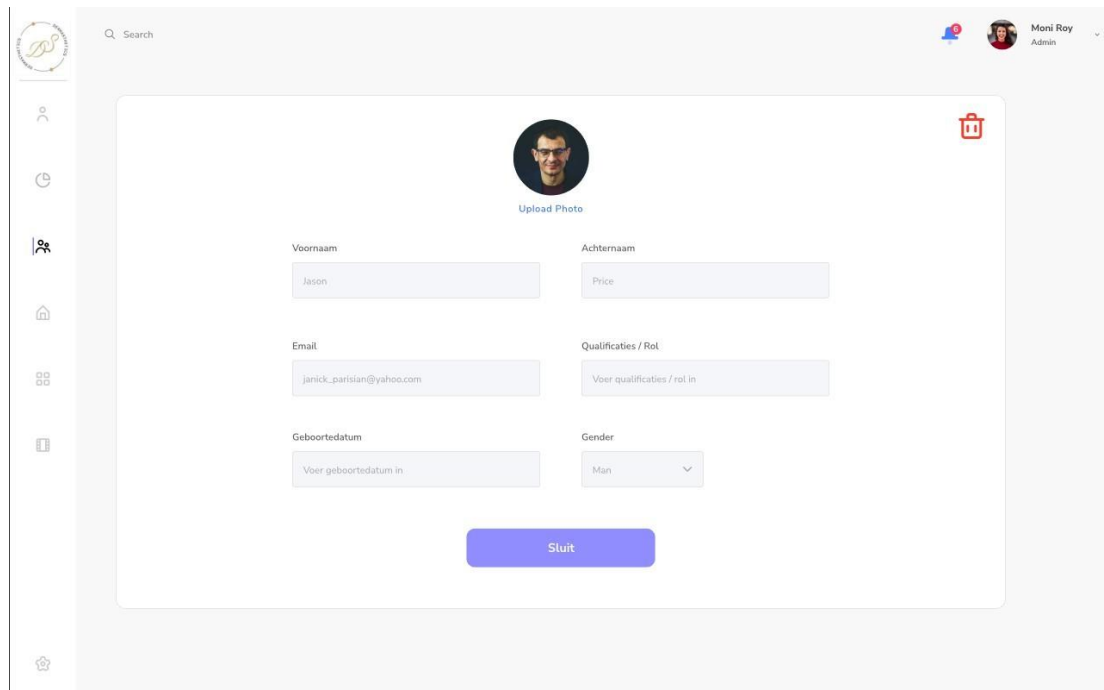
#### 5.1.4.6 Contacts



#### 5.1.4.7 Contacts (create)

The screenshot shows the 'Contacts (create)' form within the same web application. The form is presented in a clean, white box with a light gray border. At the top of the form is a circular placeholder for a profile picture with a camera icon and the text 'Upload Photo'. Below this, the form is organized into two columns of input fields. The left column contains fields for 'Voornaam' (First Name), 'Email', and 'Geboortedatum' (Date of Birth). The right column contains fields for 'Achternaam' (Last Name), 'Qualificaties / Rol' (Qualifications / Role), and 'Gender'. Each field has a placeholder text indicating where to enter the information. At the bottom center of the form is a prominent blue button labeled 'Voeg Toe' (Add). The application's sidebar and top navigation elements are visible in the background, consistent with the previous screenshot.

### 5.1.4.8 Contacts (details)



Search

Moni Roy Admin

Upload Photo

Voornaam: Jason

Achternaam: Price

Email: janick\_partisan@yahoo.com

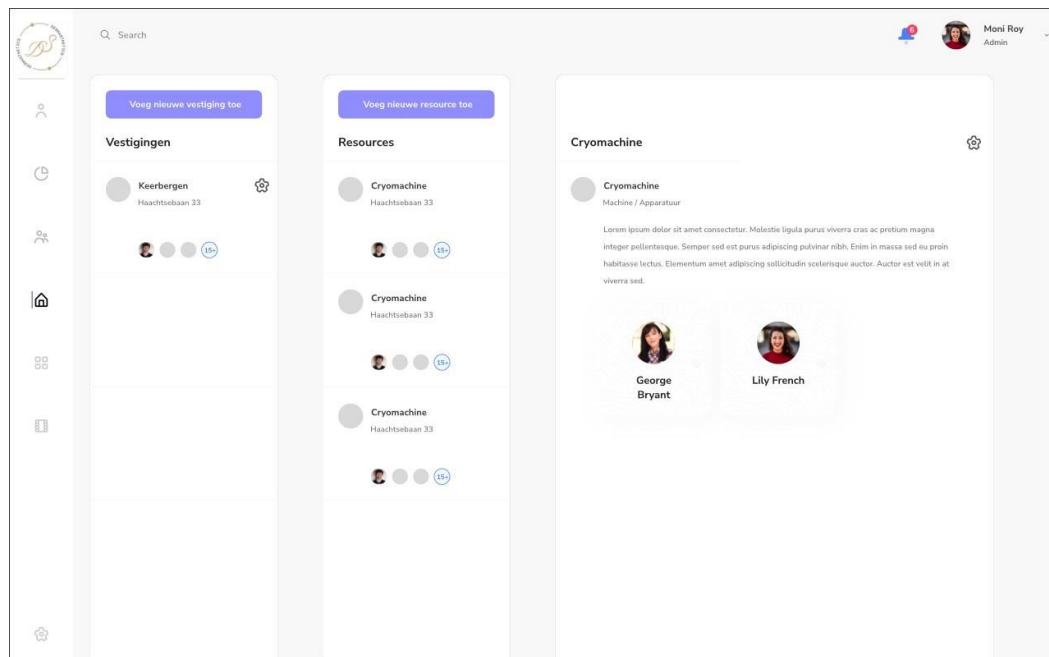
Qualifications / Rol: Voer qualifications / rol in

Geboortedatum: Voer geboortedatum in

Gender: Man

Sluit

### 5.1.4.9 Resources



Search

Moni Roy Admin

Voeg nieuwe vestiging toe

Vestigingen

Keerbergen Haachtsebaan 33

Voeg nieuwe resource toe

Resources

Cryomachine Haachtsebaan 33

Cryomachine Haachtsebaan 33

Cryomachine Haachtsebaan 33

Cryomachine

Machine / Apparatuur

Lorem ipsum dolor sit amet consectetur. Malectis ligula purus viverra cras ac primum magna integer pellentesque. Sempet sed est purus adipiscing pulvinar nibh. Enim in massa sed eu proin habitasse lectus. Elementum amet adipiscing sollicitudin scelerisque auctor. Auctor est velit in at viverra sed.

George Bryant

Lily French

#### 5.1.4.10 Resources (details)

Naam: Cryomachine

Type: Machine / Apparatuur

Beschrijving: Lorem Ipsum ...

Specificaties: Medewerker

Sluit

#### 5.1.4.11 Resources (create branch)

Naam: Keerbergen

Adres: Haachtsebaan 33, Keerbergen

Openingsuren: Maandag, Dinsdag, Woensdag, Donderdag

Vrijdag, Zaterdag, Zondag

Sluit

### 5.1.4.12 Company Settings

**Dermasthetics**  
 Haachtsebaan 33, Keerbergen  
 Tel: 015 69 03 60  
 IG: dermasthetics4you  
 FB: DermastheticsByTaoStudio

**Diensten** Pakketten

**Cryo**  
 €75.00  
 Bewerk Dienst

### 5.1.4.13 Company settings (details)

**Naam**  
 Dermasthetics

**Instagram**  
 dermasthetics4you

**Telefoon**  
 015 69 03 60


**Facebook**  
 DermastheticsByTaoStudio

**Categorieën**  
 Machine Machine Voeg toe

**Specialisaties**  
 Cryo Hifu Voeg toe

**Sluit**

5.1.4.14 Promotional codes



🔍 Search

👤

Moni Roy

Admin

🏠

📁

📄

📊

📅

🔖

🔒


Coupons

Partners

Voeg toe

ID	HOEVEELHEID	CODE	BEDRAG	DATUM	TYPE	STATUS
00001	5	9280148220	15€	14 Apr 2024	Coupon	Actief
00002	1	802021029302	100€	20 Apr 2024	Giftcard	Actief
00003	20	3930202230	20%	14 Feb 2019	Coupon	Inactief

5.1.4.15 Promotional codes (partners)



🔍 Search

👤

Moni Roy

Admin

🏠

📁

📄

📊

📅

🔖

🔒

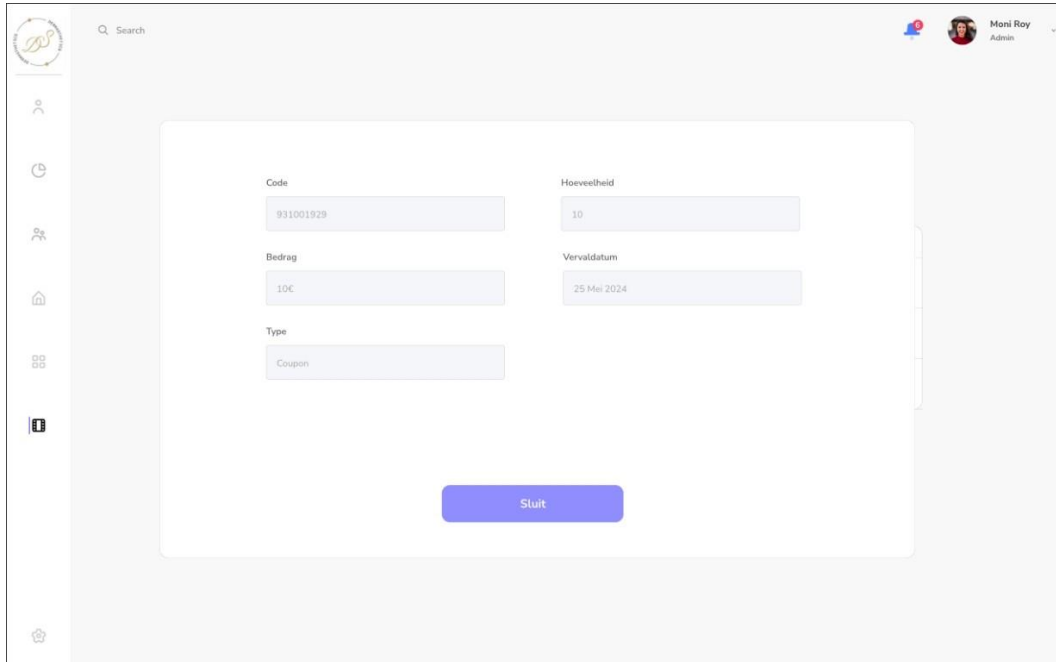
Coupons

Partners

Voeg toe

ID	PARTNER	CODE	KORTING	DATUM	COMMISSIE	STATUS
00004	Spot Group	2003944920	20%	14 Apr 2024	20%	Actief
00005	LHS	220209120	15%	22 May 2024	10%	Actief
00006	Darrell Caldwell	3930202230	40%	14 Feb 2019	20%	Inactief

## 5.1.4.16 Promotion codes (details)



The screenshot shows a web application interface for editing a promotion code. The interface includes a sidebar with navigation icons, a search bar, and a user profile in the top right corner. The main content area contains a form with the following fields:

Code	Hoeveelheid
931001929	10

Bedrag	Vervaldatum
10C	25 Mei 2024

Type
Coupon

At the bottom of the form is a blue button labeled "Sluit".

#### 5.1.4.17 Booking

**Afspraak**

Hifu

← Today →

Month Week

3 Monday	4 Tuesday	5 Wednesday	6 Thursday	7 Friday	8 Saturday	9 Sunday
11:00 Book	11:00 Bezet		11:30 Bezet			
11:30 Book	12:30 Bezet		12:50 Bezet			
12:00 Book		12:50 Bezet				
		13:30 Bezet				
13:45 Bezet				11:30 Bezet		

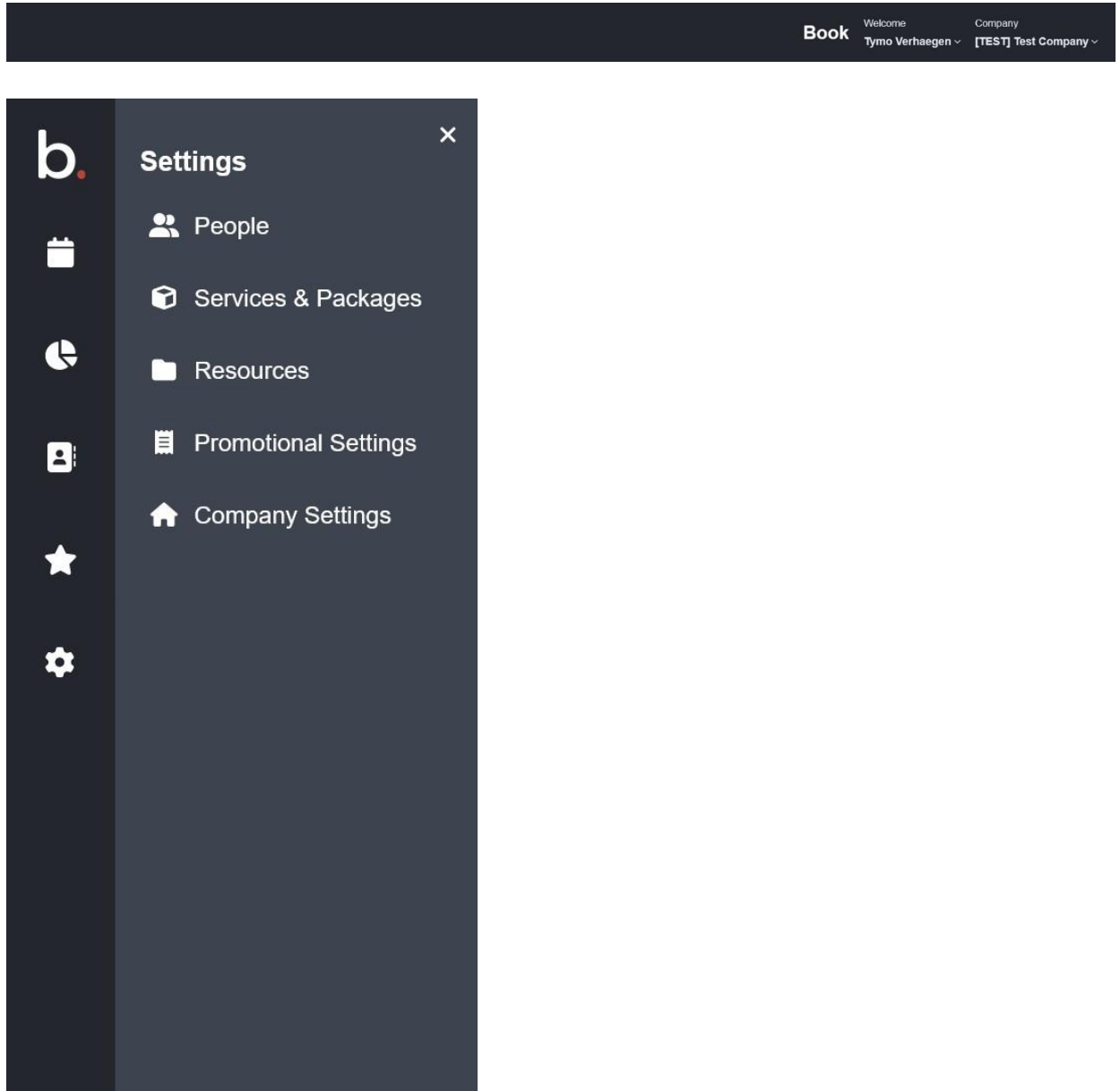
#### 5.1.4.18 Booking (cont.)

## 5.2 Front end

The front-end part of our application was created in Django & HTML, Tailwind CSS was used for styling and Javascript was used for animations, pop-up windows and more. There are both dark and light modes for the webapp.

### 5.2.1 Navigation

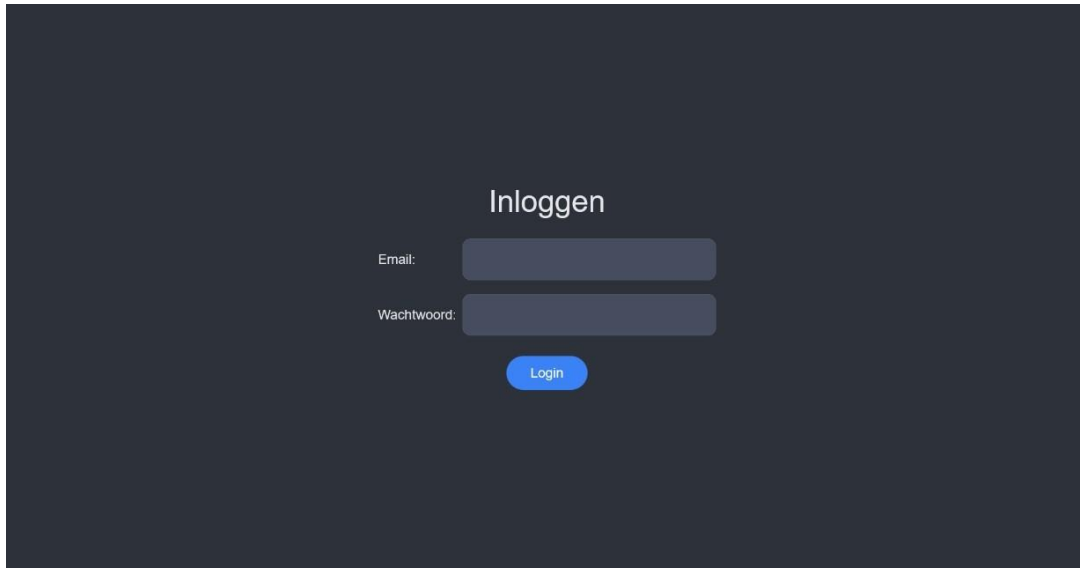
Developed by myself and Joppe Kerkhofs





### 5.2.2 Login

Developed by Joppe Kerkhofs



Inloggen

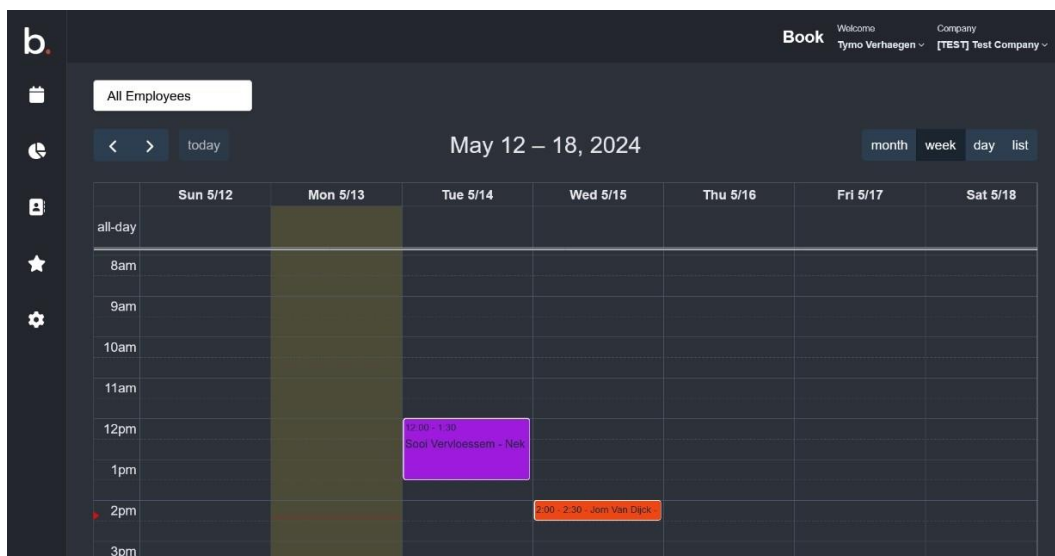
Email:

Wachtwoord:

Login

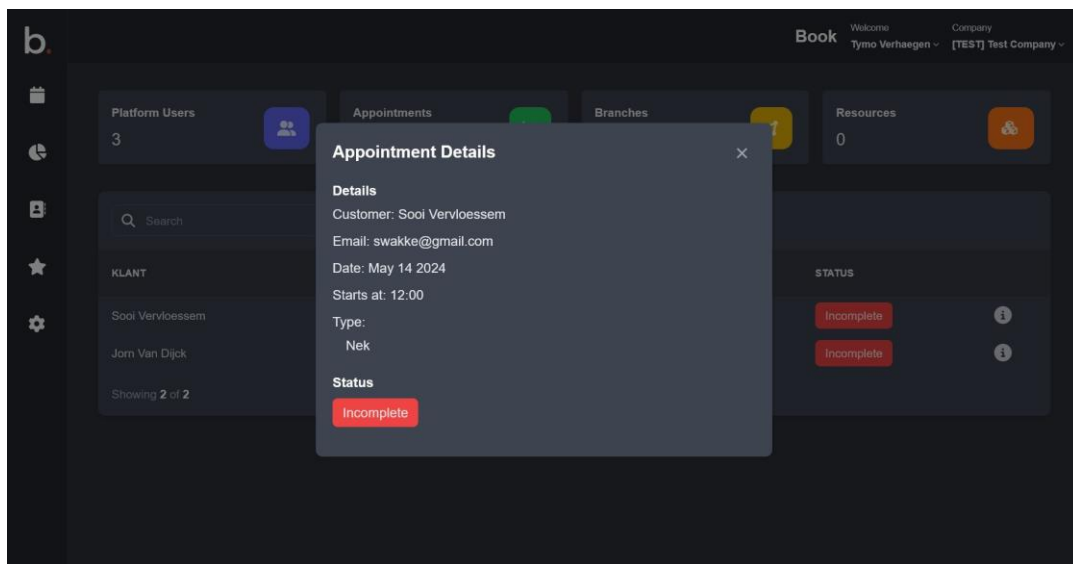
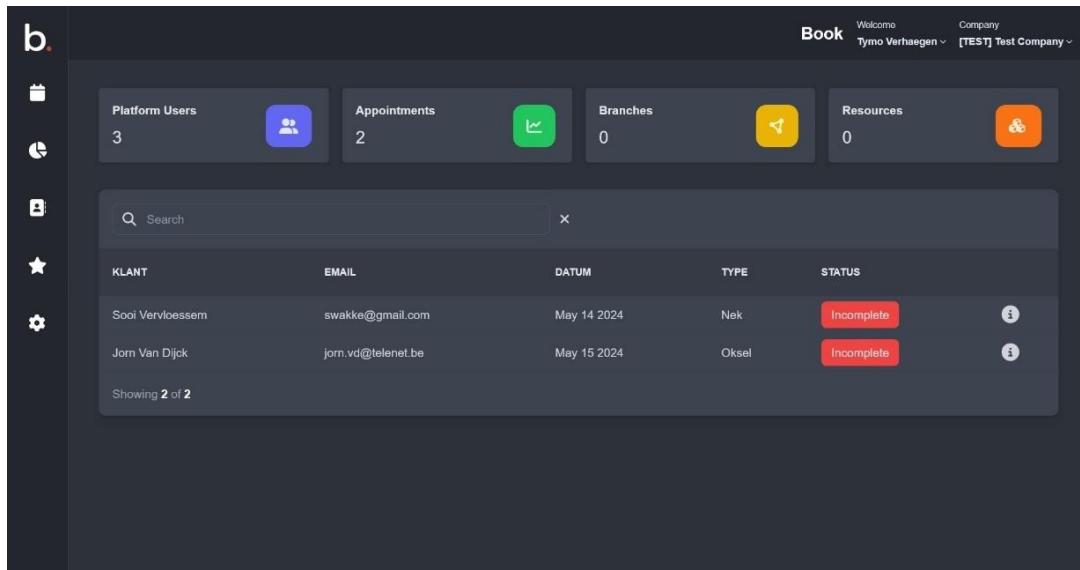
### 5.2.3 Agenda

Developed by Thomas Verbruggen



## 5.2.4 Statistics & logs

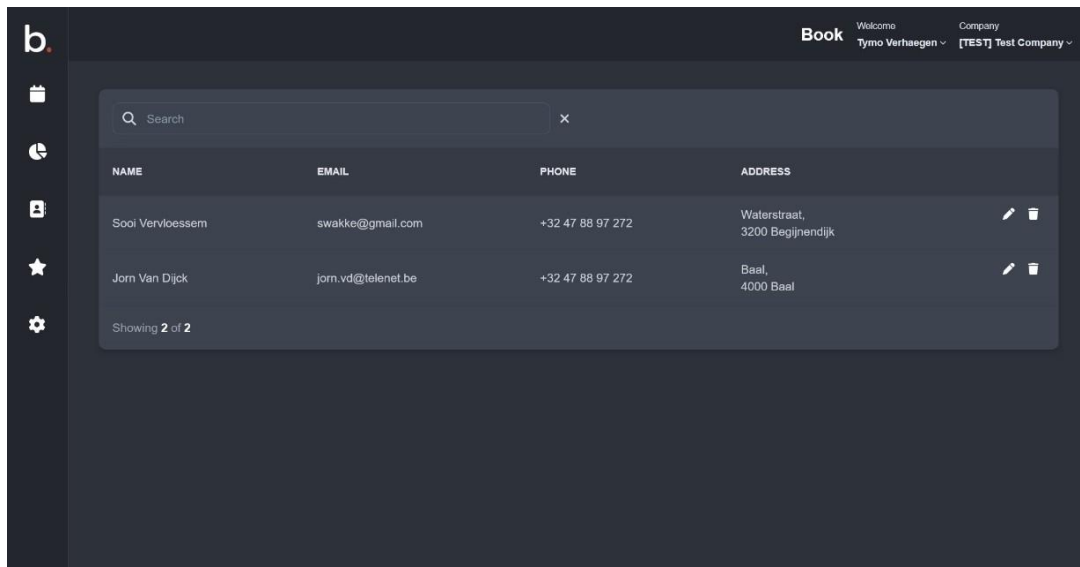
I developed this entire page, both front-end and backend.



## 5.2.5 Customers

Developed by Joppe Kerkhofs and myself

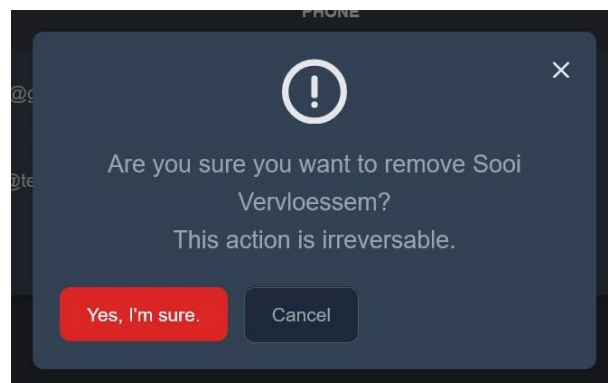
### Customer view



### Update customer

The screenshot shows a form titled 'Update - Sooi Vervloessem' with a close button 'X'. The form has several input fields for customer information. The fields are arranged in a grid: FIRST NAME (Sooi), LAST NAME (Vervloessem), EMAIL (swakke@gmail.com), PHONE (+32 47 88 97 272), ADDRESS (Waterstraat), ZIPCODE (3200), and CITY (Begijnendijk). At the bottom of the form, there is a blue button labeled 'Update Customer'.

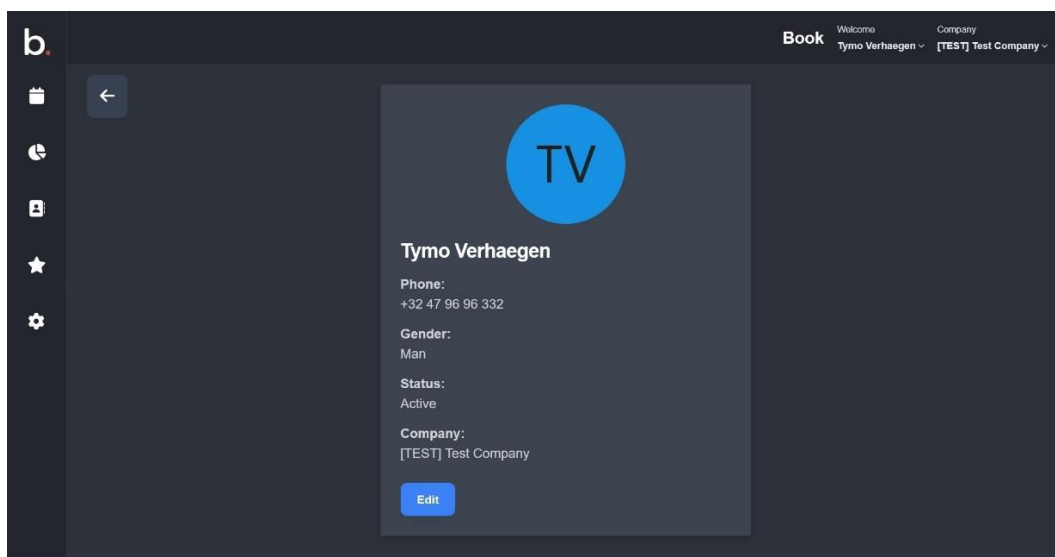
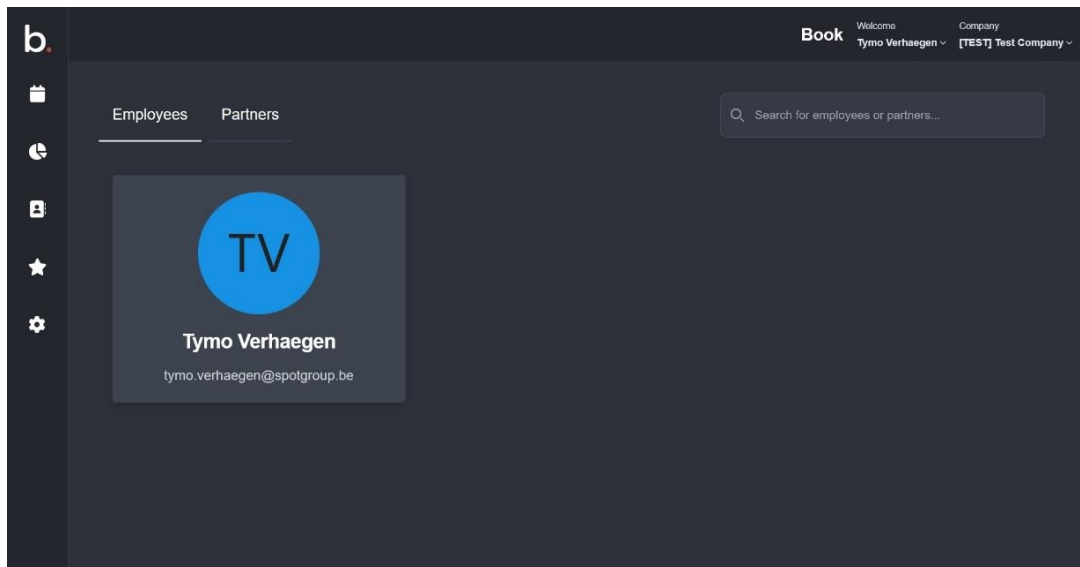
### Delete customer



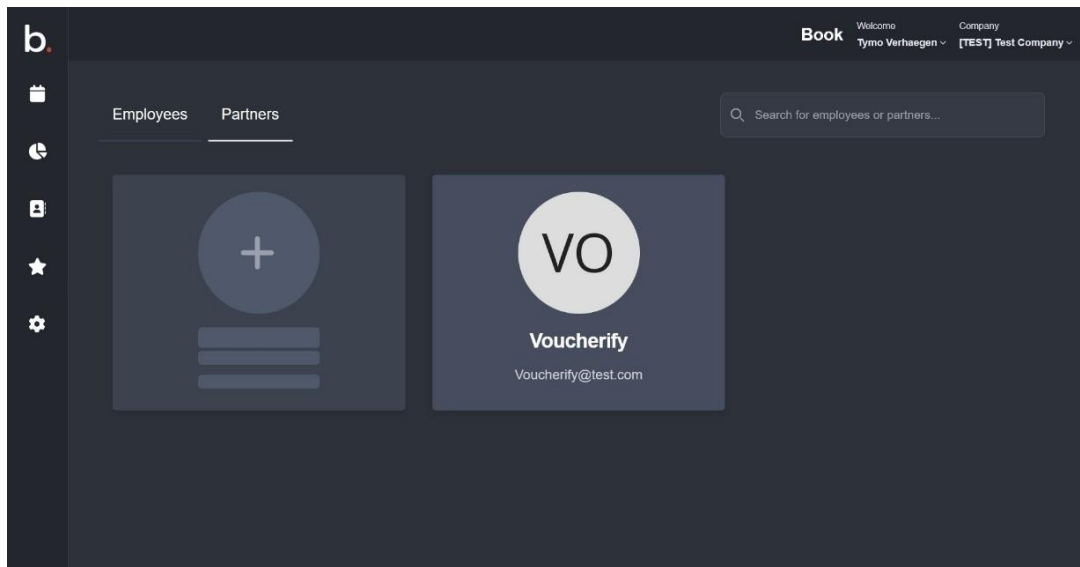
## 5.2.6 Employees

Developed by Joppe Kerkhofs and myself

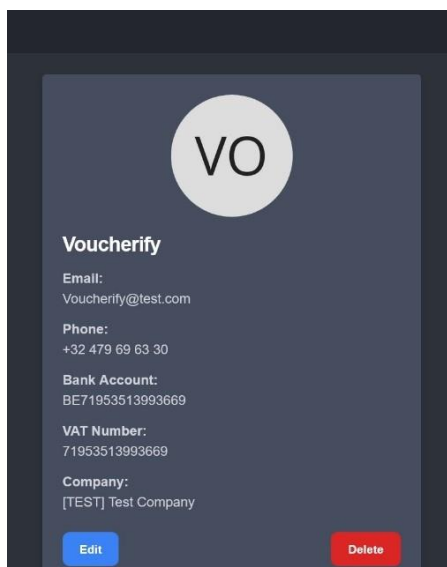
### Employee tab



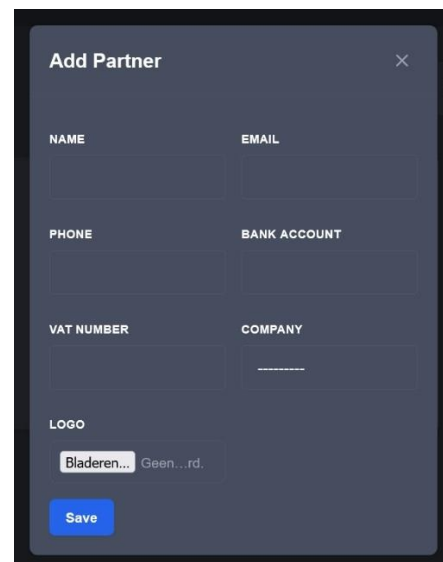
## Partner tab



## Partner info

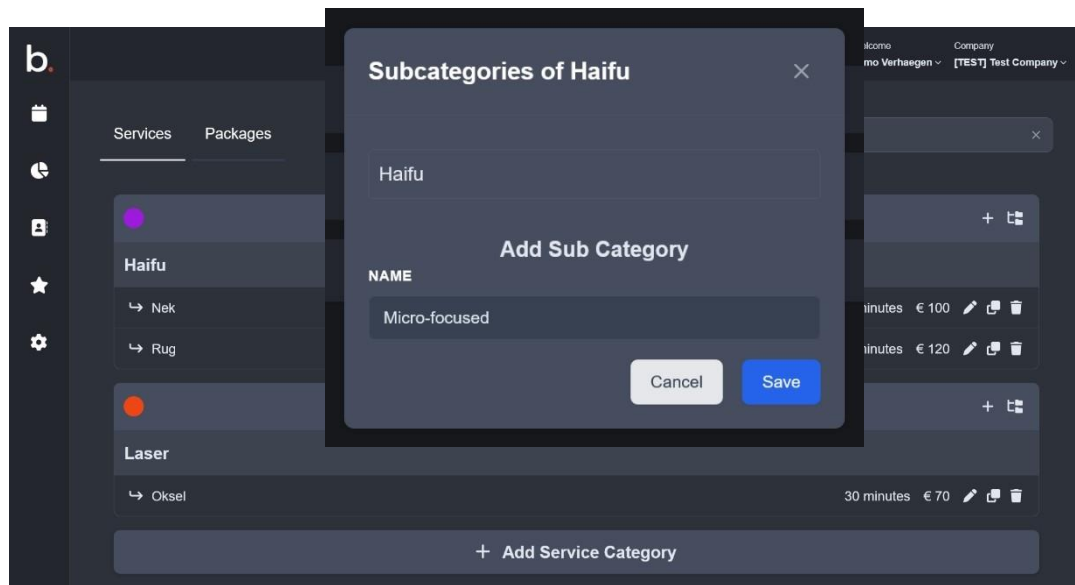


## Partner add



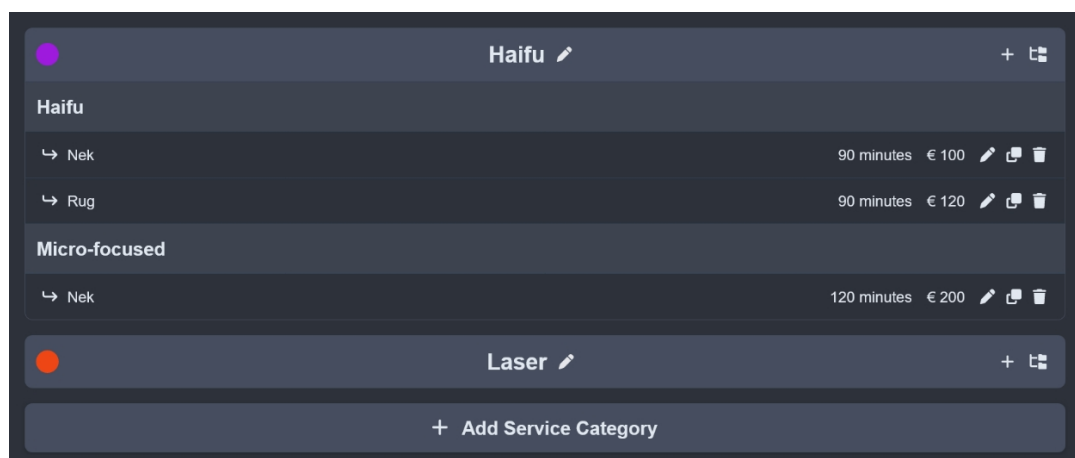
## 5.2.7 Services & packages Developed

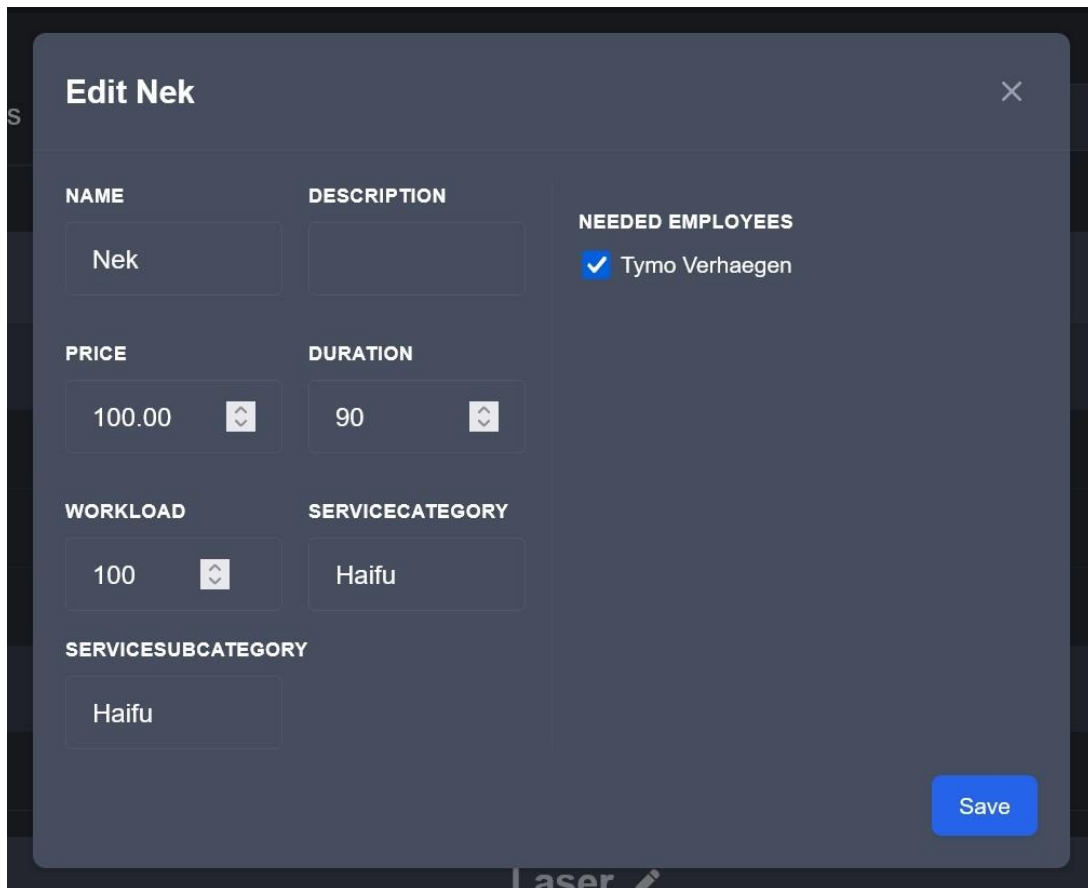
by Joppe Kerkhofs and myself **Main view**



**Create subcategory**

## Multiple subcategories

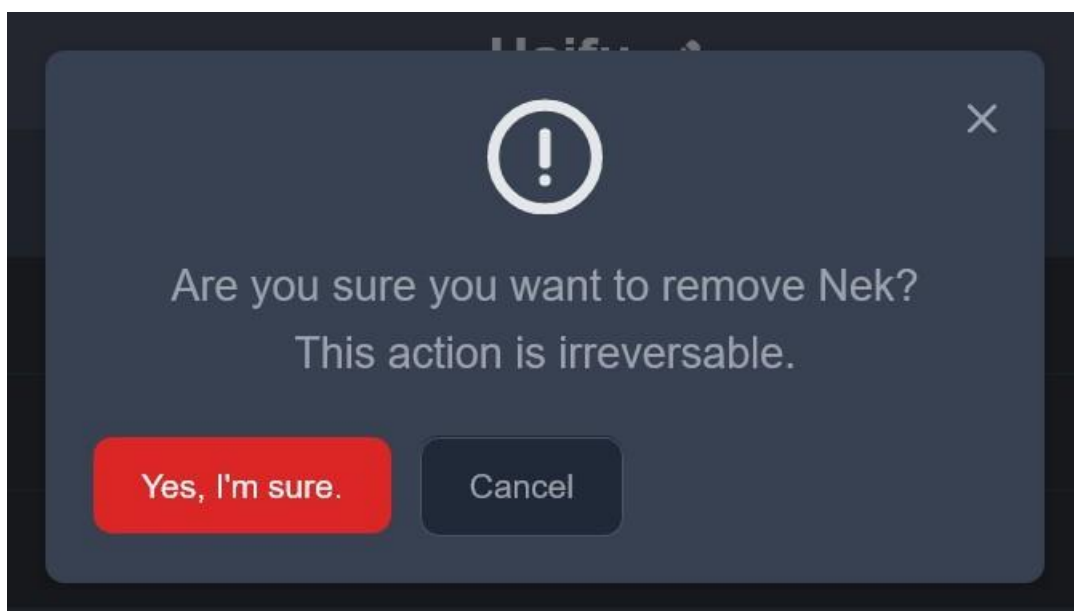


**Edit on duty**

The 'Edit Nek' dialog box is a dark-themed window with a title bar containing the text 'Edit Nek' and a close button (X). The dialog is divided into several sections for editing service details:

- NAME:** A text input field containing 'Nek'.
- DESCRIPTION:** An empty text input field.
- NEEDED EMPLOYEES:** A section with a blue checkmark icon and the text 'Tymo Verhaegen'.
- PRICE:** A text input field containing '100.00' with a small up/down arrow icon to its right.
- DURATION:** A text input field containing '90' with a small up/down arrow icon to its right.
- WORKLOAD:** A text input field containing '100' with a small up/down arrow icon to its right.
- SERVICECATEGORY:** A text input field containing 'Haifu'.
- SERVICSUBCATEGORY:** A text input field containing 'Haifu'.

A blue 'Save' button is located in the bottom right corner of the dialog.

**Removal of service**

The 'Removal of service' dialog box is a dark-themed window with a title bar containing a close button (X). It features a large white exclamation mark icon in a circle at the top center. The text inside the dialog reads:

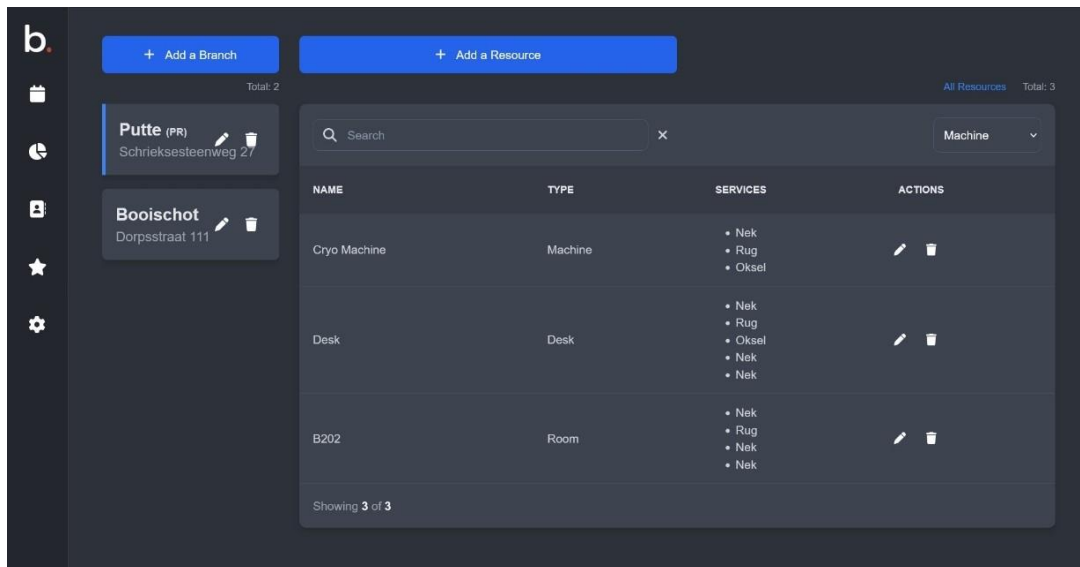
Are you sure you want to remove Nek?  
This action is irreversable.

At the bottom, there are two buttons: a red button labeled 'Yes, I'm sure.' and a dark gray button labeled 'Cancel'.

## 5.2.8 Resources

Developed by the whole group, Thomas Verbruggen, Joppe Kerkhofs and myself

### Main view

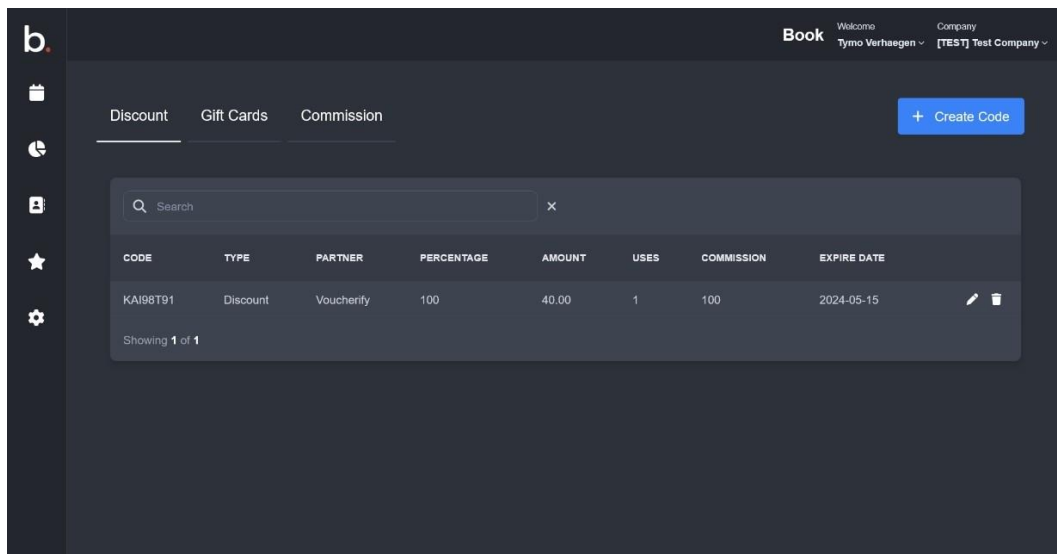


### Add resource

The screenshot shows the 'Add Resource' modal form. It has a title bar with 'Add Resource' and a close button. The form is divided into several sections: 'NAME' and 'DESCRIPTION' (with a plus sign between them), 'RESOURCE TYPE' and 'BRANCH', and a 'SERVICE' section with a list of checkboxes for 'Nek', 'Rug', 'Oksel', 'Nek', and 'Nek'. At the bottom is a blue 'Add Resource' button.

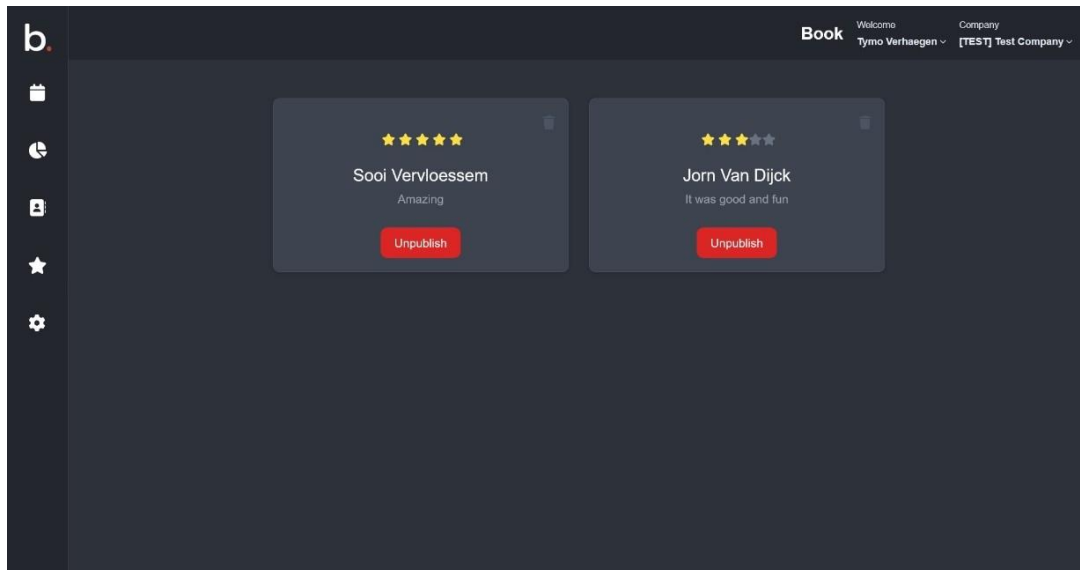


Developed by Thomas Verbruggen and myself



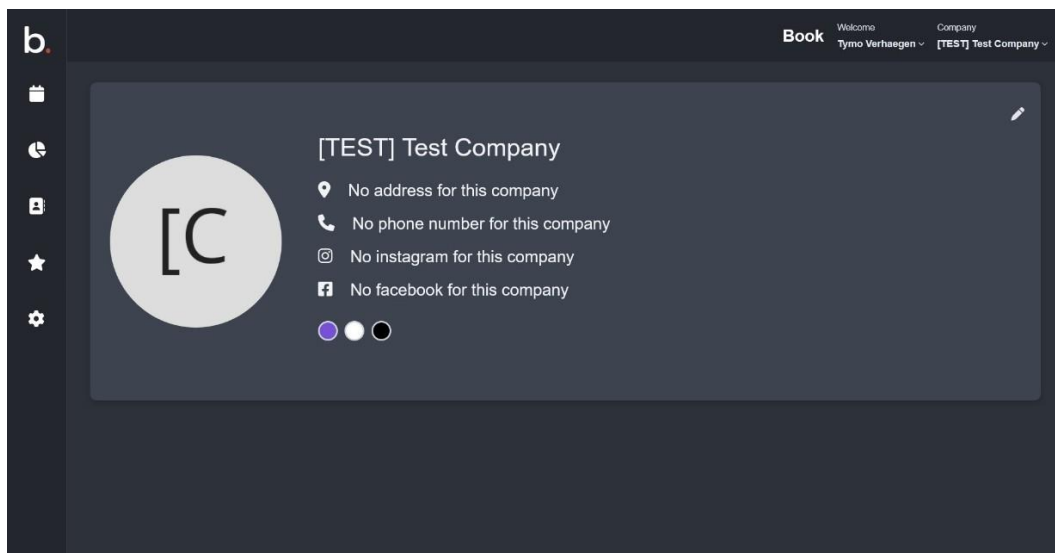
### 5.2.10 Reviews

Developed by Thomas Verbruggen



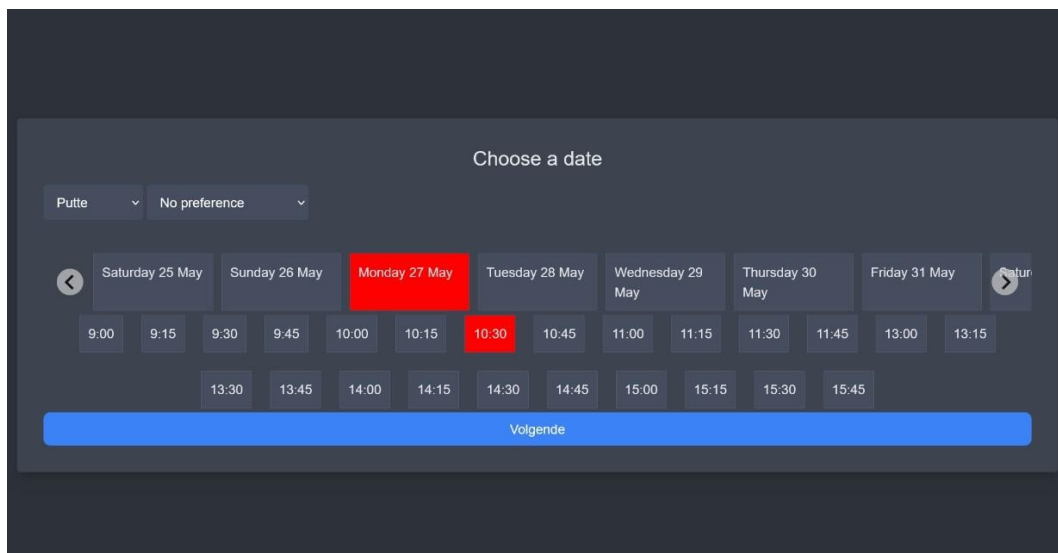
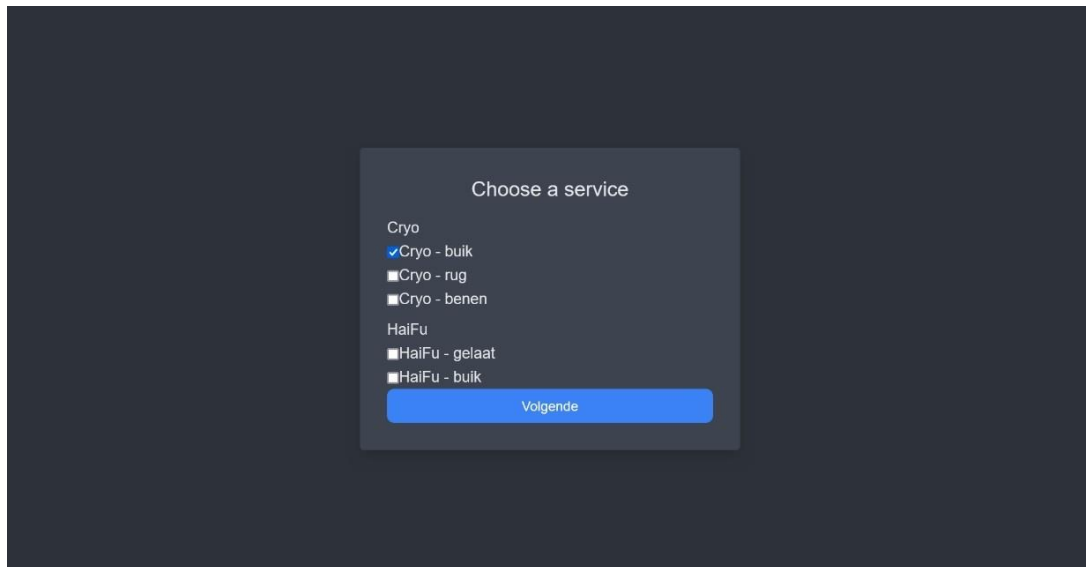
### 5.2.11 Company Settings

Developed by Thomas Verbruggen and myself



### 5.2.12 Booking (end user)

Developed by the whole group, Thomas Verbruggen, Joppe Kerkhofs and myself



Persoonlijke informatie

Tes

test

test@test.com

test

Factuur E-mail

test

3000

test

Opmerking

Betalen

## 5.3 Backend

The backend part of our web application is designed with Django views and all data is written and stored in a PostgreSQL database.

### 5.3.1 Navigation

Navigation is controlled by an `urls.py` file, which contains all possible links that an end-user can visit

Developed by Joppe Kerkhofs and myself

```
urlpatterns = [
    # Redirects to agenda page
    path('', RedirectView.as_view(url='/agenda/', permanent=False), name='home'),

    # First login page
    path('first_login/', views.FirstLoginView.as_view(), name='firstlogin'),

    # AGENDA
    path('agenda/', views.AgendaView.as_view(), name='agenda'),
    path('event/<uid:event_id>', views.EventDetailView.as_view(), name='event_details'),
    path('update_appointment_field/<uid:event_id>', views.update_appointment_field.as_view(), name='updateAppointment'),

    # COMPANY
    path('company/', views.CompanyView.as_view(), name='company'),
    path('company/edit/', views.EditCompanyView.as_view(), name='updateCompany'),

    # PEOPLE
    path('people/', views.PeopleView.as_view(), name='people'),
    # Employee CRUD operations
    path('people/user/info/<uid:user_id>', views.UserInfoView.as_view(), name='updateEmployee'),
    # Partner CRUD operations
    path('people/partner/add/', views.PartnerAddView.as_view(), name='addPartner'),
    path('people/partner/info/<uid:id>', views.PartnerView.as_view(), name='updatePartner'),
    path('people/partner/remove/<uid:id>', views.PartnerRemoveView.as_view(), name='removePartner'),
    # Customer CRUD operations
    path('customers/', views.CustomersView.as_view(), name='customers'),
    path('customer/update/<uid:pk>', views.UpdateCustomerView.as_view(), name='updateCustomer'),
    path('customer/remove/<uid:pk>', views.DeleteCustomerView.as_view(), name='removeCustomer'),

    # RESOURCES
    path('resources/', views.ResourcesView.as_view(), name='resources'),
    # Branch CRUD operations
    path('resources/<uid:branch_id>', views.EditBranchView.as_view(), name='updateBranch'),
    path('resources/', views.ResourcesView.as_view(), name='addBranch'),
```

```
path('resources/<uid:branch_id>', views.EditBranchView.as_view(), name='updateBranch'),
path('resources/', views.ResourcesView.as_view(), name='addBranch'),
path('resources/branch/remove/<uid:id>', views.RemoveBranchView.as_view(), name='removeBranch'),
# Resource CRUD operations
path('resources/info/<uid:id>', views.EditResourceView.as_view(), name='updateResource'),
path('resources/', views.ResourcesView.as_view(), name='addResource'),
path('resources/remove/<uid:id>', views.RemoveResourceView.as_view(), name='removeResource'),
path('resources/name/add', views.ResourceNameView.as_view(), name='resourceName'),
path('resources/name/update/<uid:id>', views.UpdateResourceNameView.as_view(), name='updateResourceName'),
path('resources/name/remove/<uid:id>', views.RemoveResourceNameView.as_view(), name='removeResourceName'),

# SERVICES
# Service CRUD operations
path('services/', views.ServicesView.as_view(), name='services'),
path('services/update/<uid:id>', views.EditServiceView.as_view(), name='updateService'),
path('services/remove/<uid:id>', views.DeleteServiceView.as_view(), name='removeService'),
path('services/duplicate/<uid:serviceId>', views.ServiceDuplicateView.as_view(), name='duplicateServices'),
# Service Category
path('services/category/', views.ServiceCategoryView.as_view(), name='serviceCategory'),
path('services/subcategories/', views.ServiceSubCategoryView.as_view(), name='serviceSubCategory'),
path('services/subcategories/update/<uid:id>', views.UpdateServiceSubCategoryView.as_view(), name='updateServiceSubCategory'),
path('services/subcategories/remove/<uid:id>', views.RemoveServiceSubCategoryView.as_view(), name='removeServiceSubCategory'),
# Package CRUD operations
path('packages/add/', views.AddPackageView.as_view(), name='addPackage'),
path('packages/update/<uid:id>', views.EditPackageView.as_view(), name='updatePackage'),
path('packages/remove/<uid:id>', views.DeletePackage.as_view(), name='removePackage'),

# PROMOTIONS
path('promotions/', views.PromotionsView.as_view(), name='promotions'),
path('promotions/create/', views.CreatePromotionView.as_view(), name='createCode'),
path('promotions/update/<uid:id>', views.EditPromotionView.as_view(), name='updateCode'),
path('promotions/remove/<uid:id>', views.DeletePromotionView.as_view(), name='removeCode'),

# STATISTICS
path('statistics/', views.StatisticView.as_view(), name='statistics'),
```

### 5.3.2 Login

Developed by Joppe Kerkhofs

#### FirstLoginView:

- Functionality:
  - Render view when user first logs in
  - A POST request processes the user's entered data
  - If the form is valid
    - Checks if UserInfo already exists
    - For existing UserInfo, the pre-existing information is updated
    - With new UserInfo, a new user is created
    - Companies are loaded via the method *load\_companies()*
    - The user is redirected to the calendar page
  - The *load\_companies()* method loads companies into the database by checking for CloudSpot companies and adding them to the *CompanyInfo* model
  - When context data is retrieved, a form is added to the context

### 5.3.3 Agenda

Developed by Thomas Verbruggen

#### AgendaView:

- Functionality:
  - Render view for calendar page
  - On a GET request, it checks if the user is logging in for the first time using the *check\_first\_login()* function
  - It picks up employees for the company
  - It builds an employee selection element
  - It collects appointments and associated services from the database and applies calculations to determine the duration and end time of appointments

- It filters appointments on selected employees and prepares context data to send to the template

#### **EventDetailsView:**

- Functionality:
  - This view renders the detail page of a specific event (appointment) within the calendar
  - On a GET request, it retrieves the information about the selected event and the associated client
  - It collects information about the services associated with the appointment and calculates the total price
  - It forwards the collected data to the template to display

#### **update\_appointment\_field:**

- Functionality:
  - This view processes POST requests to update the status of an appointment, such as "informed late" and "did not show up."
  - It checks that the correct fields are included in the request and updates the status of the appointment
  - It saves the changes to the database and redirects the user to the details page

### 5.3.4 Statistics & logs

Developed by myself

#### **StatisticView:**

- Functionality:
  - This view shows statistics related to appointments, users, branches and resources.
  - With a GET request, it collects the necessary data for statistics.
  - It retrieves the number of users, appointments, branches and resources.
  - It performs a search functionality on appointments based on the search query in the URL.
  - It updates the status of appointments based on the current time, marking appointments as "ended" when the expected end time has passed.
  - Pagination functionality is included to make the bulk of logs more manageable

### 5.3.5 Customers, Employees and Partners

Developed by Joppe Kerkhofs and myself

#### **PeopleView:**

- Functionality:
  - This view manages information related to "people" or agencies such as partners
  - When retrieving context data, it retrieves data from the database, including users, customers, companies and partners
  - It builds a form for adding new partners
  - With a POST request, it processes the submitted form to create a new partner
  - If the form is valid, the new partner is saved and the user is redirected to the Manage People page with a success message



**UserInfoView:**

- Functionality:
  - This view displays and updates user information
  - When retrieving context data, it retrieves user information and adds a form for updating the information
  - For a POST request, it processes the submitted form to update user information
  - If the form is valid, the user information is updated with the new data

**PartnerView:**

- Functionality:
  - This view displays and updates partner information
  - When retrieving context data, it retrieves the partner information and adds a form for updating the information
  - With a POST request, it processes the submitted form to update partner information
  - If the form is valid, the partner information will be updated with the new data

**PartnerRemoveView:**

- Functionality:
  - This view removes a partner
  - On a POST request, the partner is identified and removed from the database

**PartnerAddView:**

- Functionality:
  - This view adds a new partner

- With a valid form, it validates the data and creates a new partner
- If the partner already exists, a message is displayed

#### **CustomerView:**

- Functionality:
  - This view shows a list of customers
  - On a GET request, it retrieves the clients associated with the current user's company
  - It provides functionality for searching customers based on the search query entered
  - Pagination has been added to make a large bulk of customers more manageable

#### **UpdateCustomerView:**

- Functionality:
  - This view updates customer data
  - It uses a form to update customer data
  - With a valid form, it validates the data and updates the customer data

#### **DeleteCustomerView:**

- Functionality:
  - This view removes a customer
  - It displays a confirmation page for removing the client
  - Upon confirming deletion, the customer is deleted and the user is redirected to the list of customers

### 5.3.6 Reviews

Developed by Thomas Verbruggen

#### **ReviewsView:**

- Functionality:
  - This view shows reviews
  - It retrieves a query set of reviews filtered by appointments of the user's company

#### **DeleteReview:**

- Functionality:
  - This view deletes a review.
  - A POST request removes the review.
  - A success message is displayed after deleting the review.

#### **ToggleReviewPublicView:**

- Functionality:
  - This view toggles the public status of a review.
  - With a POST request, the public status of the review is

### 5.3.7 Services & packages

Developed by the whole group, Thomas Verbruggen, Joppe Kerkhofs and myself

#### **ServicesView:**

- Functionality:
  - Retrieves data from database and prepares context for rendering
  - Processes POST requests to add a new service
  - Provides forms for new services and/or packages to be added

#### **ServiceCategoryView:**

- Functionality:
  - Manages the creation or updating of a service category
  - Processes POST requests to add a new service category or update an existing one
  - Displays a success message after successfully adding or updating the service category

#### **ServiceSubCategoryView:**

- Functionality:
  - Manages the creation of a subcategory for services
  - Processes POST requests to add a new subcategory to a given service category
  - Verifies that the required data is present in the POST requests and then adds the new subcategory

**UpdateServiceSubCategoryView:**

- Functionality:
  - Processes POST requests to update an existing service subcategory
  - Verifies that the required data, such as the subcategory ID and name, is present in the POST request
  - Updates the service subcategory name and saves the changes

**RemoveServiceSubCategoryView:**

- Functionality:
  - Processes POST requests to remove a service subcategory
  - Checks whether the subcategory has services before deleting it
  - Remove the service subcategory if it has no services

**ServiceDuplicateView:**

- Functionality:
  - Processes POST requests to duplicate a service
  - Creates a new service with the same data as the original service
  - Also duplicates the UserAvailableService *instances* for the new service
  -

**EditServiceView:**

- Functionality:
  - Processes POST requests to update a service
  - Update the properties of an existing service with the data from the form
  - Also update the UserAvailableService *instances* for the updated service

**DeleteServiceView:**

- Functionality:
  - Processes POST requests to remove a service
  - Deletes the selected service along with all related *UserAvailableService instances*

**AddPackageView:**

- Functionality:
  - Processes POST requests to add a new packet
  - Creates a new package with the submitted data and links selected services to the package

**EditPackageView:**

- Functionality:
  - Processes POST requests to update a package
  - Updates the properties of an existing package with the data from the form and updates the associated services

**DeletePackageView:**

- Functionality:
  - Processes POST requests to remove a packet
  - Deletes the selected package from the database

### 5.3.8 Resources

Developed by the whole group, Thomas Verbruggen, Joppe Kerkhofs and myself

#### **ResourcesView:**

- Functionality:
  - Handles both GET and POST requests to manage resources
  - Retrieves resources and applies filters based on searches, branch and resource type
  - Displays resources in pagination and provides forms for adding new branches and resources
  - Handles adding new branches and resources, and links services to resources
  - Provides forms for both resource and branch creation

#### **EditBranchView:**

- Functionality:
  - Allows you to edit an existing branch office
  - Displays a form for changing affiliate information such as name, address, contact information, etc.
  - Upon submitting the form, the branch is updated with the new data and the user is redirected to the resources page with a success message

#### **RemoveBranchView:**

- Functionality:
  - Allows you to delete an existing affiliate
  - Processes POST requests for deleting an affiliate by removing the affiliate from the database
  - After successful deletion, the user is redirected to the resources page with a notification that the branch has been successfully deleted

**EditResourceView:**

- Functionality:
  - Allows editing of an existing resource
  - Displays a form for editing a resource's data, such as name, type, availability, etc.
  - Processes the form and updates the resource with the new data
  - Allows selected resource services to be chosen and added or deleted
  - After successful editing, the user is returned to the resources page with a success message

**RemoveResourceView:**

- Functionality:
  - Allows you to delete an existing resource
  - Handles POST requests for deletion of a resource by removing the resource from the database
  - After successful removal, the user is returned to the resources page with a notification that the resource was successfully removed

**ResourceNameView:**

- Functionality:
  - Allows you to add a new source name.
  - Processes POST requests for adding a new source name to the database.
  - Checks if the specified name already exists and adds it if it is unique.
  - Displays a success message if the name was successfully added, and an error message if the name already exists or is not specified.
  - After adding, the user is returned to the resources page.



**UpdateResourceNameView:**

- Functionality:
  - Allows you to update an existing source name.
  - Processes POST requests to update an existing source name in the database.
  - Retrieves the new name from the request and updates the source name.
  - Displays a success message after successfully updating the name.
  - After updating, the user is returned to the resources page.

**RemoveResourceNameView:**

- Functionality:
  - Allows you to delete an existing source name.
  - Processes POST requests to remove an existing source name from the database.
  - Removes the specified source name from the database.
  - Displays a success message after successfully deleting the name.
  - After deletion, the user is returned to the resource page

### 5.3.9 Promotions & Offers Developed by

Thomas Verbruggen and myself

#### **PromotionsView:**

- Functionality:
  - Show promotions on a page
  - Retrieves all promotional codes from the database
  - Sets the ability to search for promotional codes based on the search query
  - Displays list of promotional codes using pagination
  - Offers a form for adding new promotional codes

#### **CreatePromotionView:**

- Functionality:
  - Allows users to add a new promotional code
  - Processes the POST request to create a new promotional code
  - Displays a success message after successfully adding the promotion code

#### **EditPromotionView:**

- Functionality:
  - Allows users to edit an existing promotional code
  - Retrieves the existing promotional code to display in the edit form
  - Processes the POST request to update an existing promotional code
  - Displays a success message after successfully updating the promotion code

### **DeletePromotionView:**

- Functionality:
  - Allows you to remove a promotional code
  - Processes the POST request for removing a promotional code from the database
  - Displays success message after successful removal of promotion code

### **5.3.10 Company Settings**

Developed by Thomas Verbruggen and myself

#### **CompanyView:**

- Functionality:
  - Displays the company information page to the user
  - Retrieves company information from the logged-in user account
  - Displays company information in the context of the page

#### **EditCompanyView:**

- Functionality:
  - Allows users to edit company information
  - Load company information editing form
  - Retrieves current company information to display in the form
  - Processes the POST request to update company information
  - Checks and formats the phone number before storing it
  - Displays a success message after successfully updating company information
  - Redirects users to company information page after successful update

### 5.3.11 Booking (end user)

Developed by the whole group, Thomas Verbruggen, Joppe Kerkhofs and myself

#### **addServiceToAppointment:**

- Functionality:
  - Verifies that the specified appointment and service exist
  - Checks if there is already an AppointmentService with the same service
  - Checks if the new service fits with the already linked services of the appointment
  - Creates a new AppointmentService if the above checks succeed
  - Returns the created AppointmentService

#### **endAppointment:**

- Functionality:
  - Marks an appointment as ended
  - Returns the updated appointment

#### **checkWorkloadInTimeFrame:**

- Functionality:
  - Calculates an employee's workload in a specified time frame
  - Collects all appointments and associated employee services
  - Workload calculation based on duration and workload of services
  - Returns the total workload

**checkIfResourceAvailable:**

- Functionality:
  - Checks whether a resource is available within a specified time frame
  - Collects appointments within that time frame
  - Checks whether the resource is used by an AppointmentService within that time frame
  - Returns True if the resource is available, False otherwise

**checkIfResourceAvailable:**

- Functionality:
  - Checks if the new service fits with the already linked services of an appointment
  - Compares categories and subcategories of services
  - Returns True if the new service fits, otherwise False

## 6 CONCLUSION

During my internship at LHS, I gained valuable insights about creating websites and web applications. I also developed a deeper understanding of the entire process, starting with analysis and ending with a fully implemented project.

I also learned what it is like to work in a company and how numerous things can go wrong here, just think of miscommunication or misinterpretation of topics and issues. This is something that was very different from school projects and something I really had to learn to deal with.

One of the most important things for the project was also working independently, since LHS is a relatively small company, this was something that was made clear to us from the beginning. In the end, I think this went well and we can be proud of the project we put down.

All in all, my internship at LHS not only provided me with technical skills and knowledge but also refined my previous skills. Think about professional communication and deadline management, these are things that are taught in school but really refined for me during the internship period. These experiences will be of value to my future career.

## 7 GENERATIVE AI POLICY

During the internship, generative AI tools such as ChatGPT, Gemini and Phind were used.

These helped during stumbling blocks that came up during project development, think of error solving, debugging, ..... It was also useful to use these for the analytical phase, these tools helped with dilemmas we encountered while designing the database or brainstorming wireframes, for example.

Using these tools allowed us to work more efficiently and make bigger leaps in the development of the project. It also helped us develop new skills and increase our knowledge by providing direct support and guidance in solving problems.

For creating my realization document, I also used generative AI, for example, for terminology or for correcting spelling and grammatical errors.

In short, generative AI was a great tool that helped me a lot but also taught me a lot during this internship period.