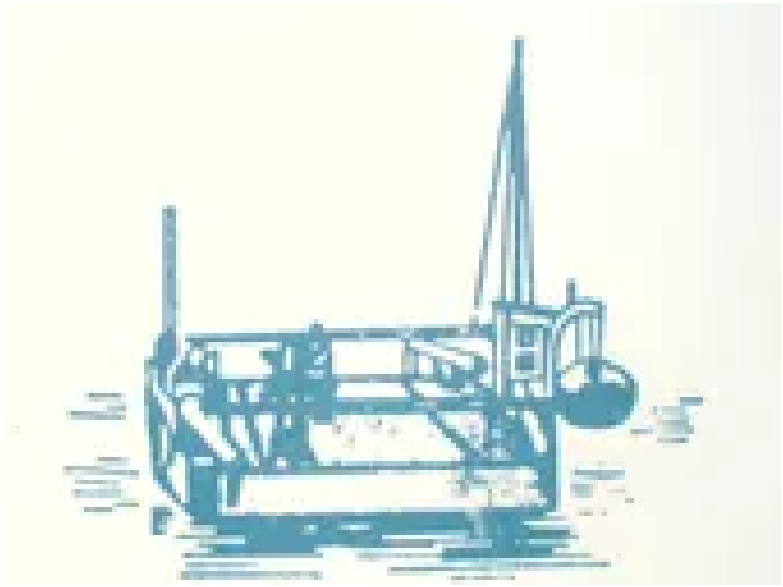
Ontwerpdocument

Ingensche Veer



In opdracht van
University of Applied Sciences Utrecht

Datum: 1 juli 2024
Versie: 2.0
Auteur: Vincent van Setten
Studentnummer: 1734729
Klas: V1C

Inhoudsopgave

1 Revisie Historie

Versie	Datum	Omschrijving
1.0	04-06-2024	Eerste Versie
2.0	30-06-2024	Tweede Versie
2.1	01-07-2024	Laatste Versie

Tabel 1: Versiegeschiedenis

2 Inleiding

Tussen Ingen en Elst (Utrecht) ligt een veerpont. Deze gaat eigenlijk constant op en neer, zonder een vast tijdschema. De veerpont is vrij groot en is voor zowel voetgangers, als auto's en fietsen. Er passen ongeveer 20 auto's op en een groot aantal fietsers en voetgangers. De veerpont is relatief druk. Er worden dagelijks honderden mensen overgezet en is erg belangrijk voor de inwoners rondom Ingen en Elst. Het lastige met de drukte op de veerpont is dat het vaak in vlagen komt. De ene overtocht heeft slechts twee auto's en bij de volgende staan er zoveel dat niet alle auto's tegelijk over kunnen. Dit komt vooral omdat mensen niet weten wanneer de veerpont aankomt, waardoor veel mensen hem vaak net missen en vervolgens lang moeten wachten.

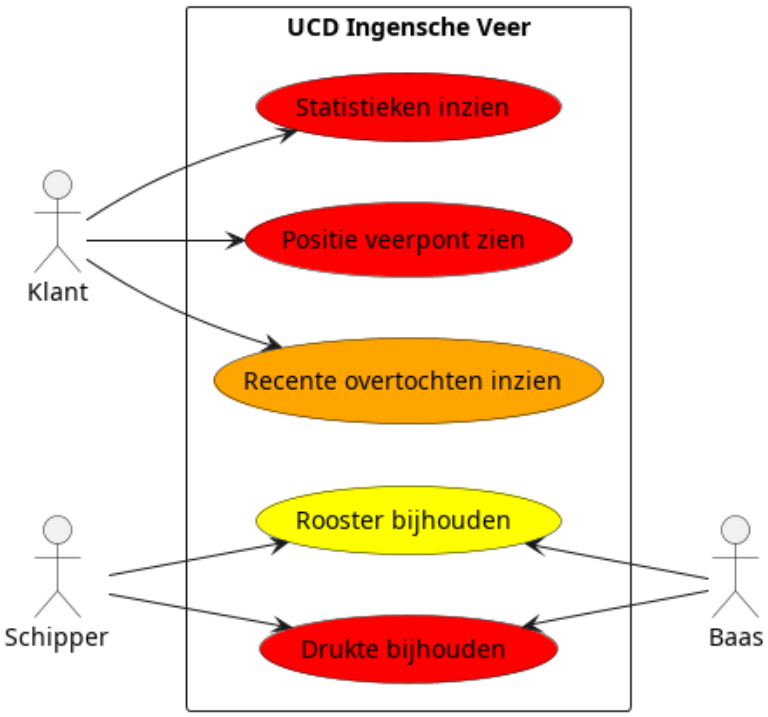
Dat zorgt voor veel frustratie bij de klanten en veroorzaakt ook stress bij de schipper. De kern van het probleem is dat klanten de aankomsttijd van de veerpont niet kunnen inschatten. Een mogelijke oplossing voor dit probleem is de ontwikkeling van een webapplicatie die de huidige locatie van de veerpont weergeeft. Met deze applicatie kunnen klanten inschatten wanneer de veerpont aankomt, wat helpt bij het spreiden van de drukte en het verminderen van de wachttijden. Dit systeem zal gebruik maken van AIS-data om real-time informatie over de veerpont te bieden.

De webapplicatie helpt ook met een ander probleem. Als het erg druk is, kan de baas van de veerpont extra hulp inschakelen. Momenteel gaat dat door puur te kijken op een bepaald moment hoe druk het is. Hiermee komt de hulp vaak net te laat, waardoor er een grote ophoping aan drukte ontstaat. Het is de bedoeling dat met de webapplicatie de schipper per overtocht kan aangeven hoe druk het is. Hiermee kan de baas zien wanneer er hulp nodig is, maar kan hiermee ook inschatten hoe druk het gaat worden op basis van historische data.

In dit ontwerpdocument wordt beschreven welke dingen er worden opgeleverd aan het eind van het IPASS project, hoe het project wordt aangepakt en er uiteindelijk uit gaat zien. Ook worden de risico's en de planning van het project beschreven.

3 Overzicht van het systeem

4 Use Cases



Figuur 1: Use Case Diagram - Ingensche Veer

In het use case diagram in figuur 1 zie je de verschillende use cases. Deze zijn gekleurd naar het MoSCoW model. Rode use cases zijn Must haves, oranje use cases zijn Should haves, gele use cases zijn Could haves en groene use cases zijn Won't haves. Het use case diagram laat alleen de daadwerkelijke geimplementeerde use cases zien. In het use case diagram zijn de aparte use cases van "Rooster bijhouden/inzien" en "Drukte bijhouden/inzien" samengevoegd in 1 voor het overzicht. Het zijn wel individuele use cases en zo zijn ze ook uitgewerkt in de use case beschrijvingen.

Terugblikkend op het Plan van Aanpak is er vrij veel geimplementeerd. Van de Must Haves is alles geimplementeerd. van de Should Haves is ook alles geimplementeerd. Van de Could Haves zijn de roosters geimplementeerd, maar de geschatte drukte, snelheid van invididuele schippers en integratie met weersomstandigheden niet. Van de Won't Haves is niks geimplementeerd. Alle statistieken van de Must haves, Could haves and Should haves zijn geimplementeerd onder de enkele use case 'statistieken inzien'.

4.1 Actoren

Actoren	**Omschrijving**
Baas	De baas is de eigenaar van de veerpont.
Klant	De klant is de persoon die gebruik maakt van de veerpont.
Schipper	De schipper is de persoon die de veerpont vaart.

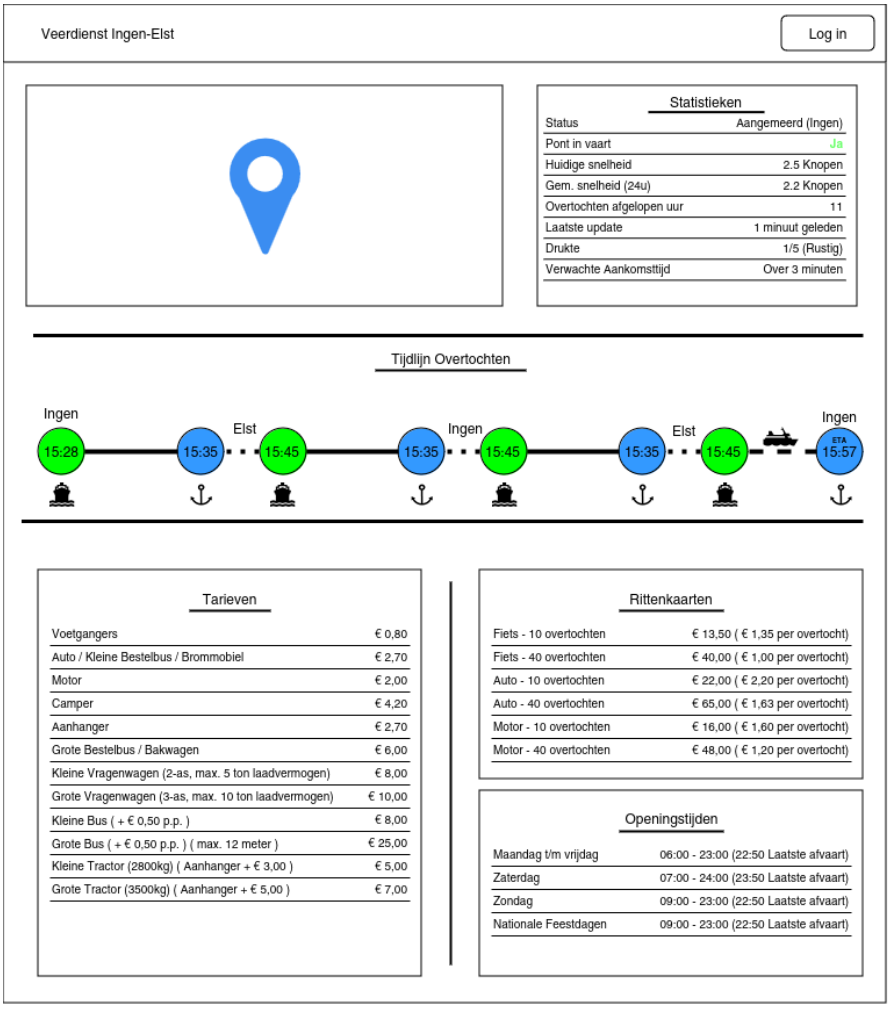
Tabel 2: Actoren

4.2 Use case beschrijvingen en wireframes

4.2.1 UC01 - Bekijk locatie veerpont

ID: UC01	**Use Case Naam:** Bekijk locatie veerpont
Actoren:	Klant
Samenvatting:	De klant kan de actuele positie van de veerpont inzien op een kaart
Preconditities:	De veerpont stuurt AIS signalen en deze worden via de API ontvangen.
Scenario:	1. De klant opent de webapplicatie 2. Systeem haalt de actuele locatie van de veerpont op. 3. Systeem toont de locatie van de veerpont op een kaart. 4. De klant kan de locatie van de veerpont inzien.
Postcondities:	nvt

Tabel 3: Use Case Beschrijving UC01 - Bekijk locatie veerpont



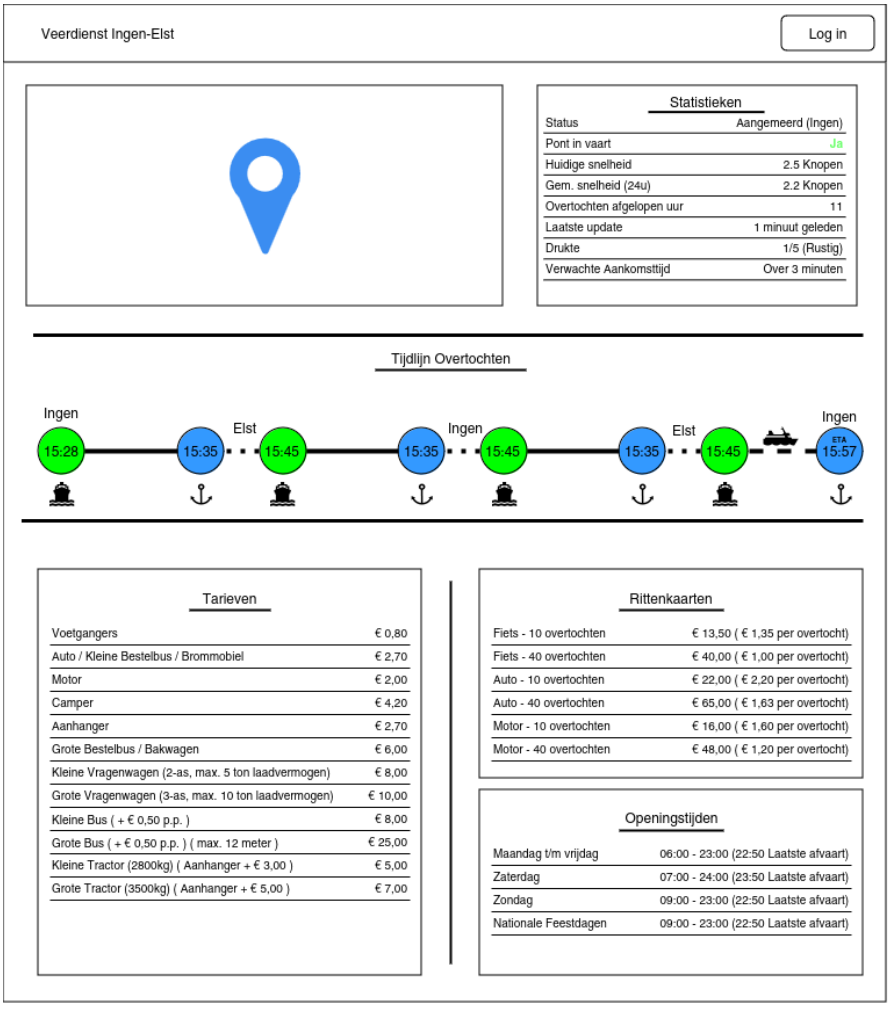
Figuur 2: Wireframe UC01 - Locatie veerpont bekijken

In figuur 2 is linksboven in te zien waar op de voorpagina de kaart zou moeten komen.

4.2.2 UC02 - Statistieken inzien

ID: UC02	**Use Case Naam:** Statistieken inzien
Actoren:	Klant
Samenvatting:	De klant kan statistieken over de veerpont inzien
Preconditie:	De veerpont stuurt AIS signalen en deze worden via de API ontvangen.
Scenario:	1. De klant opent de webapplicatie 2. Systeem haalt de ais data binnen. 3. Systeem berekent de statistieken over de data. 4. Systeem laat de statistieken zien in een tabel. 5. De klant kan de statistieken inzien.
Postcondities:	nvt

Tabel 4: Use Case Beschrijving UC02 - Statistieken inzien



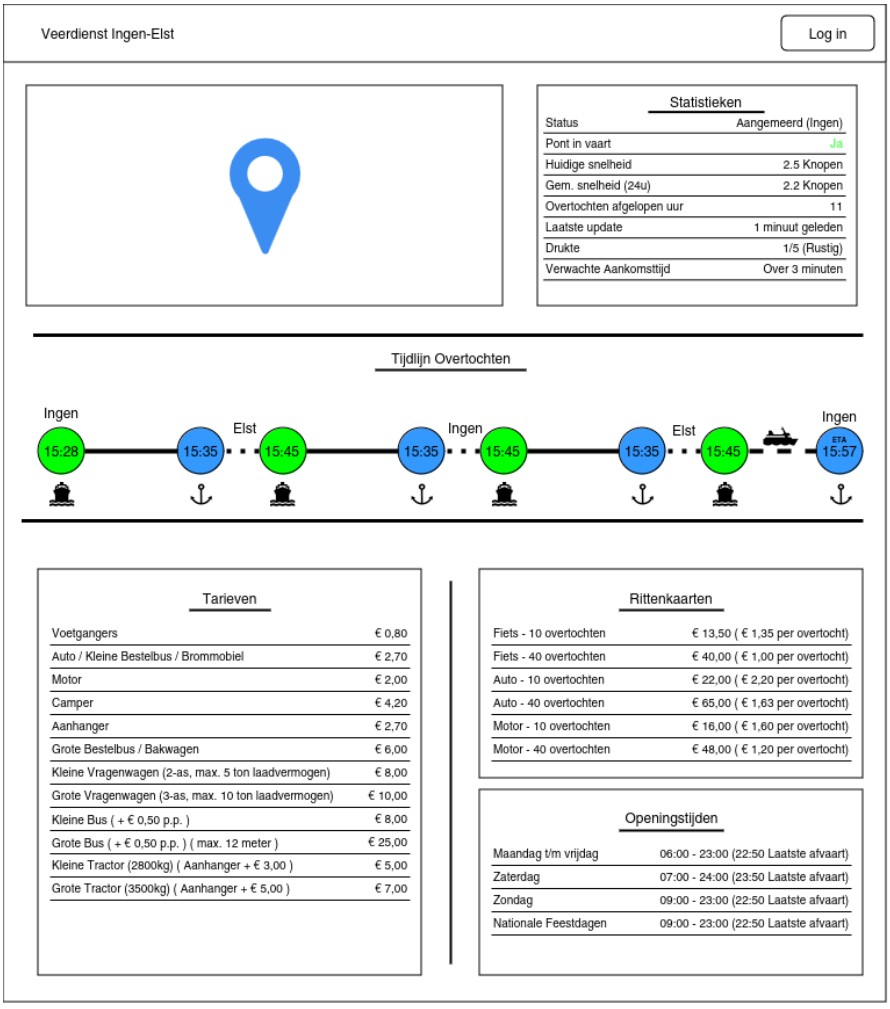
Figuur 3: Wireframe UC02 - Statistieken inzien

In figuur 3 is rechtboven te zien waar op de voorpagina de statistieken zullen staan.

4.2.3 UC03 - Recente overtochten inzien

ID: UC03	**Use Case Naam:** Recente overtochten inzien
Actoren:	Klant
Samenvatting:	De klant kan een lijst van recente overtochten zien
Preconditities:	De veerpont stuurt AIS signalen en deze worden via de API ontvangen.
Scenario:	1. De klant opent de webapplicatie. 2. Systeem haalt de ais data binnen. 3. Systeem berekent aan de hand van de ais signalen de overtochten. 4. Systeem toont een tijdlijn met de overtochten. 5. De klant kan tijdlijn van overtochten inzienn.
Postcondities:	nvt

Tabel 5: Use Case Beschrijving UC03 - Recente overtochten inzien



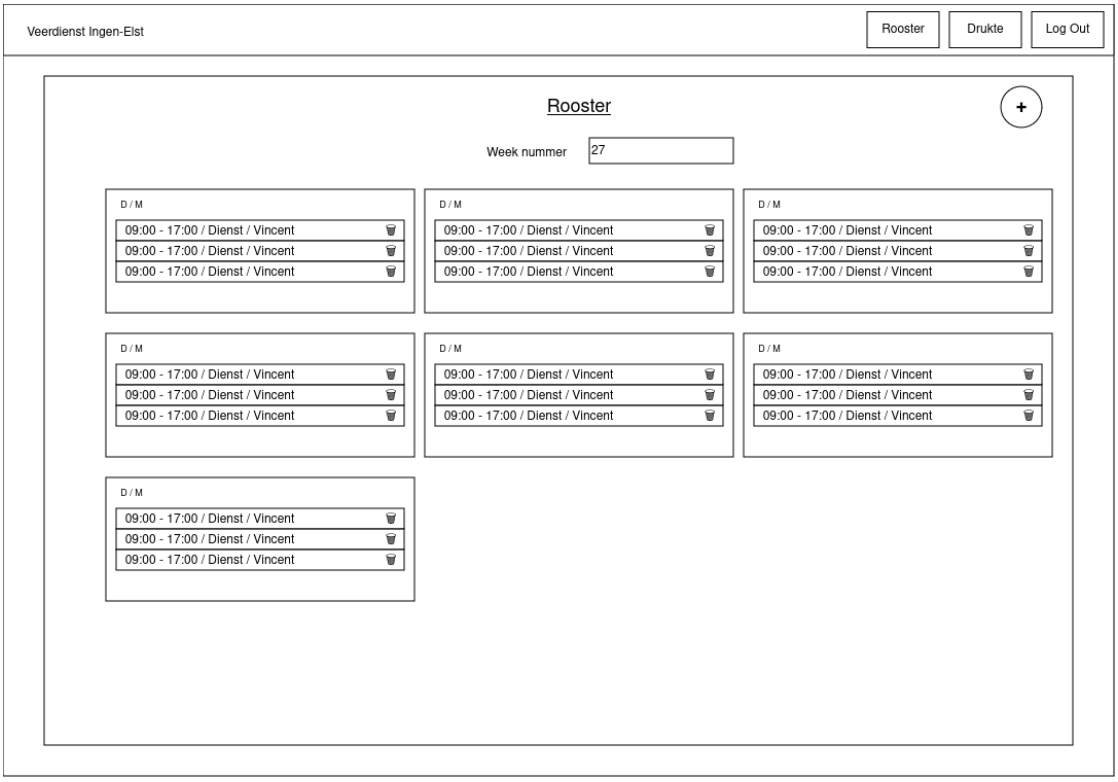
Figuur 4: Wireframe UC03 - Recente overtochten inzien

In figuur 4 is in het midden te zien waar de overtochten tijdlijn komt te staan.

4.2.4 UC04 - Rooster bijhouden

ID: UC04	**Use Case Naam:** Rooster bijhouden
Actoren:	Baas
Samenvatting:	De baas kan een rooster aanmaken en bewerken voor de schippers
Precondities:	1. De server en website staan aan en zijn verbonden met de database, 2. De baas is ingelogd.
Scenario:	1. De baas opent de website. 2. De baas klikt op de knop 'Rooster'. 3. De baas selecteert de week waarvoor hij het rooster wil aanpassen. 4. Systeem haalt de rooster items op voor die week. 5. Systeem toont de rooster items. 6. Als de baas een nieuw rooster wil aanmaken. 6.1 De baas klikt op de plus knop. 6.2 Systeem toont de popup 'rooster item toevoegen' 6.3 De baas vult in het eerste veld de gebruikersnaam van de gewenste werknemer in. 6.4 De baas vult de begin datum en tijd van de dienst in. 6.5 De baas vult de eind datum en tijd van de dienst in. 6.6 De baas vult in of het een hulpdienst of reguliere dienst betreft. 6.7 De baas klikt op de knop 'voeg toe' 6.8 Systeem de taak in de database. 6.9 Systeem herlaadt de pagina. 7. Als de baas een rooster raak wil verwijderen. 7.1 Baas klikt op het 'verwijder' knopje naast een taak. 7.2 Systeem verwijdert de taak uit de database. 7.3 Systeem herlaadt de pagina.
Postcondities:	De nieuwe staat van het rooster is opgeslagen in de database.

Tabel 6: Use Case Beschrijving UC04 - Rooster bijhouden



Figuur 5: Wireframe UC04 - Rooster bijhouden

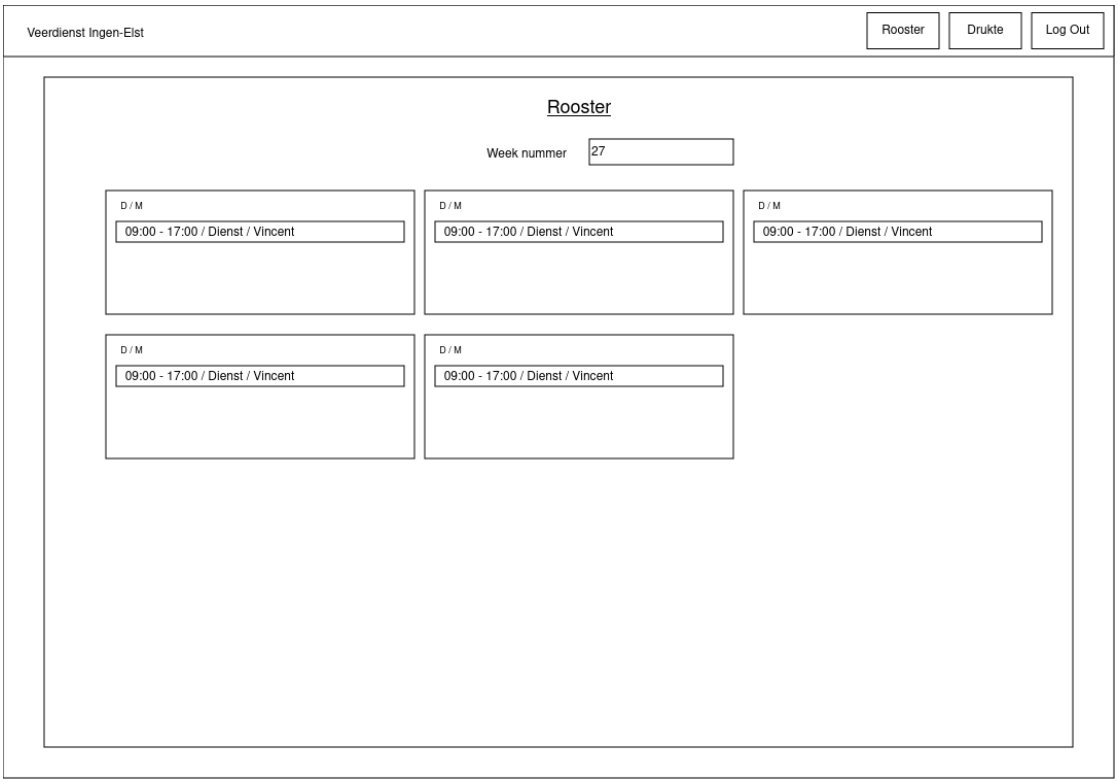
Figuur 6: Wireframe UC04 - Rooster bijhouden (Nieuw Rooster Item)

Het scherm in figuur 6 wordt getoont wanneer de baas klikt op de plus knop rechtbovens in figuur 5.

4.2.5 UC05 - Rooster inzien

ID: UC05	**Use Case Naam:** Rooster inzien
Actoren:	Schipper
Samenvatting:	De schipper kan zijn persoonlijke rooster inzien
Precondities:	1. De server en website staan aan en zijn verbonden met de database, 2. De schipper is ingelogd.
Scenario:	1. De schipper opent de website. 2. De schipper klikt op de knop 'Rooster'. 3. De schipper selecteert de week waarvoor hij het rooster wil aanpassen. 4. Systeem haalt de rooster items op voor die week. 5. Systeem toont de rooster items.
Postcondities:	nvt

Tabel 7: Use Case Beschrijving UC05 - Rooster inzien

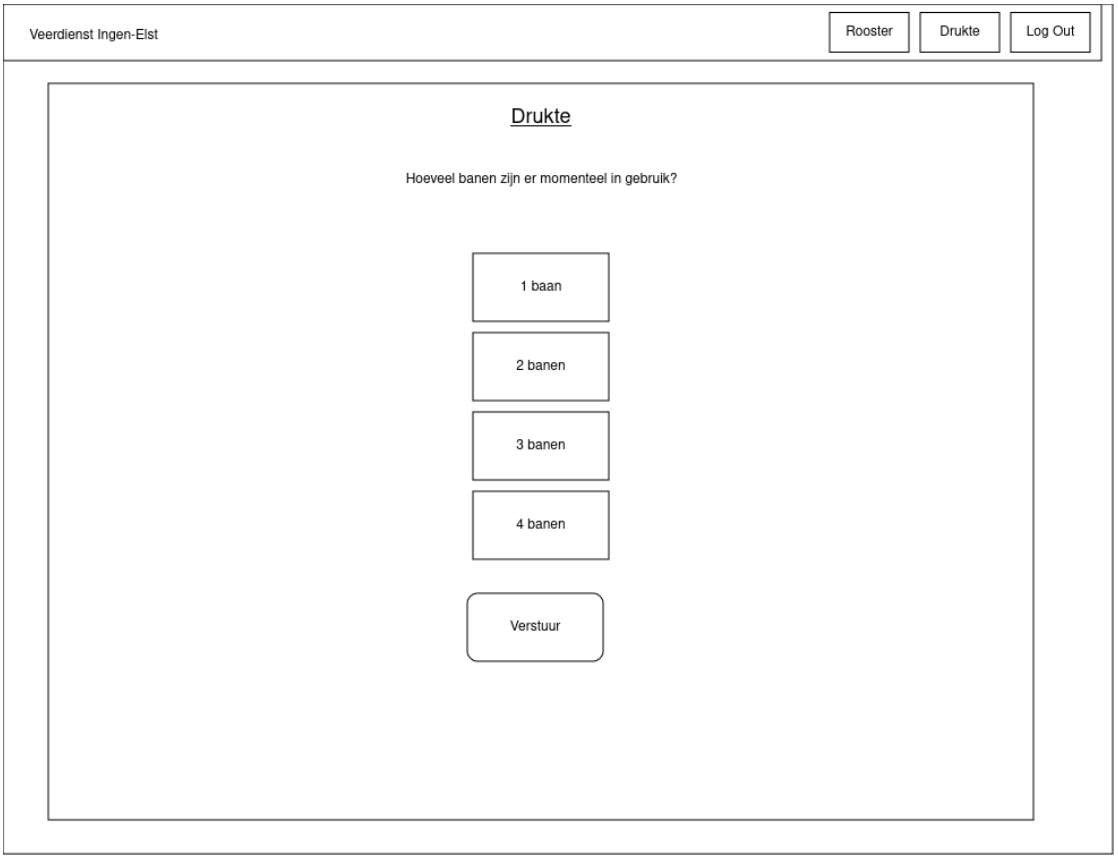


Figuur 7: Wireframe UC05 - Rooster inzien

4.2.6 UC06 - Drukte bijhouden

ID: UC06	**Use Case Naam:** Drukte bijhouden
Actoren:	Schipper
Samenvatting:	De schipper kan de drukte op de pont bijhouden.
Precondities:	1. De server en website staan aan en zijn verbonden met de database, 2. De schipper is ingelogd.
Scenario:	1. De schipper opent de website. 2. De schipper klikt op de knop 'Drukte'. 3. De schipper klikt selecteert hoeveel banen er in gebruik zijn. 4. De schipper klikt op de knop 'Verstuur' 5. Het systeem stuurt de drukte naar de database
Postcondities:	De drukte indicatie is opgeslagen in de database.

Tabel 8: Use Case Beschrijving UC06 - Drukte bijhouden

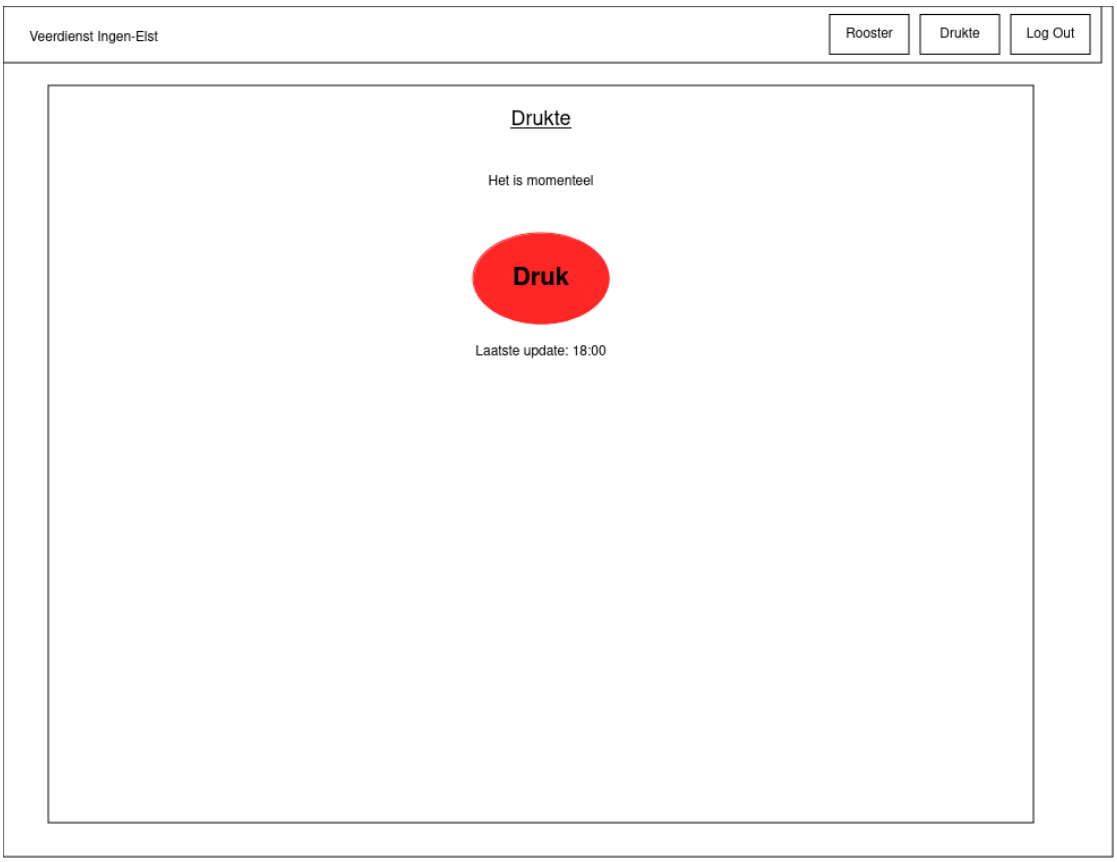


Figuur 8: Wireframe UC06 - Drukte bijhouden

4.2.7 UC07 - Drukte inzien

ID: UC07	**Use Case Naam:** Drukte inzien
Actoren:	Baas
Samenvatting:	De baas kan de drukte op de pont inzien.
Precondities:	1. De server en website staan aan en zijn verbonden met de database, 2. De baas is ingelogd.
Scenario:	1. De baas opent de website. 2. De baas klikt op de knop 'Drukte'. 3. Het systeem haalt de drukte op uit de database. 4. De baas ziet de drukte op het scherm.
Postcondities:	nvt

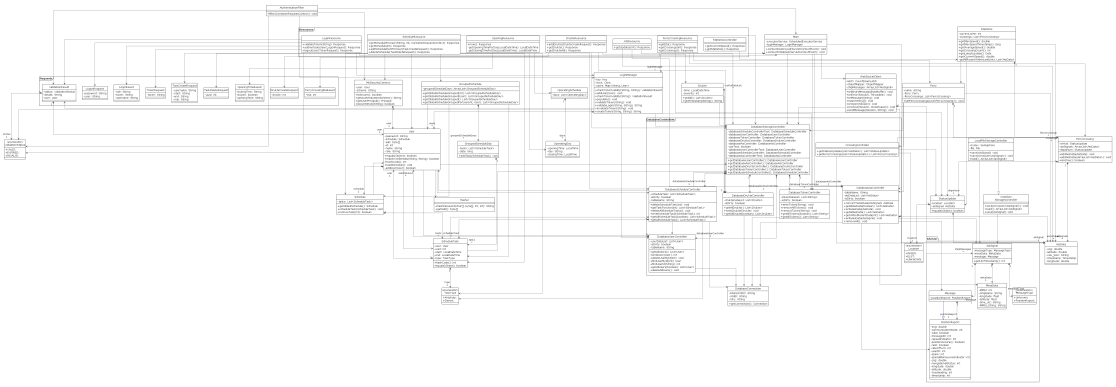
Tabel 9: Use Case Beschrijving UC07 - Drukte inzien



Figuur 9: Wireframe UC07 - Drukte inzien

5 Modellen

5.1 Domeinmodel



Figuur 10: Functioneel Domeinmodel - Ingensche Veer

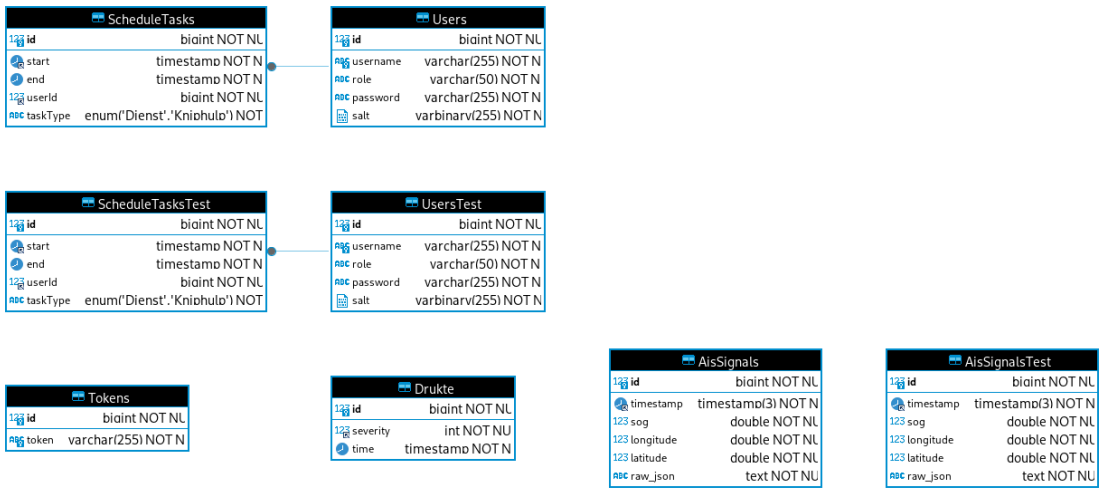
In het domeinmodel in figuur 10 zie je elke klassen in mijn implementatie. Bovenin zijn de verschillende resources te zien. De resource klassen regelen de API calls. Aan deze klassen zijn verschillende andere klassen gekoppeld die ze nodig hebben. Dit zijn veelal data klassen. Zo gebruiken de resources veel 'request' en 'result' klassen, zoals de TaskCreateRequest. Dit is een simpele klasse die helpt met het serialiseren en deserialiseren van de objecten naar de frontend. In het midden van het model zijn een aantal Database controller klassen. De klasse 'DatabaseStorageController' is de storage controller die weer verschillende DatabaseController klassen heeft. Deze klassen maken verbinding met hun eigen database tabel. Zo is DatabaseAisController verantwoordelijk voor de tabel met de AIS signalen in de database. Rechtsonderin het diagram is de AisSignal klassen te zien, samen met de subklassen. Dit zijn simpelweg json implemtatie klassen, om de inkomende json data te kunnen parsen. De testklassen zijn buiten beschouwing gelaten in dit diagram, omdat deze niet relevant zijn voor het domeinmodel en het diagram onnodig nog groter maken. *In het diagram zijn de meeste getter en setter methodes expres weggelaten om het overzicht te behouden. Enkele non-triviale of ongebruikelijke getters en setters staan er nog wel in.*

5.2 Business Rules

Bij het domeinmodel in figuur 10 horen de volgende business rules:

1. **Attribute Rule:** Een veerpont heeft minimaal een schippers.
2. **Attribute Rule:** Een schipper heeft altijd maar één baas.
3. **Entity Rule:** Elk AisSignaal heeft een snelheid, positie en tijd.
4. **Entity Rule:** Elke gebruiker heeft een gebruikersnaam en wachtwoord.
5. **Tuple Rule:** De datum van het verteksignaal is altijd kleiner dan de datum van het aankomstsignaal.

5.3 Datamodellen



Figuur 11: Database ERD - Ingensche Veer

In figuur 11 is het ERD van de database te zien. Sommige tabellen hebben een kopie. Deze heten ¡tabel¿Test. Dit zijn tabellen die gebruikt worden tijdens de tests, zodat deze niet de echte data aantasten. Ik zal de entiteiten hieronder kort toelichten.

1. **ScheduleTasks**: Deze tabel bevat een taak. Een taak is een dienst op de veerpont. Deze is gekoppeld aan een gebruiker.
2. **Users**: Deze tabel bevat de gebruikers en inlogdata van de gebruikers.
3. **Tokens**: Deze tabel bevat de JWT tokens. Deze tabel is eigenlijk onnodig en dit had lokaal gedaan kunnen worden. Ik kwam er pas later achter dat tokens niet hoeven te persisten, omdat deze geinvalidate worden zodra de applicatie opnieuw opstart.
4. **Drukte**: Deze tabel bevat een 'drukte' signaal. Een schipper kan deze doorgeven, wat de baas vervolgens weer kan inzien.
5. **AisSignals**: Deze tabel bevat de AIS data. Dit zijn de signalen die de veerpont uitzendt.

Ik heb geen fysiek of conceptueel datamodel gemaakt, omdat ik dit nooit heb gehad. Ik heb een oude versie van modelling gevolgd(in 2019) en dit niet gehad.

6 Technologieen

1. UML
2. Java
3. HTML
4. CSS
5. Javascript
6. Jax-RS(Rest)
7. HTTP-Protocol
8. MySQL

De volgende maven packages zijn gebruikt.

1. **javax.websocket.javax.websocket-api** versie 1.1
2. **com.fasterxml.jackson.core.jackson-databind** versie 2.16.1
3. **com.mysql.mysql-connector-j** versie 8.4.0
4. **javax.servlet.javax.servlet-api** versie 4.0.1
5. **org.junit.jupiter.junit-jupiter** versie 5.8.2
6. **org.glassfish.jersey.media.jersey-media-json-jackson** versie 2.35
7. **org.glassfish.jersey.containers.jersey-container-servlet** versie 2.35
8. **io.jsonwebtoken.jjwt** versie 0.9.1
9. **org.glassfish.jersey.inject.jersey-hk2** versie 2.35
10. **org.glassfish.javax.json** versie 1.1.4
11. **org.slf4j.slf4j-nop** versie 1.7.36

Voor de website heb ik de openlayers framework gebruikt om een kaart te tonen. Dit is versie 9.2.4 en gebruikt de 'BSD 2-Clause "Simplified" License' (OpenLayers, 2024). Dit is geimporteerd als een module in package.json, dus deze wordt automatisch geinstalleerd via npm install.

Verder gebruik ik alleen FontAwesome voor wat icons, deze is uitgegeven onder verschillende licenties (Font Awesome, 2024):

Icons : CC BY 4.0

Fonts : SIL Open Font License (OFL) 1.1

Code : MIT

De gebruikte versie is 5.15.4 en wordt automatisch geladen via de html head.

7 Overdracht

7.1 Installatie

1. Clone de repository
2. Ga naar de frontend folder
3. Run 'npm install'
4. Run 'npm run build'
5. Ga terug naar de source folder
6. Open het project in intellij idea
7. Laad de maven dependencies in de pom.xml
8. Stel de applicatie in als tomcat applicatie volgens de guide op canvas
9. Run de applicatie via intellij idea

Om te deployen naar azure, run het commando ‘mvn clean package azure-webapp:deploy‘.

7.2 Gebruik

Na de installatie, ga naar de website (localhost:8080) of de live versie: (veerdienst.vincentvansetten.com). Je kan inloggen via de inlog knop met de gegevens in hoofdstuk 7.3.1. Vervolgens is de website simpel te navigeren. Mochten dingen onduidelijk zijn, is dit te volgen via de use case beschrijvingen in 4.2

7.3 Wachtwoorden

7.3.1 Applicatie

- **Vincent**: admin
- **Stephan**: schipper
- **Vincentvl**: schipper
- **Baas**: baas

7.3.2 Database

De database wordt gehost op een persoonlijke AWS Lightsail instance. Hiervan zijn de gegevens:

Host ls-7f9ed97743e4c2462a9cba2e281c691d588fc281.c3co6m68qdal.eu-central-1.rds.amazonaws.com

Port 3306

Gebruiker dbmasteruser

Wachtwoord simplepw

Database veerdienst

8 Referenties

Referenties

Font Awesome. (2024). *Font Awesome License*. Verkregen juli 1, 2024, van https://github.com/FortAwesome/Font-Awesome/blob/afecf2af5d897b763e5e8e28d46aad2f710ccad6/LICENSE.txt

OpenLayers. (2024). *OpenLayers License*. Verkregen juli 1, 2024, van https://github.com/openlayers/openlayers/blob/2ba18fbafbc2dd2b056c27f71b4ec8599f9785cd/LICENSE.md