# 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.

# 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

**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***.

**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.