Project Depot­

Kwaliteitseisen document

Inhoud

[1.0 Teamleden & Rollen 1](#_Toc51056741)

[2.0 Applicatie 1](#_Toc51056742)

[2.1 Doel 1](#_Toc51056743)

[2.2 Aard 1](#_Toc51056744)

[2.3 Doelgroepen 1](#_Toc51056745)

[3.0 Agile/Scrum 2](#_Toc51056746)

[3.1 Verantwoordelijkheden 2](#_Toc51056747)

[3.2 Definition of Done 2](#_Toc51056748)

[4.0 Architectuur en Softwarepatronen 2](#_Toc51056749)

[4.1 FP 2](#_Toc51056750)

[4.1.1 Voordelen 2](#_Toc51056751)

[4.1.2 Nadelen 3](#_Toc51056752)

[4.2 MPA 3](#_Toc51056753)

[4.2.1 Voordelen 3](#_Toc51056754)

[4.2.2 Nadelen 3](#_Toc51056755)

[4.3 Keuze architectuur en softwarepatronen: 3](#_Toc51056759)

[5.0 Libraries 4](#_Toc51056760)

[5.1 React.js 4](#_Toc51056761)

[5.1.1 Voordelen 4](#_Toc51056762)

[5.1.2 Nadelen 4](#_Toc51056763)

[5.2 Chart.js 4](#_Toc51056764)

[5.2.1 Voordelen 4](#_Toc51056765)

[5.2.2 Nadelen 4](#_Toc51056766)

[5.3 Node.js met express API’s 5](#_Toc51056767)

[5.3.1 Voordelen 5](#_Toc51056768)

[5.3.2 Nadelen 5](#_Toc51056769)

[5.4 MySQL 5](#_Toc51056770)

[5.4.1 Voordelen 5](#_Toc51056771)

[5.5 Keuze frameworks en libraries 5](#_Toc51056772)

[5.5.1 Toelichting 6](#_Toc51056773)

[5.6 Hoofcomponenten 6](#_Toc51056774)

[5.7 Systeemomgeving 6](#_Toc51056775)

[5.8 Onderlinge samenwerking tussen hoofdcomponenten en systeemomgeving 6](#_Toc51056776)

[6.0 Database 7](#_Toc51056777)

[6.1 Logische datastructuur 7](#_Toc51056778)

[6.2 Gekozen implementatie en onderbouwing 7](#_Toc51056779)

[7.0 Coding standards 8](#_Toc51056780)

[7.1 Branching & commits 8](#_Toc51056781)

[7.1.1 Branches 8](#_Toc51056782)

[7.1.2 Commits 8](#_Toc51056783)

[7.2 Overige standards 9](#_Toc51056784)

[7.2.1 Controller/model conventies 9](#_Toc51056785)

[7.2.2 Component(vue) conventies 9](#_Toc51056786)

[7.2.3 SASS conventies 9](#_Toc51056787)

[7.2.4 Navigatie conventies 9](#_Toc51056788)

[7.2.5 Overige standards 9](#_Toc51056789)

[8.0 Eisen user interface 10](#_Toc51056790)

[8.1 Look & Feel 10](#_Toc51056791)

[8.2 Navigatie 10](#_Toc51056792)

[8.3 Invoer 10](#_Toc51056793)

[9.0 Organisatie 10](#_Toc51056794)

[9.1 Rolverdeling 10](#_Toc51056795)

[9.2 Bereikbaarheid 11](#_Toc51056796)

[9.3 Werkafspraken 11](#_Toc51056797)

# 1.0 Teamleden & Rollen

|  |  |
| --- | --- |
| Naam | Rol |
| Angelique Noordijk | Product Owner |
| Israa almahmoud | Developer, Scrum master |
| Brigitte heijkoop | Developer, Scrum master |
| Kevin Xiu | Developer, Scrum master |
| maks ratajczak | Developer, Scrum master |
| marouane azzouz | Developer, Scrum master |

# 2.0 Applicatie

## 2.1 Doel

**DL-01** Inzicht creëren in het gebruiken van een thuisbatterij voor de client. (Met invloeden zoals zonne-energieproductie, energieverbruik en energietarieven per uur)

**DL-02** Optimalisatie van batterijgebruik voor kostenbesparing en efficiëntie.

* 1. Simulatie van een volledige dag op uurbasis,
  2. Interactieve parameterinstellingen,
  3. Visualisatie van resultaten in grafieken,
  4. Integratie van actuele prijs- en weerdata.

## 2.2 Aard

**AA-01:** De thuisbatterij simulator is een webapplicatie gericht op het simuleren van het gebruik van een thuisbatterij in combinatie met zonne-energie, om gebruikers te helpen hun energieverbruik te optimaliseren en kosten te besparen.

**AA-02:** De applicatie richt zich op het simuleren van energieproductie, -verbruik, en -opslag per uur, en biedt gebruikers de mogelijkheid om met verschillende instellingen, zoals batterijcapaciteit, laad- en ontlaadtijden, en het aantal zonnepanelen, te experimenteren.

**AA-03:** De applicatie maakt gebruik van actuele en historische gegevens over energieprijzen en zonneschijn om simulaties realistisch en relevant te maken, waardoor gebruikers geïnformeerde beslissingen kunnen nemen over het gebruik van hun thuisbatterij.

## 2.3 Doelgroepen

De doelgroep voor de thuisbatterij simulator zijn huiseigenaren met zonnepanelen, die overwegen een thuisbatterij aan te schaffen of al in bezit hebben, en energiebewuste consumenten die hun energieverbruik willen optimaliseren om kosten te besparen. Ook geschikt voor mensen geïnteresseerd in duurzame energie en technologie voor zelfvoorziening.

# Agile/Scrum

## 3.1 Verantwoordelijkheden

**VER-01:** De **Scrum Master** is verantwoordelijk voor het op correcte wijze uitvoeren van de Scrum aanpak. Dit wordt gedaan door ervoor te zorgen dat het hele team de Scrum theorie, werkwijzen en regels naleeft. De Scrum Master communiceert met de Product Owner. Tevens heeft de Scrum Master op het moment ook de leiding over het versiebeheer.

**VER-02: Developers** nemen op eigen wijze de ontwikkeling van de applicatie in handen.

## 3.2 Definition of Done

Doorgaans is de meest gebruikte toepassing van de Definition of Done (hierna te noemen DoD) dat de Product Owner (JDR) het product beoordeeld heeft en de user story geaccepteerd heeft. De userstory is pas voltooid indien het aan de volgende criteria heeft voldaan:

**DoD-1: Unit tests zijn geslaagd (indien van toepassing).**

**Unit tests moeten worden geschreven en succesvol zijn uitgevoerd voor alle relevante delen van de code voordat de user story als voltooid kan worden beschouwd.**

**Wat houdt dit in:**

* **Unit tests:** Kleine, geautomatiseerde tests die individuele functies of methodes in isolatie controleren.
* **Test coverage:** De belangrijkste functionaliteit en randgevallen van de user story moeten door de tests worden gedekt.
* **Automatisering:** De tests worden uitgevoerd via een continuous integration (CI) systeem of lokaal, en moeten zonder fouten of mislukkingen slagen. Gebruik bijvoorbeeld programma’s zoals (JUnit voor Java of Jest voor JavaScript/TypeScript).
* **Randgevallen:** Unit tests moeten zowel normale als uitzonderlijke situaties (bijv. foutieve input) testen.

**DoD-2: Code is beoordeeld door andere programmeur(s) op functionaliteit.** (Alle acceptatiecriteria van uit de Product Backlog zijn geïmplementeerd en werken volgens de gestelde eisen.)

**DoD-3: Er wordt gekeken naar of de code is geschreven naar de regels die de groep er over gemaakt heeft. Voor verdere toelichting zie de volgende documenten op onze GitHub in de doc branch:** [**gitconvention**](https://github.com/BrigitteHx/20WekenProject/tree/doc/doc/gitconvention)**.**

**DoD-4: De code is gedocumenteerd en voldoet aan de afgesproken layout/coding standaarden. Voor verdere toelichting verwijzen wij u hier ook door naar onze GitHub (zie DoD-3).**

**DoD-5: De userstory/feature is getest door minimaal één ander persoon intern (developer).**

* + Teststappen worden beschreven
  + Test wordt uitgevoerd
  + Resultaat van test is beschreven

**DoD-6: De userstory/ feature is getest door de klant volgens de door de developer gedocumenteerde test stappen.**

**DoD-7: De code is gecommit via een pullrequest in het versiebeheersysteem met een verwijzing naar de userstory/feature.**

**Alle DoD punten moeten met succes worden afgerond voordat verdere stappen zoals code review of klantacceptatie plaatsvinden.**

# 4.0 Architectuur en Softwarepatronen

## 4.1 Functioneel programmeren (FP)

Procedureel programmeren is een programmeerparadigma waarbij code wordt georganiseerd in procedures of functies, die opeenvolgende stappen uitvoeren om een taak te voltooien. Het benadrukt lineaire uitvoering.

### 4.1.1 Voordelen

1. **Eenvoudig te begrijpen en implementeren**: Door de lineaire en gestructureerde aanpak is procedureel programmeren makkelijk te volgen en toe te passen, vooral voor beginners.
2. **Herbruikbare code**: Procedures en functies kunnen meerdere keren worden aangeroepen, wat leidt tot minder codeherhaling en makkelijker onderhoud.
3. **Duidelijke structuur**: De opdeling van programma's in functies maakt de code overzichtelijker en beter georganiseerd, wat het eenvoudiger maakt om fouten op te sporen en te verhelpen.
4. **Efficiëntie**: Procedureel geprogrammeerde code kan efficiënt worden uitgevoerd, omdat de stappen duidelijk gedefinieerd en sequentieel zijn, wat bijdraagt aan voorspelbare prestaties.

### 4.1.2 Nadelen

1. **Moeilijker schaalbaarheid**: Procedureel programmeren kan lastig worden bij het ontwikkelen van zeer grote en complexe systemen, omdat het beheren van veel functies en globale variabelen ingewikkeld en foutgevoelig kan zijn.
2. **Beperkte modulariteit**: Het hergebruik en de aanpasbaarheid van code kunnen beperkter zijn in vergelijking met object georiënteerd programmeren, waar data en gedrag beter kunnen worden ingekapseld en hergebruikt.

## 4.2 Multipage application (met react.js) (mpa)

De architectuur van een Multi-Page Application (MPA) met React.js maakt gebruik van meerdere aparte pagina's die elk door de server worden gerenderd. React.js wordt gebruikt om de interactiviteit en componentgebaseerde UI binnen deze pagina's te verzorgen. Elke pagina kan een eigen React-applicatie bevatten, die specifieke data en functionaliteit beheert. Dit maakt het mogelijk om de voordelen van React's component-gebaseerde benadering te benutten, terwijl de traditionele MPA structuur behouden blijft.

### 4.2.1 Voordelen

1. **SEO-vriendelijkheid en snelheid:** Omdat elke pagina afzonderlijk door de server wordt gerenderd, biedt dit betere ondersteuning voor zoekmachine-optimalisatie (SEO). Elke pagina heeft een eigen URL en wordt sneller geladen omdat de browser niet de volledige applicatie hoeft te laden bij elke navigatie.
2. **Modulariteit en onderhoudbaarheid:** Elke pagina kan worden ontwikkeld en onderhouden als een aparte module met zijn eigen React-componenten. Dit verhoogt de modulariteit van de applicatie, waardoor verschillende teams of ontwikkelaars kunnen werken aan specifieke delen van de applicatie zonder de rest te beïnvloeden.

### 4.2.2 Nadelen

1. **Duplicatie van code:** Bij een MPA-structuur kan er duplicatie van React-logica en componenten ontstaan tussen verschillende pagina's. Hierdoor kan het moeilijker worden om herbruikbare componenten te creëren en de codebase gestroomlijnd te houden.
2. **Navigatie en gebruikerservaring:** De gebruikerservaring kan minder vloeiend zijn vergeleken met een Single Page Application (SPA), omdat bij het wisselen tussen pagina's de volledige pagina opnieuw wordt geladen. Dit kan resulteren in langere laadtijden en een minder soepele navigatie.

## 4.3 Keuze architectuur en softwarepatronen

Voor dit project kiezen we voor functioneel programmeren (FP) omdat het helpt bij het schrijven van pure functies, wat leidt tot voorspelbare en eenvoudig te testen code. We kiezen voor een Multi-Page Application (MPA) om betere SEO, snellere laadtijden en duidelijke scheiding van functionaliteit per pagina te bereiken.

# 5.0 Frameworks en Libraries

## 5.1 React.js

React.js is een JavaScript-bibliotheek ontwikkeld door Facebook voor het bouwen van gebruikersinterfaces. Het maakt gebruik van componenten die herbruikbaar zijn en efficiënt updates doorvoeren in de UI door gebruik te maken van een virtuele DOM.

### 5.1.1 Voordelen

1. **Herbruikbare componenten:** Vergemakkelijkt de ontwikkeling en onderhoud van UI-elementen door herbruikbare componenten.
2. **Virtuele DOM:** Verhoogt de prestaties door efficiënte updates en rendering van de UI.
3. **Sterke community en ecosysteem:** Groot aantal beschikbare tools, libraries en een actieve community voor ondersteuning

### 5.1.2 Nadelen

1. **Leercurve:** Kan complex zijn voor beginners door de noodzaak om concepts zoals JSX en state management te begrijpen.
2. **Overhead:** Kan leiden tot extra overhead voor eenvoudige applicaties door het gebruik van een bibliotheek en virtuele DOM.

## 5.2 chart.js

Chart.js is een populaire JavaScript-bibliotheek voor het maken van interactieve en dynamische grafieken en diagrammen op webpagina's. Het biedt een eenvoudige API voor het visualiseren van gegevens met verschillende grafiektype zoals lijndiagrammen, staafdiagrammen en cirkeldiagrammen.

### 5.2.1 Voordelen

1. **Eenvoudig in gebruik:** Gemakkelijke integratie en configuratie met een intuïtieve API voor het maken van grafieken.
2. **Interactiviteit:** Ondersteunt interactieve elementen zoals tooltips en animaties voor een betere gebruikerservaring.

### 5.2.2 Nadelen

1. **Beperkingen in functionaliteit:** Minder geavanceerde functionaliteiten in vergelijking met andere grafiekbibliotheken zoals D3.js.
2. **Performance bij grote datavolumes:** Kan traag worden bij het visualiseren van zeer grote datasets of complexe grafieken.

## 5.3 node.js met express voor Api’s

Node.js is een JavaScript-runtime die serverside code uitvoert, en express is een minimalistische webframework voor Node.js dat het bouwen van webapplicaties en API's vergemakkelijkt door een eenvoudige API voor routen en middleware te bieden.

### 5.3.1 Voordelen

1. **Asynchrone I/O:** Node.js biedt non-blocking, asynchrone I/O operaties, wat leidt tot efficiënte verwerking van meerdere gelijktijdige aanvragen.
2. **Snelle ontwikkeling met express:** Express maakt het gemakkelijk om snel API's en webservices te ontwikkelen met zijn eenvoudige en flexibele API.
3. **Grote community en ecosysteem:** Een actieve community en een rijk ecosysteem van modules en pakketten in npm voor uitgebreide functionaliteit en ondersteuning.

### 5.3.2 Nadelen

1. **Callback hell:** Asynchrone aard kan leiden tot complexe en moeilijk te beheren callback-structuren, hoewel dit gedeeltelijk opgelost kan worden met “Promises en async/await”.
2. **Single-threaded beperkingen:** Node.js draait op een single-threaded event-loop, wat kan leiden tot prestatieproblemen bij CPU-intensieve taken.

## 5.4 mysql

MySQL is een open-source relationele databasebeheersysteem (RDBMS) dat gegevens opslaat in gestructureerde tabellen en SQL (Structured Query Language) gebruikt voor gegevensmanipulatie en -beheer.

### 5.4.1 Voordelen

1. **Betrouwbaarheid en stabiliteit:** MySQL is bekend om zijn stabiliteit en betrouwbare prestaties bij het beheren van grote hoeveelheden gegevens.
2. **Flexibele query-opties:** Ondersteunt complexe queries en transacties, wat zorgt voor krachtige gegevensmanipulatie en -analyse.
3. **Grote community en ondersteuning:** Een uitgebreide community en veel beschikbare documentatie, wat zorgt voor goede ondersteuning en veel tools voor integratie en beheer.

### 5.4.2 Nadelen

1. **Beperkingen bij schaling:** Kan moeilijker schalen bij extreem grote datasets of hoge verkeersvolumes zonder extra configuratie en optimalisatie.
2. **Complexiteit bij geavanceerde functionaliteiten:** Sommige geavanceerde functies en configuraties kunnen complex en moeilijk te beheren zijn voor beginners.

## 5.5 Keuze frameworks en libraries

We kiezen voor React.js vanwege de component gebaseerde structuur en interactiviteit, Chart.js voor gebruiksvriendelijke datavisualisatie, Node.js met Express voor efficiënte, asynchrone server-side API-ontwikkeling, en MySQL voor betrouwbare en flexibele relationele databasebeheer. Deze technologieën combineren schaalbaarheid, performance, en een rijk ecosysteem voor een robuuste applicatie.

## 5.6 Hoofcomponenten

De hoofdcomponenten zijn:

**HC-01:** - Client

**HC-02:** - Server

## 5.7 Systeemomgeving

**SO-01:** - Node.js

**SO-02:** - Express.js API

**SO-03:** - React.js

**SO-04:** - MySQL database

**SO-05:** - Chart.js (voor datavisualisatie)

**SO-06:** - JSON Web Tokens (JWT) voor authenticatie (indien van toepassing)

## 5.8 Onderlinge samenwerking tussen hoofdcomponenten en systeemomgeving

**Client (React.js):**

* Verantwoordelijk voor de gebruikersinterface en stuurt API-verzoeken naar de server via HTTP. Gebruikt Chart.js om data visueel weer te geven en beheert authenticatie met JWT.

**Server (Node.js/Express.js):**

* Ontvangt API-verzoeken van de client, verwerkt deze, en communiceert met de MySQL-database\*om data op te halen of op te slaan. Beheert ook gebruikersauthenticatie via JWT.

**MySQL Database:**

* Slaat gebruikersgegevens en zonnepaneelstatistieken op en levert deze aan de server bij verzoeken.

**Client en server communiceren via API’s, waarbij de server data uit de database ophaalt en naar de client stuurt voor weergave.**

# 6.0 Database

## 6.1 Logische datastructuur (Voorbeeld diagram)

**Afbeelding met tekst, schermopname, diagram, Lettertype

Automatisch gegenereerde beschrijving**

**Voor verdere toelichting betreft alle relaties sturen wij u door naar** [**dit document**](https://github.com/BrigitteHx/20WekenProject/blob/doc/doc/ERD/Doc_ERD.odt) **op onze GitHub repo.**

## 6.2 Gekozen implementatie en onderbouwing

In dit project is gekozen voor een relationele database in plaats van een NoSQL-oplossing, omdat een relationele database beter geschikt is voor het beheren van relaties tussen verschillende entiteiten, zoals gebruikers, simulatiegegevens, en feedback. Bij NoSQL-databases kunnen relaties moeilijker beschreven worden, wat kan leiden tot dubbele gegevens en databasevervuiling. Hoewel NoSQL sneller kan zijn bij het ophalen van grote hoeveelheden ongestructureerde data, is dit project gericht op een systeem dat relatief kleinere datasets verwerkt en sterke dataconsistentie vereist. Aangezien de database zich 'schoon' moet houden en relationele koppelingen essentieel zijn, is de keuze gevallen op MySQL.

Deze database biedt een robuuste en betrouwbare oplossing voor het opslaan en beheren van gegevens, en MySQL Workbench ondersteunt een gebruiksvriendelijke interface voor databasebeheer. Bovendien zorgt de combinatie van Node.js en Express voor een soepele integratie van de back-end met MySQL, wat essentieel is voor het uitvoeren van de API-verzoeken en het beheer van gebruikersdata.

# 7.0 Coding standards

**Project Git Styling:**

Branching Strategie

1. Main Branch:

Voordat aanpassingen aan de main branch kunnen worden doorgevoerd,

moeten deze grondig getest zijn. Dit gebeurt op de test branch.

De test branch wordt gemerged met de dev branch wanneer er

veranderingen zijn die bedoeld zijn voor de master branch.

2. Development Branch:

Om aanpassingen aan de dev branch te maken, dient er eerst een aparte

feature branch te worden aangemaakt.

Deze feature branch wordt aan het einde gemerged met de dev branch,

nadat de wijzigingen zijn goedgekeurd door andere teamleden.

## 7.1 Branching & commits

### 7.1.1 Branches

Er zijn diverse branches waar op gecommit kan worden.

**BR-01:** De **main branch** waar (voorlopige) releases op komen.

**BR-02:** De **test branch** is voor het testen van de applicatie voordat deze gereleaset kan worden naar de main branch.

**BR-03:** De **dev branch** waar voltooide features samengevoegd worden en getest worden door andere developers.

**BR-**04: De **feature branches** zullen worden gebruikt voor het ontwikkelen van verschillende features.

**BR-**05: De **doc branch** is voor de studenten zelf om de nodige documentatie met elkaar te delen.

### 7.1.2 Commits

**Commit Conventies:**

We gebruiken het conventionele commit styling format voor onze commit messages. Dit

zorgt voor een consistente en duidelijke commit historie.

Format:

*<type>: <beschrijving>*

*[optionele inhoud]*

*[optioneel voetstuk]*

Type:

* feat: Een nieuwe functionaliteit
* fix: Een bug fix
* docs: Alleen documentatie veranderingen
* style: Veranderingen die geen invloed hebben op de betekenis van de code (bijv.
* white-space, formatting)
* refactor: Een code verandering die geen bug oplost, en ook geen nieuwe feature
* toevoegt
* perf: Een code verandering die de performance verbetert
* test: Toevoegen van ontbrekende tests of corrigeren van bestaande tests
* build: Veranderingen die het build systeem of externe dependencies beïnvloeden
* ci: Veranderingen aan onze CI configuratie bestanden en scripts
* chore: Andere veranderingen die geen invloed hebben op de src of test bestanden
* revert: Terugdraaien van een eerdere commit

Inhoud:

De inhoud is optioneel en bevat een meer gedetailleerde uitleg over wat er in

deze commit is veranderd.

De inhoud moet in volledige zinnen geschreven worden.

Voetstuk:

In het voetstuk kunnen referenties naar issues of belangrijke notities worden

opgenomen.

Voorbeeld:

*feat: added SmsService for creating and sending SMS messages*

*Introduced a new class called SmsService for handling the creation and submission of SMS messages using the Vonage API. This service abstracts the API interactions and simplifies SMS communication within the application.*

*Additionally, added a custom exception class (SmsSubmissionException) for handling failed SMS submissions.*

*Updated the Gradle build configuration to include the Vonage API dependency.*

## 7.2 Overige standards

### 7.2.1 Controller/model conventies

**CM-01:** - **Controllers** worden benoemd in enkelvoud met een hoofdletter en de postfix ‘Controller’.  
Bijv. UserController.

**CM-02:** - **Controllers** worden georganiseerd per functionaliteit in submappen.  
Bijv. API-gerelateerde controllers worden opgeslagen in een api-map.

**CM-03:** - **Models** worden benoemd in enkelvoud met een hoofdletter en bevatten de database-structuren.  
Bijv. User

### 7.2.2 Component(vue) conventies

**CC-01:** - Alle **React-components** worden gestructureerd in mappen per subcategorie.

**CC-02:** - Components worden benoemd in PascalCase.  
Bijv. UserProfile

**CC-03:** - Imports en registraties van components gebruiken altijd PascalCase.

### 7.2.3 css/ scss conventies

**SA-01:** - Het hoofd-**SCSS**-bestand blijft app.scss en bevindt zich direct in de scss-map.

**SA-02:** - De SCSS-mappenstructuur is als volgt ingedeeld:

1. - abstracts (bevat mixins, variables, functions)
2. - base (bevat basiselementen zoals typography)
3. - components (bevat component-specifieke stijlen)
4. - layout (bevat header, footer, grids)
5. - pages (bevat pagina-specifieke stijlen)

### 7.2.4 Navigatie conventies

**NC-01:** - URL's volgen een gestructureerde opbouw:

1. - controller/action
2. - controller/controllerparameter/action
3. - controller/action/actionparameter

**NC-02:** - Voor secties zoals admin wordt een prefix toegevoegd.

1. - admin/controller/action

### 7.2.5 Overige standards

**OS-01:** - Alle HTML-pagina's moeten responsive zijn, met layouts voor grote, middelgrote en kleine schermen.

# 8.0 Eisen user interface

## 8.1 Look & Feel

De look en feel van de applicatie zijn visueel vastgelegd in wireframes. Deze wireframes bieden een duidelijk overzicht van de gebruikersinterface, inclusief de indeling van elementen, kleurenschema's, en typografie. De interface moet gebruiksvriendelijk en aantrekkelijk zijn, met een moderne uitstraling die aansluit bij de branding van de applicatie.  
Voor alle afbeeldingen van de wireframes, volg [deze link](https://github.com/BrigitteHx/20WekenProject/tree/doc/doc/wireframes) naar de juiste locatie in onze GitHub repo.

## 8.2 Navigatie

De navigatie binnen de applicatie is gevisualiseerd in een flowchart. Deze flowchart toont de verschillende pagina’s en hun onderlinge relaties, inclusief de navigatiepaden die gebruikers kunnen volgen. Belangrijke navigatie-elementen omvatten het hoofdmenu, sub-menu’s, en links naar belangrijke secties zoals het dashboard, gebruikersaccount, en instellingen.

Afbeelding met diagram, lijn, schermopname, Plan

Automatisch gegenereerde beschrijving

De link naar de PDF hiervan kunt u [hier](https://github.com/BrigitteHx/20WekenProject/tree/doc/doc/flowchart) vinden.

## 8.3 Invoer

De invoer- en interactie-elementen van de applicatie zijn beschreven in een stijlgids. Deze stijlgids omvat richtlijnen voor de vormgeving van invoervelden, knoppen, en andere interactieve elementen. Het doel is om consistentie in het ontwerp te waarborgen, met aandacht voor gebruiksvriendelijkheid en toegankelijkheid. Elementen zoals kleuren, lettergroottes, en hover-effecten worden duidelijk gedefinieerd om een uniforme gebruikerservaring te creëren.

Ook deze kunt u in onze GitHub repo vinden; [het document](https://github.com/BrigitteHx/20WekenProject/tree/doc/doc/stijlgids).

# 9.0 Organisatie

## 9.1 Rolverdeling

|  |  |
| --- | --- |
| Naam | Rol |
| Angelique Noordijk | Product Owner |
| Israa almahmoud | Developer, Scrum master |
| Brigitte heijkoop | Developer, Scrum master |
| Kevin Xiu | Developer, Scrum master |
| maks ratajczak | Developer, Scrum master |
| marouane azzouz | Developer, Scrum master |

## 9.2 Bereikbaarheid

* Point of contact vanuit studenten team: 99073629@mydavinci.nl
* Product owner vanuit Depot: A.Noordijk@depotsoftware.com

## 9.3 Werkafspraken

**Samenwerkingscontract 20-weken project Depot  
  
1. Wekelijkse inspanningsverplichting en werktijden:**

Deelnemers dienen zich te houden aan de volgende werktijden:

* Maandag: Volledig volgens het schoolrooster.
* Dinsdag: Standaard werkdag bij het bedrijf, ongeacht schoolroosterwijzigingen tot 16:00.
  + Marouane, Max, Kevin: Aankomst tussen 11:30 en 12:00 i.v.m. AVO vakken.
  + Israa, Bri: Aankomst om 09:00.
* Woensdag:
  + Israa, Bri: Vrij.
  + Marouane, Max, Kevin: Volledig volgens het schoolrooster.
* Donderdag: Stand-up meeting en werken volgens het schoolrooster.
* Vrijdag:
  + Marouane, Max, Kevin: Vrij.
  + Israa, Bri: Volledig volgens het schoolrooster.

Deelnemers zijn verplicht hun aanwezigheid en actieve deelname op de afgesproken dagen te waarborgen. Bij structurele afwezigheid kan dit gevolgen hebben (zie sectie 10).

Om een idee te geven over de doorloop van een werk sprint;

* Dinsdag: Op bedrijf aanwezig, sprint planning meeting.
* Woensdag: Werkdag. (verschillende schoolroosters)
* Donderdag: Stand-up op school, werkdag.
* Vrijdag: Werkdag. (verschillende schoolroosters)
* Maandag: Stand-up op school, werkdag.
* Dinsdag: Op bedrijf aanwezig, werkdag.
* Woensdag: Werkdag. (verschillende schoolroosters)
* Donderdag: Stand-up op school, wrap-up afgelopen sprint.
* Vrijdag: Werkdag. (verschillende schoolroosters)
* Maandag: Stand-up op school, laatste dingen gereed wrap-up & onderlinge retrospective.
* Dinsdag: Weer naar de top van deze lijst.

**2. Pauzes**

De pauzetijden volgen in principe de standaard schooltijden. Indien een extra pauze gewenst is, dient dit vooraf gecommuniceerd te worden via de groepsapp, met vermelding van de duur van de pauze.

**3. Communicatie bij afwezigheid en ziekteverzuim**

Afwezigheid of ziekte dient tijdig te worden gemeld in de daarvoor bedoelde WhatsApp- of Discord-groep, onder vermelding van de reden. Dit dient minimaal één uur voor de start van de werkdag/ schooldag te gebeuren.

Ook dient dit bij de werkgever/ docent gemeld te worden. Onder docent valt de eigen studie loopbaan begeleider, onder werkgever valt Angelique Noordijk.

**4. Te laat komen en absentie**

Na drie keer onredelijk te laat komen of afwezig zijn, wordt de persoon verantwoordelijk gehouden en dient er melding te worden gemaakt bij de betrokken docent.

Redenen zoals vertraging in het openbaar vervoer of geplande afspraken moeten tijdig via WhatsApp worden gecommuniceerd om misverstanden te voorkomen en zullen wanneer tijdig gecommuniceerd niet onder “te laat komen” vallen.

Structureel te laat komen of onregelmatige afwezigheid kan leiden tot verdere maatregelen (zie sectie 10).

**5. Centraal communicatiekanaal**

* “Onbelangrijke” zaken zoals ziekte, notities, of informele mededelingen: via WhatsApp of Discord.
* “Belangrijke” zaken zoals afspraken, deadlines, documenten, en mededelingen over code: via de officiële school-e-mail.

**6. Documentatieopslag**

Alle documentatie en projectbestanden dienen opgeslagen te worden in een gezamenlijke repository op GitHub. Elk teamlid is verantwoordelijk voor het tijdig uploaden en bijwerken van zijn of haar bijdragen.

**7. Thuiswerken**

Thuiswerken is niet verplicht op vrije dagen, wel is het verplicht op korte school dagen; ben je bijvoorbeeld om 12:15 klaar, maak de nodige werkzaamheden dan thuis af. Indien er vanuit huis wordt gewerkt, moet de voortgang gedeeld worden via Discord of, indien de voortgang essentieel is, via de e-mail.

Daarnaast wordt van elke deelnemer verwacht dat er een log wordt bijgehouden van werkzaamheden uitgevoerd vanuit huis, om terugkijken en evaluatie mogelijk te maken.

**8. Rolverdeling en taken**

De rolverdeling en taken worden beheerd via een ticket-systeem op Trello of Jira. Taken worden verdeeld op basis van ervaring en vaardigheden. De voortgang en herverdeling van taken wordt besproken tijdens de stand-up meetings, voornamelijk op korte dagen.

Er zal een constante afwisseling zijn van Scrum Masters, dit zodat iedereen in deze rol kan staan en de verantwoordelijk hiervan tot zich kan nemen.

**9. Consequenties bij niet-naleving van regels**

Wanneer een deelnemer voor de derde keer onredelijk te laat komt of een andere regel overtreedt, volgen strengere consequenties:

* Stap 1: Een officiële waarschuwing wordt gegeven, en de mentor of begeleidende docent wordt op de hoogte gebracht.
* Stap 2: Bij een vierde overtreding volgt een gesprek met de projectgroep en de leraar, waarbij de mogelijkheid bestaat om de deelnemer uit het project te zetten.
* Stap 3: Indien na het gesprek opnieuw overtredingen plaatsvinden, kan de deelnemer formeel van het project worden verwijderd. In dit geval wordt de evaluatie door de projectgroep en de school apart gedaan, en krijgt de deelnemer geen volledige projectbeoordeling.

**10. Escalatie bij structurele problemen**

Indien er sprake is van herhaaldelijke overtreding van de afspraken, zonder duidelijke verbetering, zal een formeel gesprek plaatsvinden met de begeleidende leraar. Het is mogelijk dat de deelnemer op dat moment uit het project wordt gezet, zonder verdere waarschuwingen.

De strikte handhaving van deze regels is noodzakelijk om een professionele en effectieve samenwerking te waarborgen en zal niet ter discussie staan. De aangewezen Scrum master van de werk sprint is verantwoordelijk om er voor te zorgen dat dit contract zal worden nageleefd.