Project Depot­

Kwaliteitseisen document

Inhoud

[1.0 Teamleden & Rollen 4](#_Toc180664569)

[2.0 Applicatie 5](#_Toc180664570)

[2.1 inleiding 5](#_Toc180664571)

[2.2 Doel 5](#_Toc180664572)

[2.3 aard 5](#_Toc180664573)

[2.4 Doelgroepen 6](#_Toc180664574)

[3.0 Agile/Scrum 7](#_Toc180664575)

[3.1 Verantwoordelijkheden 7](#_Toc180664576)

[3.2 Definition of Done 7](#_Toc180664577)

[4.0 Architectuur en Softwarepatronen 9](#_Toc180664578)

[4.1 Functioneel programmeren (FP) 9](#_Toc180664579)

[4.1.1 Wat zijn de voordelen van deze manier van werken? 9](#_Toc180664580)

[4.1.2 Zijn er ook nadelen? 9](#_Toc180664581)

[4.1.3 Hoe lossen we dat op? en wat is ons plan? 9](#_Toc180664582)

[4.2 Multipage application (met react.js) (mpa) 10](#_Toc180664583)

[4.2.1 Wat zijn de voordelen hiervan? 10](#_Toc180664584)

[4.2.2 zijn er ook nadelen? 11](#_Toc180664585)

[4.2.2 Hoe lossen we dat op? en wat is ons plan? 11](#_Toc180664586)

[4.3 conclusie Keuze architectuur en softwarepatronen 11](#_Toc180664587)

[5.0 Frameworks en Libraries 12](#_Toc180664588)

[5.1 React.js 12](#_Toc180664589)

[5.1.1 Wat zijn dan de voordelen van react.js? 12](#_Toc180664590)

[5.1.2 zitten er ook nadelen aan het gebruik van react.js? 12](#_Toc180664591)

[5.1.3 HOE LOSSEN WE DAT OP EN WAT IS ONS PLAN? 13](#_Toc180664592)

[5.2 chart.js 13](#_Toc180664593)

[5.2.1 Wat zijn de Voordelen van Chart.js? 13](#_Toc180664594)

[5.2.2 zijn er ook nadelen? 13](#_Toc180664595)

[5.2.3 HOE LOSSEN WE DAT OP EN WAT IS ONS PLAN? 13](#_Toc180664596)

[5.3 node.js met express voor Api’s 14](#_Toc180664597)

[5.3.1 wat zijn de voordelen van node.js met express? 14](#_Toc180664598)

[5.3.2 zijn er nog nadelen? 14](#_Toc180664599)

[5.3.3 hoe lossen we dat op en wat is ons plan? 14](#_Toc180664600)

[5.4 mysql 15](#_Toc180664601)

[5.4.1 wat zijn de voordelen van mysql? 15](#_Toc180664602)

[5.4.2 heeft het dan nog nadelen? 15](#_Toc180664603)

[5.4.3 hoe lossen we dat op en wat is ons plan? 16](#_Toc180664604)

[5.5 Keuze frameworks en libraries conclusie en toelichting 16](#_Toc180664605)

[5.6 Hoofcomponenten 16](#_Toc180664606)

[5.7 Systeemomgeving 17](#_Toc180664607)

[5.8 Onderlinge samenwerking tussen hoofdcomponenten en systeemomgeving 17](#_Toc180664608)

[6.0 Database 18](#_Toc180664609)

[6.1 Logische datastructuur (ERD) 18](#_Toc180664610)

[6.2 Gekozen implementatie en onderbouwing 21](#_Toc180664611)

[7.0 Coding standards 22](#_Toc180664612)

[7.1 Branching & commits 22](#_Toc180664613)

[7.1.1 werken mmet verschillende versies (branches) 22](#_Toc180664614)

[7.1.2 Wijzigingen opslaan en documenteren (Commits) 23](#_Toc180664615)

[7.2 Overige standards 23](#_Toc180664616)

[7.2.1 Controller/model conventies 23](#_Toc180664617)

[7.2.2 Component(vue) conventies 24](#_Toc180664618)

[7.2.3 css conventies (Style) 24](#_Toc180664619)

[7.2.4 Navigatie conventies 24](#_Toc180664620)

[7.2.5 Overige standards 25](#_Toc180664621)

[8.0 Eisen user interface 26](#_Toc180664622)

[8.1 Look & Feel 26](#_Toc180664623)

[8.2 Navigatie 27](#_Toc180664624)

[8.3 Invoer 28](#_Toc180664625)

[9.0 Organisatie 33](#_Toc180664626)

[9.1 Rolverdeling 33](#_Toc180664627)

[9.2 Bereikbaarheid 33](#_Toc180664628)

[9.3 Werkafspraken 33](#_Toc180664629)

[10.0 Bronnenlijst en auteurs 36](#_Toc180664630)

[10.1 bronnen 36](#_Toc180664631)

[10.2 Auteurs 37](#_Toc180664632)

# 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 |
| max ratajczak | Developer, Scrum master |
| marouane azzouz | Developer, Scrum master |

# 2.0 Applicatie

## 2.1 inleiding

**INL-01** **Wat is een thuisbatterij?**

* **Definitie**: Een thuisbatterij is een grote batterij dat energie kan opslaan voor gebruik in en rond het huis. Dit gebeurt wanneer de zonnepanelen meer energie genereren dan de klant op dat moment verbruikt.
* **Werking**: Wanner de zon schijnt, genereren zonnepanelen stroom op. Als de klant die stroom niet meteen nodig hebt, gaat deze naar de batterij. De batterij kan deze energie opslaan en later gebruiken , zoals ‘s avond of de dagen dat het minder zonnig is.
* **Voordelen**: Het kan helpen om te verminderen van energiekosten en meer van eigen zonne-energie gebruiken.

**INL-02** **Waarom is deze simulator belangrijk?**

* De simulator laat zien hoe de batterij de hele dag werkt. Dit helpt om te begrijpen hoeveel energie verbruikt.
* Met de simulator kunnen gebruikers uitproberen hoe ze hub batterij het beste kunnen gebruiken om geld te besparen.
* Het helpt gebruikers om betere beslissingen te maken over hun energieverbruik.
* Met meer inzicht in energiegebruik, kunnen gebruikers beter omgaan met energie en bijdragen aan beter toekomst voor het milieu.

## 2.2 Doel

**DL-01** Klanten helpen begrijpen hoe een thuisbatterij werkt en hoe deze samenwerkt met zonnepanelen. Hierbij kijken we naar factoren zoals hoeveel stroom de zonnepanelen genereren, hoeveel energie er wordt verbruikt, en wat de energieprijzen per uur zijn.

**DL-02** Het doel is om de batterij zo efficiënt mogelijk te gebruiken, zodat de klant kosten kan besparen.

Wat gaan we doen?

* 1. We maken een simulatie die laat zien wat er op elk uur van de dag gebeurt met de batterij. Bijvoorbeeld wanneer de batterij oplaadt of wanneer het energie afgeeft.
  2. Gebruikers kunnen zelf verschillende instellingen veranderen. Ze kunnen bijvoorbeeld kiezen hoeveel stroom de batterij kan opslaan, hoeveel zonnepanelen ze hebben, en wanneer de batterij oplaadt of juist stopt met opladen.
  3. De simulatie gebruikt echte prijsinformatie en weerdata van het moment, zodat de simulatie precies klopt met wat er in het echte gebeurt. Bijvoorbeeld: als het zonnig is, laad je batterij sneller op door de zonnepanelen.
  4. De resultaten worden duidelijk in grafieken weergegeven. Elke grafiek toont een specifiek aspect, zoals de hoeveelheid energie die de batterij opslaat, de hoeveelheid energie die wordt gebruikt en de status van de batterij op verschillende momenten van de dag.

## 2.3 aard

**AA-01:** De thuisbatterij simulator is een website die laat zien hoe een thuisbatterij samenwerkt met zonnepanelen. Dit helpt gebruikers om hun energieverbruik beter te plannen en geld te besparen.

**AA-02:** De website laat zien hoeveel energie er wordt genereert, gebruikt en opgeslagen, per uur. Gebruikers kunnen verschillende instellingen aanpassen, zoals wanneer de batterij oplaadt of ontlaadt, en hoeveel zonnepanelen ze hebben.

**AA-03:** De simulatie gebruikt echte gegevens over energieprijzen en hoeveel zon er is. Dit zorgt ervoor dat de simulatie klopt met de werkelijkheid en dat gebruikers beter beslissingen kunnen nemen over het gebruik van hun batterij.

## 2.4 Doelgroepen

De simulator is bedoeld voor mensen die zonnepanelen hebben of willen kopen en nadenken over een thuisbatterij. Het is ook handig voor iedereen die bewuster met energie wil omgaan en geld wil besparen. Daarnaast is het geschikt voor mensen die geïnteresseerd zijn in duurzame technologie en zelfvoorzienend willen zijn.

# Agile/Scrum

## 3.1 Verantwoordelijkheden

**Binnen ons team werken we volgens de Scrum-methode. Dit is een gestructureerde manier van werken die ons helpt om efficiënt software te ontwikkelen en flexibel in te spelen op veranderingen. Hier is een uitleg van de belangrijkste rollen en verantwoordelijkheden binnen ons team:**

**VER-01:** De **Scrum Master** is de persoon die ervoor zorgt dat ons team werkt volgens de Scrum-methode. Dit betekent dat iedereen zich houdt aan de afspraken, werkwijzen en regels van Scrum. De Scrum Master zorgt ervoor dat het proces soepel verloopt en communiceert veel met de **Product Owner** om te zorgen dat alles op één lijn blijft.  
Daarnaast heeft de Scrum Master op dit moment ook de verantwoordelijkheid over het **versiebeheer** van de code. Dat wil zeggen dat hij of zij ervoor zorgt dat de verschillende versies van de software netjes worden bijgehouden en beheerd.

**VER-02:** De **ontwikkelaars** (developers) in ons team hebben de vrijheid om de technische kant van de applicatie op hun eigen manier vorm te geven. Dit betekent dat zij zelfstandig beslissingen nemen over hoe de functies worden gebouwd en hoe de code wordt geschreven. Natuurlijk gebeurt dit altijd binnen de afspraken die we als team hebben gemaakt en in lijn met de wensen van de klant.

## 3.2 Definition of Done

Bij softwareontwikkeling werken we met gebruikersverhalen (user stories) om duidelijk vast te leggen wat een bepaalde functie in het systeem moet doen. Maar hoe weten we of een verhaal écht klaar is? Daarvoor gebruiken we de **Definition of Done (DoD)**. Dit is een lijst van criteria waaraan we ons houden om te bepalen of een functie volledig en goed is uitgevoerd. Pas als aan alle punten is voldaan, beschouwen we de functie als af. Wanneer is dan de userstory echt af? Hieronder staat de lijst met criteria:

**DoD-1: De functie is getest:**

**Wat houdt dit in:**

* We testen de code met zogenaamde **unit tests**. Dit zijn kleine, automatische tests die controleren of de losse onderdelen van de code goed werken. Deze tests zijn belangrijk om te checken of de functie zich gedraagt zoals bedoeld, ook in uitzonderlijke situaties (zoals foutieve invoer). Voor het uitvoeren van deze tests gebruiken we testsoftware zoals **JUnit** (voor Java) en **Jest** (voor JavaScript en TypeScript). Deze tests draaien automatisch binnen ons **Continuous Integration (CI)**-systeem, wat betekent dat we meteen een melding krijgen als er iets misgaat.

**DoD-2: Een andere programmeur heeft de code gecontroleerd op de functionaliteit.**

* Nadat de eerste programmeur de code heeft geschreven, kijkt een tweede programmeur deze grondig na. Dit noemen we een **code review**. Hierbij controleren we of de code voldoet aan alle **acceptatiecriteria** die we vooraf hebben opgesteld. Zo zorgen we ervoor dat alles werkt zoals afgesproken.

**DoD-3: De code is in lijn met de teamafspraken.**

* Onze programmeurs houden zich aan bepaalde teamafspraken over hoe de code moet worden geschreven. Dit gaat bijvoorbeeld over de structuur en opmaak van de code. We hebben deze regels vastgelegd in ons **GitHub**, waar je ze altijd kunt terugvinden onder de sectie "gitconvention" in de doc branch.

**DoD-4: De code voldoet aan de layout- en opmaakregels.**

* De code moet niet alleen technisch kloppen, maar ook netjes en overzichtelijk zijn geschreven. Hiervoor hebben we regels opgesteld, zodat iedereen in het team de code gemakkelijk kan begrijpen en onderhouden. Deze regels kun je ook terugvinden in onze GitHub.

**DoD-5: De functie is getest door een andere programmeur.**

* Naast de eerste automatische tests, wordt de functie ook handmatig getest door een tweede programmeur. Hierbij doorloopt hij of zij de teststappen om te kijken of de functie echt goed werkt. Deze resultaten worden vastgelegd, zodat er altijd een overzicht is van wat er is getest.

**DoD-6: De klant heeft de functie getest.**

* Voordat we de functie definitief afronden, vragen we jou, als klant, om de functie te testen. Dit doe je aan de hand van de stappen die de programmeur heeft opgesteld. Zo kunnen we er zeker van zijn dat de functie ook in de praktijk naar wens werkt.

**DoD-7: De code is toegevoegd aan ons versiebeheersysteem.**

* Als alles is goedgekeurd, voegen we de nieuwe code toe aan ons versiebeheersysteem via een zogenaamde pull request. Dit is een verzoek om de code officieel op te nemen in de definitieve versie van het product. Bij deze stap voegen we ook een link toe naar het bijbehorende gebruikersverhaal, zodat alles netjes is gedocumenteerd.

**Door deze stappen te volgen, zorgen we ervoor dat elke functie grondig getest, gecontroleerd en goedgekeurd is. Op die manier garanderen we dat jij een betrouwbaar en goed werkend product ontvangt.**

# 4.0 Architectuur en Softwarepatronen

## 4.1 Functioneel programmeren (FP)

Afbeelding met tekst, schermopname, Lettertype, software

Automatisch gegenereerde beschrijvingBij het bouwen van software volgen we een bepaalde aanpak, die 'procedureel programmeren' heet. Dit betekent dat we de code opdelen in kleine stapjes, ofwel 'procedures'. Elke procedure voert één taak uit, zoals het verwerken van data of het tonen van een bericht. Door de code in logische stappen te verdelen, blijft alles overzichtelijk en kunnen we makkelijk zien wat er gebeurt.

### 4.1.1 Wat zijn de voordelen van deze manier van werken?

1. **Makkelijk te begrijpen:**

Omdat we de code stap voor stap opbouwen, is het voor iedereen duidelijk wat er gebeurt. Het is alsof je een kookrecept volgt: je doet één ding tegelijk, in een logische volgorde. Dit maakt het eenvoudiger voor iedereen, zelfs voor mensen die minder ervaring hebben met programmeren, om de code te begrijpen en te gebruiken.

1. **Herbruikbare code:**

Wanneer we een taak, zoals het berekenen van een bedrag of het versturen van een melding, eenmaal hebben gemaakt, kunnen we die procedure steeds opnieuw gebruiken. Dit voorkomt dat we steeds dezelfde code opnieuw moeten schrijven. Dat bespaart tijd en verkleint de kans op fouten, omdat we zeker weten dat het eerder goed heeft gewerkt.

1. **Overzichtelijke opbouw**:

Doordat we alles in kleine stappen opdelen, blijft de code netjes en overzichtelijk. Dit helpt ons snel fouten te vinden als er iets niet werkt, en het maakt het makkelijker om later veranderingen aan te brengen.

1. **Snel en duidelijk**:

De code wordt in een vaste volgorde uitgevoerd, wat betekent dat het programma precies weet welke stappen het moet nemen. Hierdoor werkt alles snel en kunnen we van tevoren goed voorspellen hoe het programma zich gedraagt. Dit zorgt voor betrouwbare en voorspelbare prestatie.

### 4.1.2 Zijn er ook nadelen?

1. **Lastig bij grote projecten:**

Als een project heel groot wordt, kan het moeilijk zijn om het overzicht te houden. Er zijn dan veel stappen en variabelen die allemaal beheerd moeten worden, wat soms voor verwarring kan zorgen.

1. **Minder flexibel:**

In vergelijking met andere manieren van programmeren, zoals objectgeoriënteerd programmeren, is het moeilijker om de code in kleine stukjes op te delen die je makkelijk opnieuw kunt gebruiken of aanpassen.

### 4.1.3 Hoe lossen we dat op? en wat is ons plan?

Hoewel deze manier van programmeren goed werkt voor kleinere projecten, komen er bij grotere en complexere systemen soms uitdagingen naar voren, zoals beschreven bij de nadelen. Hieronder staat hoe we dat oplossen!

1. **Grotere projecten beheren:**

Voor grotere en ingewikkelde projecten gebruiken we soms een andere manier van werken (objectgeoriënteerd programmeren). Dit maakt het makkelijker om alles te organiseren en te zorgen dat de code klaar is voor de toekomst. OOP en FP zijn makkelijk samen te gebruiken, het zorgt zelfs voor het beste resultaat. Tot nu toe verwachten wij dat FP het meeste van dit project in beslag zal nemen. Maar we hebben dit wel bewust gekozen mochten we moeten overstappen naar een gedeelte OOP.

1. **Meer flexibiliteit creëren:**

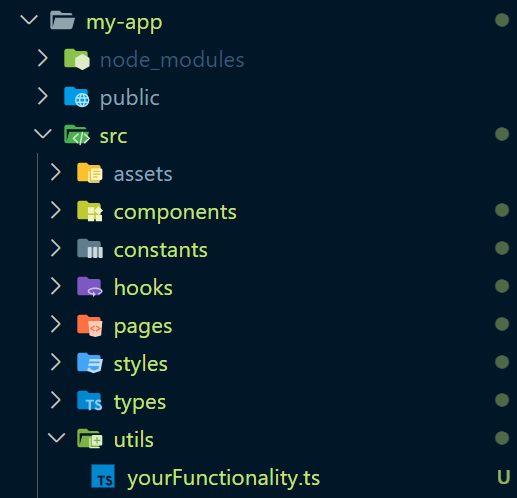
We maken de code flexibel door deze in kleinere onderdelen te splitsen. Zo kunnen we makkelijker dingen veranderen of hergebruiken zonder dat we alles opnieuw moeten schrijven.

1. **Testen om de kwaliteit te waarborgen:**

Om ervoor te zorgen dat de code goed blijft werken, zelfs als er veranderingen zijn, maken we gebruik van automatische tests. Zo kunnen we snel eventuele problemen opsporen en verhelpen. 2.0 Agile, Definition of Done, hebben we beschreven hoe dit in zijn werking gaat.

**Ons plan:**  
Door deze aanpak koppelen we direct terug naar de nadelen van functioneel programmeren en lossen we ze op door OOP en modulaire code waar nodig in te zetten. Zo behouden we de efficiëntie en duidelijkheid van FP, terwijl we schaalbaarheid en flexibiliteit toevoegen.

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

Bij het maken van een Multipage application (MPA) met React.js gebruiken we verschillende pagina's. Elke pagina wordt door de server gemaakt. React.js helpt om deze pagina's interactief te maken, zodat ze leuk en gebruiksvriendelijk zijn. Dit betekent dat elke pagina zijn eigen kleine React-app heeft, die alles beheert wat je op die pagina ziet en doet. Op deze manier kunnen we genieten van de voordelen van React terwijl we toch met aparte pagina's werken.

### 4.2.1 Wat zijn de voordelen hiervan?

1. **Beter vindbaar op internet:**

Elke pagina heeft zijn eigen adres (URL), wat goed is voor zoekmachines. Omdat de server elke pagina apart laadt, gaat dit ook sneller, omdat je niet elke keer de hele app hoeft te laden.

1. **Makkelijker te onderhouden:**

Elke pagina kan apart worden gemaakt met zijn eigen onderdelen (componenten). Dit maakt het makkelijker om aan verschillende pagina's te werken zonder dat je andere pagina's beïnvloedt.

### 4.2.2 zijn er ook nadelen?

1. **Herhaling van code:**

Soms kan het gebeuren dat we dezelfde code op verschillende pagina's moeten schrijven. Dit kan het moeilijk maken om alles netjes en georganiseerd te houden.

1. **Minder vloeiende ervaring:**

Als je van de ene naar de andere pagina gaat, moet de hele pagina opnieuw laden. Dit kan ervoor zorgen dat het langer duurt voordat je de nieuwe pagina ziet, wat niet zo fijn is voor de gebruiker.

### 4.2.2 Hoe lossen we dat op? en wat is ons plan?

1. **Minder herhaling:**

We creëren gemeenschappelijke onderdelen die we op meerdere pagina's kunnen gebruiken. Dit voorkomt dat we steeds dezelfde code schrijven en helpt om alles overzichtelijk te houden. Dit heet het gebruiken van “componenten”.

1. **Vlottere navigatie:**

We zorgen ervoor dat de overgang tussen pagina's soepeler gaat. Bijvoorbeeld door een laadscherm te gebruiken, zodat de gebruiker weet dat de pagina aan het laden is. We kunnen ook kijken of we sommige delen van de app als een Eénpagina-app (SPA) kunnen maken, zodat de navigatie sneller gaat.

1. **Luisteren naar gebruikers**:

We vragen regelmatig feedback van gebruikers. Door te horen wat ze leuk vinden en wat niet, kunnen we verbeteringen aanbrengen. Dit helpt ons om de ervaring van de gebruikers te optimaliseren.

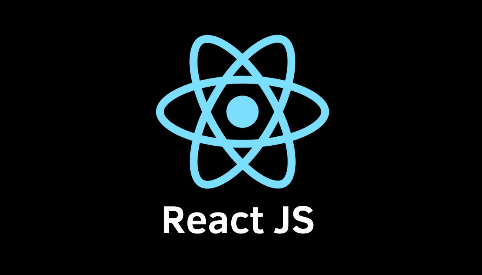
**Ons plan:**   
We willen de voordelen van een MPA behouden en tegelijkertijd de nadelen aanpakken. Door slim om te gaan met herhaling van code, de gebruikerservaring te verbeteren en goed naar feedback te luisteren, zorgen we ervoor dat onze app gemakkelijk te gebruiken en goed te onderhouden is. Hierdoor krijgen we het beste van deze methode bij elkaar!

## 4.3 conclusie Keuze architectuur en softwarepatronen

Voor dit project kiezen we voor functioneel programmeren (FP). Dit maakt het mogelijk om duidelijke en voorspelbare functies te schrijven, wat het testen van de code veel gemakkelijker maakt. Daarnaast kiezen we voor een Multipage application (MPA). Dit zorgt ervoor dat onze pagina's beter vindbaar zijn op het internet, dat ze sneller laden en dat we de verschillende functies goed van elkaar kunnen scheiden. Door deze keuzes kunnen we een app maken die gebruiksvriendelijk, goed te onderhouden is en goed zal werken naar de eisen van de klant.

# 5.0 Frameworks en Libraries

## 5.1 React.js

Het is al eerder genoemd, maar wat is dat dan nu “React.js”? React.js is een handige tool ontwikkeld door Facebook om websites en apps interactief te maken. Het helpt ons om verschillende onderdelen van een gebruikersinterface te maken die we gemakkelijk opnieuw kunnen gebruiken. Dit betekent dat we snel en efficiënt updates kunnen doorvoeren, wat zorgt voor een prettige ervaring voor de gebruiker.

### 5.1.1 Wat zijn dan de voordelen van react.js?

1. **Herbruikbare onderdelen:**

Een van de grootste voordelen van React.js is dat we *componenten* kunnen maken. Dit zijn kleine, zelfstandige stukjes code die we op verschillende plekken kunnen gebruiken. Dit maakt het ontwikkelen en onderhouden van de interface veel makkelijker, omdat we niet steeds opnieuw dezelfde dingen hoeven te maken.

1. **Snellere prestaties:**

React.js maakt gebruik van iets dat de *virtuele DOM* heet. Dit klinkt ingewikkeld, maar het betekent gewoon dat React slimmer omgaat met het bijwerken van de interface. In plaats van alles opnieuw te laden, werkt het alleen de dingen bij die echt veranderd zijn. Dit maakt de app sneller en zorgt voor een soepelere gebruikerservaring.

1. **Sterke ondersteuning:**

Er is een grote en actieve community rondom React.js. Dit betekent dat er veel tools en extra hulp beschikbaar zijn. Als je ergens tegenaan loopt, is er vaak wel iemand die een oplossing heeft of een handige bibliotheek kan aanbevelen.

### 5.1.2 zitten er ook nadelen aan het gebruik van react.js?

1. **Leren kost tijd:**

Voor beginners kan React.js soms overweldigend zijn. Er zijn nieuwe concepten zoals JSX (een manier om HTML te schrijven in JavaScript) en *state management* (hoe we de gegevens in onze app beheren) die je moet begrijpen. Dit kan even duren voordat je het onder de knie hebt.

1. **Extra lasten voor simpele apps:**

Als je een hele simpele app maakt, kan het gebruik van React.js wat extra werk met zich meebrengen. Het kan wat meer tijd en moeite kosten om alles op te zetten dan wanneer je een eenvoudige oplossing zonder bibliotheek zou gebruiken.

### 5.1.3 HOE LOSSEN WE DAT OP EN WAT IS ONS PLAN?

1. **Eenvoudiger leren:**

We kunnen beginners helpen door duidelijke en simpele tutorials te maken. Door stap voor stap uit te leggen hoe React werkt, maken we het toegankelijker voor iedereen.

1. **Slimme keuzes maken:**

Als je alleen een eenvoudige app wilt maken, kunnen we goed nadenken of React de juiste keuze is. Voor kleine projecten kunnen we ook kiezen voor simpelere oplossingen, zodat het niet te ingewikkeld wordt. Toch hebben wij veel grote ideeën voor dit project dus zullen wij wel gebruik maken van React!

1. **Samenwerken met de community:**

We willen gebruik maken van de sterke community rond React.js. Door regelmatig contact te hebben met andere ontwikkelaars, kunnen we nieuwe tips en tools ontdekken die ons helpen om onze apps beter en makkelijker te maken.

**Ons plan nog even samengevat:**

* React.js biedt veel voordelen zoals herbruikbare onderdelen en snellere prestaties. Maar we moeten ook rekening houden met de uitdagingen. Door goed te plannen en te leren van de community, zorgen we ervoor dat we het beste uit React halen, terwijl we het eenvoudig en begrijpelijk houden voor iedereen!

## 5.2 chart.js

Chart.js? Wat is dat? Chart.js is een populaire tool die we gebruiken om interactieve grafieken en diagrammen te maken op webpagina's. Het is superhandig, omdat het een eenvoudige manier biedt om gegevens visueel weer te geven met verschillende soorten grafieken, zoals lijndiagrammen, staafdiagrammen en cirkeldiagrammen.

### 5.2.1 Wat zijn de Voordelen van Chart.js?

1. **Gebruiksvriendelijk:**

Een van de grootste pluspunten van Chart.js is dat het heel makkelijk te gebruiken is. Het heeft een intuïtieve interface waarmee je snel grafieken kunt maken en aanpassen. Dit betekent dat je niet veel tijd hoeft te besteden aan het leren van ingewikkelde code, maar meteen aan de slag kunt.

1. **Interactiviteit:**

Met Chart.js kun je interactieve elementen toevoegen, zoals *tooltips* (kleine pop-upvensters met informatie) en animaties. Dit maakt het leuker en gemakkelijker voor gebruikers om met de grafieken te werken en de gegevens beter te begrijpen.

### 5.2.2 zijn er ook nadelen?

1. **Beperkingen in werking:**

Hoewel Chart.js een geweldige tool is, heeft het wel enkele beperkingen. Het biedt misschien niet alle geavanceerde functies die andere grafiekbibliotheken, zoals D3.js, hebben. Dit betekent dat als je hele complexe visualisaties wilt maken, je mogelijk andere opties moet overwegen.

1. **Prestaties bij grote gegevens:**

Chart.js kan traag worden als je heel grote datasets of complexe grafieken probeert weer te geven. Dit kan frustrerend zijn voor gebruikers die snel door de informatie willen navigeren.

### 5.2.3 HOE LOSSEN WE DAT OP EN WAT IS ONS PLAN?

1. **Slimme keuzes maken:**

We kunnen goed nadenken over de soorten grafieken die we willen maken. Voor eenvoudigere visualisaties is Chart.js perfect. Voor complexere projecten kunnen we andere bibliotheken overwegen die meer functies bieden.

1. **Gegevens optimaliseren:**

Als we met grote datasets werken, kunnen we technieken toepassen om de gegevens te optimaliseren. Dit kan helpen om de prestaties te verbeteren, zodat de grafieken sneller laden en reageren.

**Samenvatting en ons plan:**

Chart.js is een handige tool voor het maken van grafieken die eenvoudig te gebruiken is en een leuke gebruikerservaring biedt. Maar we moeten ook bewust zijn van de beperkingen, vooral bij grotere datasets. Door slimme keuzes te maken en onze gegevens goed te beheren, kunnen we het meeste uit Chart.js halen en mooie, interactieve visualisaties creëren!

## 5.3 node.js met express voor Api’s

Wat is Node.js met Express?Node.js is een krachtige tool die ons helpt om JavaScript-code op de server te draaien, in plaats van alleen in de browser. Express is een populair framework dat bovenop Node.js is gebouwd. Het maakt het gemakkelijker om webapplicaties en API's (dat zijn de koppelingen tussen verschillende software) te maken. Met Express krijg je een eenvoudige manier om routes te beheren en middleware te gebruiken, wat het proces nog eenvoudiger maakt

### 5.3.1 wat zijn de voordelen van node.js met express?

1. **Snel en efficiënt:**

Node.js kan veel aanvragen tegelijk verwerken omdat het niet wacht op één taak om te eindigen voordat het met de volgende begint. Dit maakt het snel en efficiënt.

1. **Eenvoudig en snel ontwikkelen:**

Met Express kun je heel snel API's en webdiensten maken. Het is makkelijk te gebruiken, zodat je zonder veel gedoe aan de slag kunt.

1. **Veel hulp en ondersteuning:**

Node.js en Express hebben een grote en actieve community. Dit betekent dat je veel hulp kunt krijgen en dat er veel extra tools en modules beschikbaar zijn in npm (de package manager voor Node.js). Dit maakt het eenvoudig om meer functies aan je projecten toe te voegen.

### 5.3.2 zijn er nog nadelen?

1. **Moeilijke code structuur:**

Door de asynchrone manier van werken kunnen er ingewikkelde structuren ontstaan die moeilijk te begrijpen zijn, ook wel *callback hell* genoemd. Dit kan verwarrend zijn, maar er zijn technieken zoals *Promises* en *async/await* die helpen om de code overzichtelijker te maken.

1. **Beperkingen bij zware Taken:**

Node.js werkt met een *single-threaded event-loop*. Dit betekent dat het problemen kan hebben met taken die veel rekenkracht vereisen. Dit kan de prestaties beïnvloeden als je zware berekeningen doet.

### 5.3.3 hoe lossen we dat op en wat is ons plan?

1. **Makkelijker beheer van code:**

We gebruiken technieken zoals *Promises* en *async/await* om de code eenvoudiger te maken en problemen met *callback hell* te voorkomen.

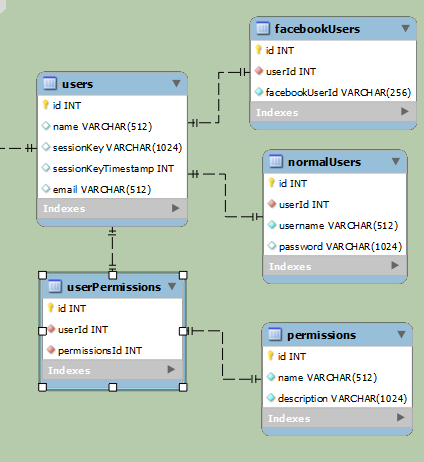
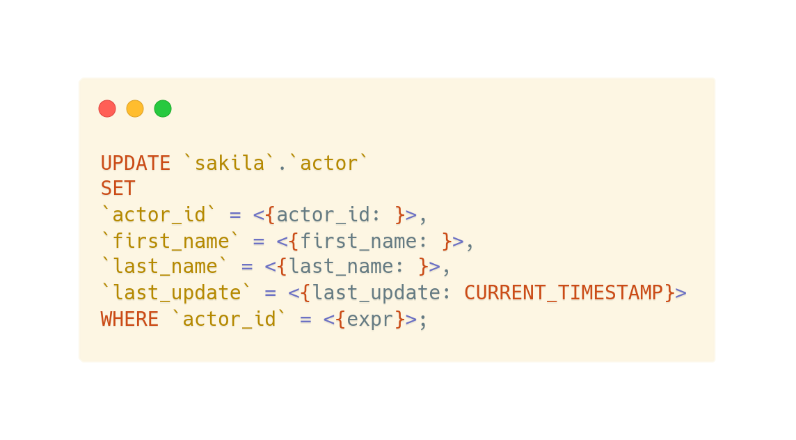
1. **Slimme verdeling van taken:**

Als we applicaties bouwen, zorgen we ervoor dat zware taken niet op de hoofdthread draaien. We kunnen gebruikmaken van *worker threads* of *microservices* om de belasting beter te verdelen en de prestaties te verbeteren.

**Samenvatting en ons plan:**

Node.js met Express is een geweldige combinatie voor het bouwen van webapplicaties en API's. Het heeft veel voordelen zoals snelheid en eenvoud, maar we moeten ook letten op de uitdagingen. Door slimme technieken te gebruiken en goed na te denken over onze taken, kunnen we het meeste uit deze tools halen en sterke applicaties maken.

## 5.4 mysql

En dan last but not least, hebben we nog iets nodig voor de databases van dit project! Daarvoor komen we bij: SQL. MySQL is een populaire, open-source database die helpt om gegevens op te slaan in gestructureerde tabellen. Het gebruikt een taal die SQL heet (Structured Query Language) om gegevens te beheren en te bewerken. Dit betekent dat je makkelijk informatie kunt opslaan, opvragen en aanpassen in de database.

### 5.4.1 wat zijn de voordelen van mysql?

1. **Betrouwbaarheid en stabiliteit:**

MySQL staat bekend om zijn betrouwbaarheid en stabiliteit. Het kan grote hoeveelheden gegevens goed beheren zonder dat het langzaam wordt of crasht. Dit maakt het een goede keuze voor bedrijven die veel data hebben en deze veilig willen opslaan.

1. **Flexibele zoekmogelijkheden:**

Met MySQL kun je moeilijke zoekopdrachten en transacties uitvoeren. Dit betekent dat je gemakkelijk verschillende gegevens kunt combineren en analyseren. Hierdoor krijg je krachtige mogelijkheden om inzicht te krijgen in je informatie.

1. **Weer veel ondersteuning te vinden:**

MySQL heeft een grote community van gebruikers en ontwikkelaars. Dit betekent dat je veel hulp kunt vinden als je vragen hebt. Er is veel documentatie beschikbaar, wat het makkelijk maakt om meer te leren en de database goed te beheren.

### 5.4.2 heeft het dan nog nadelen?

1. **Moeilijkheden bij grootschalige toepassingen:**

Als je werkt met zeer grote datasets of veel bezoekers op je website, kan MySQL moeilijk te schalen zijn. Dit betekent dat je mogelijk extra instellingen en optimalisaties moet doen om ervoor te zorgen dat alles soepel blijft draaien.

1. **Complexiteit van geavanceerde functies:**

Sommige van de meer geavanceerde functies van MySQL kunnen ingewikkeld zijn, vooral voor beginners. Het kan even duren voordat je deze goed begrijpt en kunt toepassen. Het is belangrijk om geduldig te zijn en jezelf de tijd te geven om te leren.

### 5.4.3 hoe lossen we dat op en wat is ons plan?

1. **Voorbereiding op grootschalige gebruik:**

Als we weten dat we met veel gegevens of gebruikers gaan werken, kunnen we van tevoren plannen. We kunnen optimalisaties toepassen en de database zo inrichten dat deze beter bestand is tegen een hoge belasting

1. **Leren en oefenen:**

Om de complexiteit van geavanceerde functies te overwinnen, kunnen we tutorials en online cursussen volgen. Door te oefenen met voorbeelden en scenario’s kunnen we onze vaardigheden verbeteren en beter leren omgaan met MySQL.

**Samenvatting en ons plan:**

MySQL is een krachtige database voor het opslaan en beheren van gegevens. Het biedt veel voordelen zoals betrouwbaarheid en flexibele zoekmogelijkheden. Maar we moeten ook letten op de uitdagingen, zoals schaling en complexiteit. Door goed te plannen en te leren, kunnen we het meeste uit MySQL halen en effectief werken met onze gegevens!

## 5.5 Keuze frameworks en libraries conclusie en toelichting

In ons project kiezen we voor verschillende handige tools om een sterke en gebruiksvriendelijke applicatie te maken. **React.js** helpt ons om kleine onderdelen te gebruiken, waardoor de applicatie makkelijk te bouwen en interactief is voor gebruikers. Voor het laten zien van gegevens gebruiken we **Chart.js**, waarmee we informatie op een mooie en duidelijke manier kunnen weergeven.

Aan de achterkant van de applicatie gebruiken we **Node.js met Express**. Dit zorgt ervoor dat we snel en efficiënt kunnen werken, waardoor gebruikers minder moeten wachten. **MySQL** helpt ons om gegevens veilig op te slaan en gemakkelijk te bewerken.

Door deze tools samen te gebruiken, maken we een applicatie die kan groeien en goed werkt. Dit zorgt ervoor dat we aan de behoeften van onze gebruikers kunnen voldoen. Samen vormen deze tools een sterke basis voor een betrouwbare en handige applicatie.

## 5.6 Hoofcomponenten

We hebben twee belangrijke onderdelen in ons systeem:

**HC-01:** - Client: Dit is de interface waarmee gebruikers werken, zoals een website of een app.

**HC-02:** - Server: Dit is de computer die verzoeken van de client ontvangt en de gegevens beheert.

## 5.7 Systeemomgeving

De tools en programma's die we gebruiken zijn:

**SO-01:** - Node.js: Dit helpt ons om JavaScript op de server te draaien.

**SO-02:** - Express.js API: Dit maakt het makkelijker om gegevens te versturen tussen de client en de server.

**SO-03:** - React.js: Dit helpt ons om de gebruikersinterface te bouwen.

**SO-04:** - MySQL database: Dit slaat alle belangrijke gegevens veilig op.

**SO-05:** - Chart.js (voor datavisualisatie): Dit gebruiken we om gegevens op een mooie manier weer te geven, zoals grafieken.

**SO-06:** - JSON Web Tokens: Dit helpt ons om gebruikers veilig te laten inloggen (als dat nodig is).

## 5.8 Onderlinge samenwerking tussen hoofdcomponenten en systeemomgeving

**Client (React.js):**

* Dit is wat de gebruiker ziet en waarmee hij of zij werkt. De client stuurt verzoeken naar de server om gegevens op te vragen. Met Chart.js laat de client de gegevens mooi zien en zorgt JWT ervoor dat alleen de juiste gebruikers toegang hebben.

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

* De server ontvangt de verzoeken van de client, verwerkt deze en praat met de MySQL-database om de nodige gegevens op te halen of op te slaan. Ook zorgt de server ervoor dat de juiste gebruikers inloggen met JWT.

**MySQL Database:**

* Dit is waar we alle gebruikersgegevens en informatie over zonnepanelen veilig opslaan. De server vraagt deze gegevens op wanneer dat nodig is..

**De client en de server praten met elkaar via API's. Wanneer de client iets wil weten, vraagt het de server om de gegevens. De server haalt deze gegevens uit de database en stuurt ze terug naar de client, zodat ze kunnen worden weergegeven.**

# 6.0 Database

## 6.1 Logische datastructuur (ERD)

**Afbeelding met tekst, schermopname, diagram, Lettertype

Automatisch gegenereerde beschrijving**

**Verdere toelichting over alle “relaties” (connecties) in de database:   
  
Tables:**

PK = Primary Key  
FK = Foreign Key

* User:- user\_id (PK)  
  - role\_id (FK naar Role)  
  - solar\_panel\_id (FK naar Solar\_Panel)  
  - battery\_id (FK naar Battery)  
  - name  
  - email  
  - password  
  - address
* Role:  
  - role\_id (PK)  
  - role\_name ("Admin", "User" etc)  
  - permissions (optioneel, maar kan worden toegevoegd om te definiëren waartoe elke rol toegang heeft)  
  - Solar\_Panel
* Solar\_Panel:  
  - solar\_panel\_id (PK)  
  - user\_id (FK naar User)

- panel\_capacity (in kW)

- installation\_date

- location (for simmulatie miss handig)

* Energy\_Generation:

- generation\_id (PK)

- solar\_panel\_id (FK naar Solar\_Panel)

- generation\_date (datum)

- energy\_generated (in kWh)

* Battery:

- battery\_id (PK)

- user\_id (FK naar User)

- battery\_capacity (in kWh)

- installation\_date (datum)

* Energy\_Storage:

- storage\_id (PK)

- battery\_id (FK naar Battery)

- storage\_date (datum)

- energy\_stored (in kWh)

- energy\_consumed (in kWh)

* Energy\_Sale\_Simulation:  
  - simulation\_id (PK)  
  - user\_id (FK naar User)  
  - price\_id (FK naar Energy\_Price)  
  - simulation\_date (datum)  
  - energy\_stored\_for\_sale (in kWh)  
  - sale\_price (in valuta, dynamisch opgehaald uit de API)  
  - revenue\_estimate (berekend)
* Energy\_Price:

- price\_id (PK)

- price\_date (Datum)

- energy\_price (per kWh)

- price\_source (API)

**Relaties:**

1. Gebruiker naar zonnepaneel: één gebruiker kan één zonnepaneelinstallatie hebben.
2. Gebruiker naar rol: elke gebruiker heeft één rol (beheerder of gewone gebruiker).
3. Rol naar gebruiker: een rol kan aan meerdere gebruikers worden gekoppeld.
4. Zonnepaneel naar energieopwekking: een zonnepaneel kan dagelijks energie opwekken.
5. Gebruiker naar batterij: een gebruiker kan één/meerdere of geen batterij hebben.
6. Batterij naar energieopslag: een batterij kan dagelijks energie opslaan en verbruiken.
7. Energy\_Generation naar Energy\_Sale\_Simulation: de simulatie gebruikt historische energieopwekkingsgegevens om de energieverkoop te schatten.
8. Energy\_Price: de simulatie verwijst naar energieprijzen voor elke dag van de externe API.

**ERD-uitleg:**

1. Gebruiker: houdt gebruikersinformatie bij en koppelt deze aan hun zonnepaneel en batterij.
2. Zonnepaneel & energieopwekking: vertegenwoordigt de zonnepaneelgegevens en de dagelijkse opwekkingsgeschiedenis.
3. Batterij & energieopslag: beheert de batterijcapaciteit, opgeslagen energie en verbruikte energie elke dag.
4. Energy\_Sale\_Simulation: voert simulaties uit op basis van batterijgebruik en energieprijzen om te berekenen of de gebruiker energie moet verkopen.
5. Energy\_Price: slaat de prijs van energie op, dynamisch opgehaald van een externe API voor simulatiedoeleinden.
6. Je kan dit model uitbreiden indien nodig, bijvoorbeeld om de verkooptransacties bij te houden als de gebruiker daadwerkelijk energie gaat verkopen.

Vol. Blz.

## 6.2 Gekozen implementatie en onderbouwing

In dit project hebben we gekozen voor een relationele database in plaats van een NoSQL-oplossing, omdat een relationele database beter past bij het vastleggen van relaties tussen dingen zoals gebruikers, simulatiegegevens en feedback. NoSQL maakt het soms wat lastiger om die relaties goed vast te leggen, wat kan zorgen voor dubbele gegevens en een rommelige database. Hoewel NoSQL sneller kan zijn met grote hoeveelheden data, gaat het in dit project om kleinere datasets waarbij het belangrijk is dat de data altijd kloppen.

We hebben MySQL gebruikt voor de database, omdat het betrouwbaar is en goed werkt voor dit soort projecten. We hebben MySQL Workbench gebruikt om ons ERD (Entity Relationship Diagram) te maken, en een handige functie daarvan is dat je meteen de database kunt genereren vanuit dat diagram. Hierdoor kan iedereen makkelijk dezelfde database gebruiken zonder dat er fouten ontstaan.

Voor de back-end gebruiken we Node.js en Express, wat goed samenwerkt met MySQL. Dit maakt het makkelijker om API-verzoeken en gebruikersgegevens te beheren

# 7.0 Coding standards

In dit project werken we met regels en afspraken om ervoor te zorgen dat we op een gestructureerde manier samen aan de software kunnen werken. Door deze regels te volgen, kunnen we ervoor zorgen dat alles soepel verloopt, dat fouten makkelijker worden gevonden en opgelost, en dat iedereen weet wat er van hen wordt verwacht. Het doel is om iedereen op dezelfde manier te laten werken, zodat er geen verwarring ontstaat over hoe de code moet worden geschreven of hoe we samenwerken.

**CO-01: Waarom zijn deze regels belangrijk**?

* Als iedereen op een andere manier de code schrijft of opslaat, kan het moeilijk zijn om veranderingen bij te houden of fouten te vinden. Ook kan het voorkomen dat verschillende versies van de software door elkaar gaan lopen, wat tot problemen kan leiden. Daarom hebben we duidelijke afspraken gemaakt over hoe we onze code opslaan, hoe we samenwerken aan nieuwe functies, en hoe we testen uitvoeren voordat iets wordt toegevoegd aan de belangrijkste versie van de software.

## 7.1 Branching & commits

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijvingWanneer meerdere mensen aan een project werken, moet iedereen efficiënt kunnen samenwerken zonder per ongeluk elkaars werk te verstoren. Dit doen we door met "versies" van de code te werken, die we **"branches"** noemen. Door gebruik te maken van deze branches, krijgt iedere aanpassing een eigen plek waar er aan gewerkt kan worden. Pas als alles goed is getest en gecontroleerd, voegen we die aanpassingen toe aan de hoofdversie van de software. Op deze manier blijft het project overzichtelijk en kan iedereen veilig aan zijn/haar deel werken.

### 7.1.1 werken mmet verschillende versies (branches)

Er zijn verschillende soorten branches, elk met een specifiek doel. Hier leggen we uit hoe we deze branches gebruiken:

* **Hoofdversie (Main Branch):**  
  Dit is de belangrijkste versie van de software. Alles wat hier in staat, is helemaal klaar en goedgekeurd. Pas nadat iets grondig getest is, voegen we het toe aan deze hoofdversie, zodat we zeker weten dat de software goed werkt.
* **Testversie (Test Branch):**  
  Voordat iets naar de hoofdversie gaat, wordt het eerst getest in de testversie (Test Branch). Hier kunnen we fouten vinden en verbeteren voordat de wijziging definitief wordt.
* **Ontwikkelversie (Dev Branch):**  
  Hierin werken we aan nieuwe ideeën en functies. Als iemand een nieuwe functie of verbetering heeft gemaakt, wordt die in de development Branch gezet. Andere teamleden kijken dan mee om te controleren of alles goed werkt.
* **Feature-versies (Feature Branches):**  
  Voor elke nieuwe functie maken we een aparte feature versie. Dit zorgt ervoor dat we aan één ding tegelijk kunnen werken zonder dat andere delen van de software worden beïnvloed.
* **Documentatieversie (Doc Branch):**  
  Hierin bewaren we alle documentatie. Denk hier bijvoorbeeld aan project-portfolio’s, PID, Iteratiedocumenten en andere informatie die belangrijk is voor het project.

### 7.1.2 Wijzigingen opslaan en documenteren (Commits)

**WIJZ-O1: Wat is een Commit?**

* Elke keer dat we iets veranderen in de code, slaan we die verandering op als een "commit". Een commit is eigenlijk een soort opname van wat er precies is aangepast op dat moment. Het helpt ons om later terug te kijken en te zien wat we hebben gedaan.

**WIJZ-02: Voordelen van Commit.**

* Een commit is handig omdat je veranderingen kunt volgen en terugdraaien als dat nodig is. Stel je voor dat je aan een project werkt en iets fout gaat, dan kun je dankzij die "opname" makkelijk teruggaan naar hoe het was voordat de fout erin kwam.

**WIJZ-O2: Format van commit-berichten:**

* We gebruiken een vast format voor de commit-berichten. Dit betekent dat we altijd op dezelfde manier opschrijven wat we hebben veranderd, zodat het voor iedereen duidelijk is.
* Voorbeeld van de format:

<type>: <beschrijving>

[optionele inhoud] => Meer gedetailleerde uitleg over wat er in deze commit is veranderd.

[optioneel voetstuk] => Referenties naar issues of belangrijke notities worden opgenomen.

* + Type en beschrijving:
    - **feat**: Dit betekent dat er een nieuwe functie is toegevoegd.
      * Voorbeeld: *feat: nieuwe knop toegevoegd voor het versturen van berichten.*
    - **fix**: Dit geeft aan dat er een fout is opgelost
      * Voorbeeld: *fix: probleem opgelost bij het inloggen van gebruikers.*
    - **docs**: Als er iets is veranderd in de documentatie, gebruiken we dit label.
      * Voorbeeld: *docs: handleiding aangepast voor nieuwe gebruikers.*
    - **style**: Voor aanpassingen van de style.
      * Voorbeeld: *style: marges van knoppen verbeterd.*

**Voorbeeld:**

**feat**: toegevoegd wachtwoord-reset functie voor gebruikers  
Een nieuwe functie toegevoegd waarmee gebruikers hun wachtwoord kunnen resetten via een e-mail link. Er is een WachtwoordResetController aangemaakt die de stappen beheert om een resetverzoek in te dienen en de gebruiker naar een pagina te sturen om een nieuw wachtwoord in te stellen.   
De e-mailtemplate voor het resetten van wachtwoorden is ook toegevoegd.

## 7.2 Overige standards

Dit onderdeel bevat enkele extra afspraken over hoe de code en het project gestructureerd worden, om alles zo overzichtelijk en begrijpelijk te houden.

### 7.2.1 Controller/model conventies

**CM-01:** Controllers, de onderdelen die bepalen wat er gebeurt wanneer een gebruiker iets doet (zoals inloggen of een account maken), krijgen altijd een naam die begint met een hoofdletter, gevolgd door het woord ‘Controller’. **Bijv.:** Een controller voor gebruikers heet ***UserController***.

**CM-02:** Controllers worden gesorteerd op hun functie. Bijvoorbeeld: alle controllers die te maken hebben met de API (de manier waarop systemen met elkaar praten) komen in een aparte map genaamd **api**.

**CM-03:** - Models, de structuren waarin de data wordt opgeslagen (zoals gegevens van een gebruiker), krijgen ook altijd een naam die begint met een hoofdletter.  
**Bijv.**: Het model voor gebruikers heet ***User***.

### 7.2.2 Component(vue) conventies

Afbeelding met schermopname, Lettertype, Kleurrijkheid, Graphics

Automatisch gegenereerde beschrijving**CC-01:** Voor onderdelen die in de software terugkomen (bijv. knoppen, header, formulieren), worden alle *React* componenten netjes georganiseerd in submappen per categorie. Zo blijft alles overzichtelijk.

**CC-02:** De naam van een component begint altijd met een hoofdletter en volgt het *PascalCase* format, waarbij ieder woord met een hoofdletter begint.  
**Bijv.**: ***LoginFormulier***.

**CC-03:** Wanneer componenten in de code worden geïmporteerd of geregistreerd, wordt altijd dezelfde naamconventie gebruikt, namelijk *PascalCase*.

### 7.2.3 css conventies (Style)

**SA-01:** Voor elke component maken we een aparte CSS-bestand dat dezelfde naam heeft als de component. Dit helpt om de stijlen gemakkelijk te vinden en te koppelen aan de juiste component.

**SA-02:** De structuur is als volgt:

* Voor een component met de naam ***UserProfile*** maken we een map met de naam ***Style***, en daarin een bestand genaamd ***UserProfile.css***.

Afbeelding met tekst, schermopname, scherm, Multimediasoftware

Automatisch gegenereerde beschrijving**SA-03:** Deze indeling maakt het makkelijker om stijlen te beheren, aangezien elke component zijn eigen styling bevat en duidelijk te herkennen is.

### 7.2.4 Navigatie conventies

**NC-01:** De URL's (de link naar een bepaalde pagina) zijn gestructureerd zodat je altijd weet wat waar te vinden is. Dit werkt op de volgende manier:

* **controller/action**:
  + **Bijv.***: user/login* || leidt naar de pagina waar een gebruiker kan inloggen.
* **controller/controllerparameter/action**: Hier is nog meer informatie toegevoegd, zoals een specifieke gebruiker.
  + **Bijv.**: *user/123/profile* || leidt naar het profiel van gebruiker 123.
* **controller/action/actionparameter**: Nog een extra toevoeging om bijvoorbeeld een actie te specificeren.

**NC-02:** Voor sectie van de website die speciaal voor beheerders zijn, voegen we altijd ‘**admin’** toe aan de URL.

* **Bijv.**: *admin/user/edit* || voor het bewerken van gebruikers door een beheerder.

### 7.2.5 Overige standards

**OS-01:** Alle pagina's moeten *responsieve* zijn. Dit betekent dat de website goed werkt op verschillende schermgroottes, zoals op een computer, tablet of telefoon.

# 8.0 Eisen user interface

## 8.1 Look & Feel

De "look en feel" van de app is visueel weergegeven in wireframes. Dit zijn schetsen die laten zien hoe de gebruikersinterface eruitziet, met de plaatsing van knoppen, kleuren, en lettertypes. Ze helpen duidelijk te maken hoe de app er visueel uitziet en hoe je ermee kunt werken.

Het ontwerp moet gemakkelijk te gebruiken zijn en er aantrekkelijk uitzien, met een moderne stijl die past bij het merk van de app.

Hieronder ziet u een snippet van alle afbeeldingen van de wireframes, als u ze per stuk wilt bekijken, volg [deze link](https://app.visily.ai/projects/1505268c-3545-4b19-a086-2ea6c52beb8d/boards/1277773) naar de juiste locatie.

Snippet’s:

Afbeelding met tekst, software, schermopname, Multimediasoftware

Automatisch gegenereerde beschrijvingAfbeelding met tekst, software, Multimediasoftware, Computerpictogram

Automatisch gegenereerde beschrijving

## 8.2 Navigatie

De manier waarop je door de app navigeert, is weergegeven in onze flowchart. Deze diagram laat zien welke pagina’s er zijn en hoe ze met elkaar verbonden zijn. Je kunt zien welke routes gebruikers kunnen volgen om door de app te gaan. Belangrijke onderdelen zijn het hoofdmenu, sub-menu’s, en links naar belangrijke secties zoals het dashboard, je account, en de instellingen.

De flowchart:

Afbeelding met diagram, lijn, schermopname, Plan

Automatisch gegenereerde beschrijving

Voor get gemak van de klant, hebben wij ook een wireflow gemaakt, hierop kunt u in de flowchart zien welke content waar zal worden opgehaald/ zichtbaar zal zijn.

Vol. Blz.

De wireflow:

A diagram of a company

Description automatically generated with medium confidence

## 8.3 Invoer

De invulvelden en knoppen van de app zijn beschreven in een stijlgids. Deze gids geeft regels voor hoe alles eruit moet zien en werken, zodat het ontwerp consistent blijft. Er wordt vooral gelet op dat het makkelijk te gebruiken is en toegankelijk voor iedereen.

In de stijlgids staat ook informatie over kleuren, lettergroottes, en hoe dingen reageren als je er met de muis overheen gaat. Dit zorgt voor een gelijkmatige gebruikerservaring

Deze kunt u in onze GitHub repo vinden; [het document](https://github.com/BrigitteHx/20WekenProject/tree/doc/doc/stijlgids). Toch zal ik alle inhoud van het document hieronder nog eens weergeven.

**Inleiding:**

Deze stijlgids is bedoeld om consistentie en een professionele uitstraling te waarborgen voor de Solar panel simulation web application.

De richtlijnen in dit document helpen bij het creëren van een samenhangende visuele identiteit door middel van logo-gebruik, kleuren, typografie en andere ontwerpelementen.

Deze stijlgids is gebaseerd op de laatste trends in webdesign en is speciaal ontworpen voor een gebruiksvriendelijke, intuïtieve interface voor eindgebruikers. Zoals ook zichtbaar in de eerder ontworpen wireframes.

In deze stijlgids hebben wij keuzes gemaakt die overeenkomen met de huisstijl van de Lucrasoft BV, wij hebben hun professionele uitstraling gevolgd. We hebben ons ook verdiept in de structuur van hun eigen websites (denk aan bijvoorbeeld; waar worden donkere kleuren weer gegeven en waar lichte?). Deze uitstraling hebben wij overgenomen en in deze stijlgids vast gelegd.

**Inhoudsopgave:**

1. Logo gebruik
2. Kleurcodes
3. Typografie
4. UI componenten
5. Taal
6. Grafieken en datavisualisatie
7. Navigatie
8. Iconen
9. Slot

**1. Logo gebruik:**

Officieel logo:

* Het officiële logo dat op het platform gebruikt moet worden, is het logo uit het branding-bestand ('branding\_and\_logo.png'). Zorg ervoor dat het logo duidelijk zichtbaar is, niet vervormd en passend geschaald.

Hoofd header:

* Plaats het logo in de linkerbovenhoek van de website zoals te zien in de wireframe.

Footer:

* Hetzelfde logo dient aanwezig te zijn in de footer.

Logo grootte richtlijnen:

* Header: 50px hoogte
* Footer: 40px hoogte
* Spatiëring: zorg voor 20px vrije ruimte rondom het logo om een rommelige uitstraling te voorkomen.

**2. Kleurcodes:**

Primaire kleur:

* HEX: #0a3a61 (donkerblauw)
* Gebruikt in de header, navigatiebalken en knoppen voor een strakke en professionele uitstraling.

Secundaire kleur:

* HEX: #f9b233 (oranje)
* Gebruikt voor highlights en call-to-actions zoals knoppen, grafieken of belangrijke informatie.

Achtergrondkleur:

* HEX: #ffffff (wit)
* Gebruikt voor de hoofdachtergrond voor een heldere gebruikerservaring.

Tekstkleur:

* Primaire tekst: #000000 (zwart)
* Secundaire tekst: #4f4f4f (grijs)
* Links/klikbare elementen: #0a3a61 (donkerblauw)

**3. Typografie**

Hoofdlettertype:

* Lettertypefamilie: Arial, sans-serif (volgens de lucrasoft richtlijnen).
* Fallback fonts: Helvetica, sans-serif.
* Reden: Arial is een universeel beschikbaar en webveilig lettertype dat voldoet aan professionele standaarden.

Titel/headers:

* Hoofdtekengrootte (h1): 32px
* Subkoppen (h2): 24px
* Sectietitels (h3): 20px
* Letterdikte: Vetgedrukt voor alle heading-niveaus.
* Letterkleur: Gebruik de primaire kleur #0a3a61 voor alle kopteksten.

Lichaamstekst:

* Lettergrootte: 16px
* Letterdikte: Normaal.
* Regelhoogte: 1.5 voor betere leesbaarheid.
* Kleur: #000000 (zwart)

Knoptekst:

* Lettergrootte: 14px
* Letterdikte: Vetgedrukt.
* Letterkleur: #ffffff (wit) op een achtergrond van #f9b233 (oranje).

**4. UI componenten**

Knoppen:

* Gebruik afgeronde hoeken met een straal van 5px.
* Primaire knopachtergrondkleur: #f9b233
* Hover state van de knop: verander naar een lichtere oranje kleur #ffc759.

Formulieren:

* Invoervelden moeten een border-radius van 3px hebben, met een randkleur van #cccccc.
* Placeholder-tekst moet de kleur #4f4f4f gebruiken.

Layout en spatiëring:

* Container breedte: gebruik een maximale breedte van 1200px voor contentgebieden, gecentreerd met marges aan beide kanten voor grote schermen.
* padding: standaard padding tussen secties moet 20px zijn.
* marges: 30px marge tussen elementen zoals knoppen en grafieken voor een schone spatiëring.
* grid: gebruik een 12-koloms gridsysteem voor responsieve layouts.

**5. Taal**

* Primaire taal: engels voor alle openbare content op de webapp.
* Uitzonderingen: voor specifieke regio's kan lokalisatie van toepassing zijn.

**6. Grafieken en datavisualisatie**

De grafieken en datavisualisaties in de wireframe moeten een duidelijke en minimalistische opzet volgen om gebruikers snel inzicht te geven.

Lijngrafieken:

* Lijnkleur: oranje #f9b233
* Achtergrond: Transparant.
* Labels: Gebruik kleine tekst met een lettergrootte van 12px in #4f4f4f.

Cirkeldiagrammen:

Gebruik een complementair kleurenpalet:

* Verbruikte energie: oranje #f9b233
* Beschikbare energie: blauw #0a3a61.

**7. Navigatie**

Hoofdnavigatiebalk:

* Vast aan de bovenkant met een achtergrondkleur van #0a3a61.
* Actieve links moeten worden gemarkeerd met #f9b233.

Footer design:

Achtergrond van de footer:

* Kleur: #f9f9f9 (lichtgrijs).

Footer tekst:

* Lettergrootte: 14px
* Letterkleur: #4f4f4f

**8. Iconen**

* Gebruik alleen officiële social media-iconen met een grootte van 24px, netjes uitgelijnd in het voettekstgebied.

**9. Slot**

De richtlijnen in deze stijlgids moeten strikt worden nageleefd om uniformiteit en herkenbaarheid van de webapplicatie te waarborgen.

Consistente toepassing van het logo, de kleuren, typografie en UI-componenten zal bijdragen aan een prettige gebruikerservaring en een sterke merkidentiteit voor de Solar panel simulation web application.

Bij vragen over het gebruik van de stijlgids kan contact worden opgenomen met het ontwerpteam.

# 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.

# 10.0 Bronnenlijst en auteurs

Hieronder vind u per onderwerp in dit bestand de gebruikte bronnen, wij hadden deze is los per onderwerp toegevoegd maar een bronnenlijst met structuur die bij de rest van het document past leek ons uiteindelijk toch beter.

## 10.1 bronnen

**2.0 Applicatie:**

* Aangeleverd opdracht document van bedrijf
* Toelichting tijdens meeting met Dhr. Lucas Vos op 15-10-2024

**3.0 Agile/ Scrum:**

* [What is Agile? (pmi.org)](https://www.pmi.org/disciplined-agile/agile/whatisagile#:~:text=As%20a%20way%20of%20working%2C%20agile%20is%20an,methodologies%2C%20including%20Scrum%2C%20SAFe%20%C2%AE%2C%20and%20Extreme%20Programming.)
* [Definition of Done, uitleg + voorbeelden | Wat is Scrum?](https://watisscrum.nl/definition-of-done/)
* School lessen en leerpaden met toelichting verschillende werk methodieken

**4.0 Architectuur en software patronen:**

* Bronnen voor FP:
  + [Deep Dive into Functional Programming in Javascript - DEV Community](https://dev.to/alexmercedcoder/deep-dive-into-functional-programming-in-javascript-851)
  + [Functioneel programmeren in JavaScript - GeeksforGeeks](https://www.geeksforgeeks.org/functional-programming-in-javascript/)
* Bronnen voor MPA:  
  + [Getting Started – React (reactjs.org)](https://legacy.reactjs.org/docs/getting-started.html)
  + [Optimize React apps using a multi-layered structure - LogRocket Blog](https://blog.logrocket.com/optimize-react-apps-using-a-multi-layered-structure/)
  + [Single-Page Applications vs Multi-Page Applications: The Battle of the Web Apps - Mind Studios (themindstudios.com)](https://themindstudios.com/blog/spa-vs-mpa/)
  + [Multi Page Application with React | by Przemek Nowicki | ITNEXT](https://itnext.io/building-multi-page-application-with-react-f5a338489694)

**5.0 Frameworks en libraries:**

* Bronnen voor React:   
  + [React](https://react.dev/)
  + [React resources - Learn web development | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_resources)
* Bronnen Chart.js:  
  + [Chart.js | Chart.js (chartjs.org)](https://www.chartjs.org/docs/latest/)
* Bronnen voor Node.js met express Api’s:  
  + [Express - Node.js web application framework (expressjs.com)](https://expressjs.com/)
  + [Express/Node introduction - Learn web development | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction)
  + [How To Get Started with Node.js and Express | DigitalOcean](https://www.digitalocean.com/community/tutorials/nodejs-express-basics)
* Bronnen voor MySQL:   
  + [MySQL](https://www.mysql.com/)
  + [MySQL Tutorial (w3schools.com)](https://www.w3schools.com/MySQL/default.asp)

**6.0 Database:**

* <https://planetscale.com/learn/articles/mysql-vs-nosql>
* <https://www.mongodb.com/resources/basics/databases/nosql-explained/nosql-vs-sql>

1. **Coding standards:**

* <https://www.atlassian.com/nl/git/tutorials/saving-changes/git-commit>
* <https://www.w3schools.com/git/git_branch.asp>
* <https://www.geeksforgeeks.org/reactjs-components/>

1. **Eisen user interface:**

* <https://www.nngroup.com/articles/wireframe-stages>
* <https://www.interaction-design.org/literature/article/what-are-user-flows>
* <https://www.interaction-design.org/literature/topics/information-architecture>
* <https://www.uxpin.com/studio/blog/style-guides-in-ux-design/>

## 10.2 Auteurs

Auteur’s:

1. – Brigitte
2. – Israa
3. – Kevin
4. – Brigitte
5. – Brigitte
6. – Max
7. – Israa
8. – Marouane
9. – Brigitte & Israa
10. - Brigitte