Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i Data	Vårsemesteret, 2021 Åpen
Forfatter: Maksim Dreggevik	MAKSIM DREGGEVIK (signatur forfatter)
Fagansvarlig: Rong Chunming Veileder(e): Rong Chunming, Stian Øien	
Tittel på bacheloroppgaven: Integrasjon av solcelleenergi i Easee laderobot. Engelsk tittel: Easee Solar	
Studiepoeng: 20	
Emneord: Easee, solcelle, elbillading, strømstyring	Sidetall: 76 + vedlegg/annet: 0 Stavanger, 13.05.2021 (dato/år)

easee®
SOLAR

INTEGRASJON AV SOLCELLEENERGI I EASEE LADEROBOT

MAKSIM DREGGEVIK, VÅREN 2021

1. Sammendrag

Denne bacheloroppgaven hadde som formål å integrere solcelleenergien i styringsløkken til Easee sine smartprodukter. Målet var å lage et simuleringsverktøy, samt finne måter å regulere Easee laderoboten basert på strømforbruket og egenprodusert energi. Målet var også å optimalisere ladesyklusen for å oppnå at brukerne kunne utnytte energien i høyest mulig grad.

For å løse oppgaven ble det utviklet en nettbasert portal som kommuniserer med utvalgte Easee enheter via Websocket og Rest API kommunikasjonsprotokoller. Solcelle simulator er en “headless” raspberry pi enhet som mottar analoge signaler fra et lite solcellepanel, og via “port forwarding” verktøy kringkastes data videre til alle aktive klienter på internett.

Web grensesnittet er hovedsakelig ment som utviklingsverktøy internt i Easee og dermed har funksjonalitet hatt større fokus enn brukervennlighet.

2. Forord

Jeg vil takke samboeren min Marianne Alstveit for støtten og hjelp med oppgaveretting og ikke minst som har vært en fantastisk mor til vår datter gjennom hele prosessen. Takk til Easee som har gitt meg denne muligheten, og spesielt takk til Stian Øien og Daniel Ahlberg for veiledning og tilrettelegging. Jeg vil også takke Prof. Chunming Rong ved Universitet i Stavanger for råd og veiledning.

3. Innledning

3.1 Oppgavebeskrivelse

Flere og flere går til anskaffelse av solcellepanel for å gjøre det mulig å produsere store deler av sitt eget strømforbruk. Når slike muligheter er til stede er det viktig at strømmen i boligen utnyttes på en riktig måte. Når skal man lade elbilen? Når skal man bruke varmtvannsberederen? Når skal man benytte elektrisk oppvarming? Og når skal man selge overskuddsstrømmen tilbake til nettet igjen? Eller burde man det i det hele tatt?!

Begrepet «kortreist strøm» betyr at energien blir produsert lokalt og ikke trenger en lang reise for å bli konsumert. Denne løsningen skal utformes på en slik måte at bruk av den blir det mest fordelaktige valget for sluttbrukeren. I tillegg til batteriet i elbilen kan varmtvannsbereder anses som energibeholder der oppvarming av den vil være energilagring. Ved god nok planlegging og tilrettelegging kan både oppvarming av varmtvannsberederen, og lading av elbil utsettes til husstanden har nok overskuddsstrøm så mye det lar seg gjøre.

3.2 Litt om Easee systemet

Fig. 1. Easee Laderobot

Easee laderobot er en smart og kraftfull elbillader som kan lade på både 1- og 3-fase og støtter lading opp til 22 kW. Tidspunktet for lading kan enten settes manuelt av brukeren, eller ved hjelp av mer dynamiske tredjepartstjenester som for eks. via Tibber, som kan lade når strømprisene er estimert lavest.

Fig. 2. Easee Equalizer

Easee equalizer er husets hjerte som sørger for å bruke all ledig kapasitet til lading uten at hovedsikringen ryker. Den kan enten kobles til strømmåleren via HAN-porten, eller eksterne strømsensorer for å kunne gi en bedre oversikt over hva som blir brukt i en husstand.

Både laderobot og equalizer har en kontinuerlig tilkobling til skyen via cellular/WiFi, slik produktene kan kommunisere med hverandre uten å ta hensyn til plassering. Både elbillader og equalizer har API inngangsport som gjør det mulig å sende kommandoer f.eks. for å redusere tilgjengelig strømmengde, eller sette lading på pause. Det går også an å lese blant annet sanntidsdata som spenninger, strømmer og temperaturer på alle faser.

3.3 Mål

Fig. 3. Konseptet

Målet med dette prosjektet var å finne en god algoritme som kunne øke eller strupe dynamisk tilgjengelig strøm til elbilen, basert på mengden lokal produksjon og øvrig forbruk i boligen. Oppgaven ble bygget rundt “on-grid” inverter prinsippet, som betyr at lagring av strøm ikke er tatt i betraktning og all energien går direkte fra kilden til forbruket. Noen caser å være bevisst på var:

- Dersom husstanden har lavt energiforbruk og egenproduksjon av energi er tilstrekkelig, skal elbilen lades i høyest mulig grad på egenprodusert energi.
- Dersom elbilen er koblet til laderen, samtidig som forbruket i husstanden er økende behøver anlegget mer energi enn den egenproduserte strømmen kan dekke, skal elbilen lades med en forsyning fra begge kildene.
- Dersom energiforbruket i husstanden er kritisk høy, skal algoritmen kunne forutse om egenproduksjonen har stigende eller synkende trend. Og ut i fra dette enten la bilen lade videre eller pause ladingen.

I tillegg til algoritmen som dynamisk skal håndtere de forskjellige tilstandene som er presentert ovenfor, var målet å bygge et brukergrensesnitt i form av web interface eller en portal som ville gi enkel oversikt over den dynamiske prosessen, samt la brukeren simulere visse prosesser. Det er viktig å presisere at for hyppig regulering kan lage svingninger på strømnettet, og bør dermed tas i betraktning og unngås dersom mulig.

3.4 Motivasjon

Elbil salget i Norge og andre land har tatt av de siste 10 årene, som har hatt stor innvirkning på både strømnettet og økonomien. Hvis økningen av elbiler i det private fortsetter i samme vekstrate, vil nåværende strømnett ikke klare å takle de store belastningene som vil by på store utfordringer i fremtiden. Løsningen vil være å bygge ut strømnettet, som vil føre til enorme kostnader for samfunnet, og enda mer ødeleggelse av naturen. Motivasjonen for denne oppgaven var å lære mer om energiutnyttelse av fornybar energi i de private hjem, og kanskje bidra til en bedre fremtid.

4. Teori

Dette kapitlet skal gi leseren forenklet teoretiske forklaringer om forskjellige komponenter og teknologier som ble brukt til gjennomføring av prosjektet. Basert på hva som er relevant for gjennomføringen blir noen emner forklart mer grundig enn andre.

4.1 Teknologier

4.1.1 JavaScript

JavaScript[1] (JS) er et høynivå programmeringsspråk som hovedsakelig brukes av nettlesere, men som også ble mer populær for bruk av andre runtime-miljøer etter lanseringen av rammeverk som Nodejs. JS gjør at nettsider og webapplikasjoner blir mye mer reaktive og dynamiske. Med pakkebehandling som npm, kan utviklere installere hvilken som helst av de nesten 1.5 millioner bibliotek i prosjektet sitt. Selv om JS har noen sikkerhetsutfordringer som XSS (cross-site scripting) og mistillit relasjon mellom klienten og serveren, er de største nettleserne som Firefox, Chrome og Safari fortsatt sterkt avhengig av JS.

Vanligvis består en nettside av tre grunnleggende komponenter som er HTML, CSS og JavaScript. Med disse komponentene er det mulig å lage alt fra enkle statiske nettsider til mer avansert nettapplikasjoner. Etter hvert som variasjonen av enheter som ble brukt til å betjene nettsteder økte, ble det mer komplisert å lage standard kode for alt. Noe måtte gjøres for å standardisere bransjen, og slik fikk JavaScript-rammeverk sitt gjennombrudd. Et rammeverk tilbyr som regel en rekke pre definerte funksjoner som gjør det lettere å skrive kode, og som samler koden til noe hver plattform kan forstå. I dag finnes det rammeverk som både kan brukes til bygging av webapplikasjoner, og i tillegg kan kompileres til mobile “native” applikasjoner.

4.1.2 Vue

VueJS[2] (uttales ***/vju:/***, likt som det engelske ordet “view”) Utviklet av Evan You med åpen kildekodelisens. Rammeverket har balansert ressursforbruk, og blir brukt til å bygge brukergrensesnitt og enkeltsides webapplikasjoner eller SPA (single-page application). Vue

har en stor brukerbase som gjør det enkelt å finne eksempler på nesten ethvert tilfelle, og det er enkelt å lære.

Det er flere måter å sette opp et vue prosjekt på. CLI (Command-line interface), er kanskje den mest vanlige måten. Som navnet antyder, går installasjonsprosessen gjennom konsoll eller terminal, og lar brukerne konfigurere prosjekter før den blir opprettet. Det er også mulig å sette opp prosjektet via grafisk verktøy (vue-ui) som gir en mer brukervennlig opplevelse. Når et prosjekt er opprettet, vil brukeren få servert en start mal som en type "hello-world" prosjekt som startes via terminal med en npm kommando, `npm run serve`.

```
C:\Users\Default\Documents
λ vue --version
@vue/cli 4.5.12

C:\Users\Default\Documents
λ vue create test-project
```

Et typisk vue prosjekt som er opprettet gjennom vue-cli vil være komponentbasert, som betyr at brukeren vil opprette ny "visning" for hver side eller nettadresse. Visninger (views) inneholder vanligvis flere komponenter, avhengig av applikasjonen. En komponent (component) i en visning kan for eksempel være navigasjonsfelt eller en bunntekst. Siden navigasjonsfeltet sannsynligvis vil bli brukt i flere visninger, er det en god ide å gjøre den gjenbrukbar.

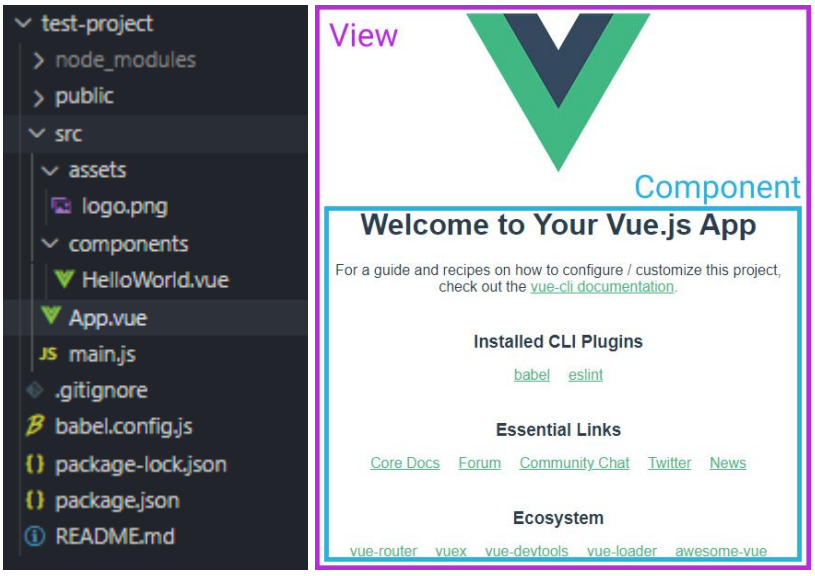
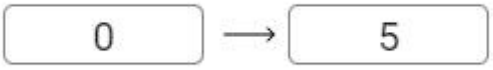


Fig. 4. Vue komponenter

Kode utkast 1 viser hvordan en bruker noen av vue funksjonene til å lage et responsivt program. Når brukeren klikker på knappen, vil tallet som vises på knappen øke.

```
<template>
  <div>
    <button @click="addOne()" id="btn">{{ count }}</button>
  </div>
</template>

<script>
export default {
  data: () => ({
    count: 0
  }),
  methods: {
    addOne() {
        this.count++
    }
  }
};
</script>

<style scoped>
#btn {
    margin: 0px 0px 50px 50px;
    width: 100px;
    height: 30px;
    font-size: 20px;
    border: 1px solid gray;
    border-radius: 5px;
}
</style>
```

Kode utkast 1.



Vue har en god del metoder for hendelseshåndtering både inn i script tag, og i template.

- Lifecycle hooks: **created**, **destroyed**, **updated**, **mounted**.
- Computed properties: **watch**, **computed**
- Event handlers som **methods** og **v-on** eller forkortelse @ som vist i eksempelet ovenfor.

Disse krokene, håndteringsenheter og metodene bruker vanlig JavaScript under panseret for å gjøre vue til et reaktivt rammeverk med optimal gjengivelse (re-rendering).

Vanlig praksis for webutvikling er å ha en global styling fil (.css) der styling for hele prosjektet blir definert. I vue er det vanlig å bruke “scoped" -alternativet slik at stiler definert i filen vil gjelde kun for komponenter i samme fil. Det er selvfølgelig mulig å ha en global styling fil også, da dette er avhengig av prosjektet. Det er ingen løsning som passer alle tilfeller.

4.1.3 Routing

Vue router[3] støtter muligheten fullt ut til å bygge, og betjene SPAer og gir grensesnittet for å endre og vise forskjellige sider basert på url-banen. Denne funksjonen gjør det mulig å betjene forskjellige sider eller komponenter uten at brukerne må oppdatere nettleseren. Et godt eksempel er SPA e-postklient. Når en bruker går inn i innboksen sin, i stedet for å laste hver eneste e-post på en gang, laster applikasjonen bare e-postene som er rettet mot eller klikket av brukeren. Det er mange fordeler med dette, det viktigste er kanskje hastighet eller hvor raskt applikasjonen lastes inn.

Vue router er også veldig nyttig for å kunne lage vaktfunksjoner (guard). Ved hjelp av metasegment er det mulig å lage logikk som vil kontrollere om brukeren oppfyller kravene for å gå videre til spesifikke baner eller ikke. Det er spesielt nyttig når applikasjoner krever at brukeren må logge inn for å få tilgang til innholdet. Kode utkast 2 viser slik brukstilfelle. Router sjekker med databasen om brukerinformasjonen er gyldig eller ikke. Hvis det er gyldig rutes applikasjonen videre til dashbordet, hvis ikke forblir brukeren i påloggingsvisningen.

```
import Vue from "vue";
import VueRouter from "vue-router";
import Login from "../views/Login.vue";
import Dashboard from "../views/Dashboard.vue";
import NotFound from "../views/NotFound.vue";
import firebase from “firebase/app";

Vue.use(VueRouter);

const routes = [
    {
        path: "/",
        name: "login",
        component: Login,
        meta: {
            requiresGuest: true
        }
    },
    {
        path: "/dashboard",
        name: "dashboard",
        component: Dashboard,
        meta: {
            requiresAuth: true
        }
    },
    {
        path: "*",
        name: "NotFound",
        component: NotFound
    }
];

const router = new VueRouter({
    mode: "history",
    base: process.env.BASE_URL,
```

```
    routes
});

router.beforeEach((to, from, next) => {
    if (to.matched.some(record => record.meta.requiresAuth)) {
        if (!firebase.auth().currentUser) {
            next({
                path: "/",
                query: {
                    redirect: to.fullPath
                }
            });
        } else {
            next();
        }
    } else if (to.matched.some(record => record.meta.requiresGuest)) {
        if (firebase.auth().currentUser) {
            next({
                path: "/dashboard",
                query: {
                    redirect: to.fullPath
                }
            });
        } else {
            next();
        }
    } else {
        next();
    }
});

export default router;
```

Kode utkast 2.

4.1.4 Vuex

En av utfordringene med SPAer er å kunne kontrollere tilstander på tvers av hele applikasjonen. Når nettstedet går over til en annen visning, vil alle lokale variabler fra forrige visning bli slettet. Vuex[4] fungerer ved å ha en sentral lagring for tilstander. Det gir metoder som gjør det mulig for komponenter fra hele applikasjonen å få tilgang til tilstander til enten å lese eller skrive, liknende getters og setter fra andre programmeringsspråk. Måten koden blir skrevet i vuex legger også til god praksis for applikasjoner fordi tilstander skal kun endres gjennom mutations eller actions og hente data bare via getters. Vuex består hovedsakelig av fem komponenter:

- **state** er et enkelt tree struktur som lagrer data og tilstander.
- **getters** er for metoder som skal hente ut data og tilstander.
- **mutations** er kun ment for metoder som skal endre tilstander eller data.
- **actions** er veldig lik mutasjoner, den brukes mest i situasjoner der det er behov for asynkron.

- **modules** er en mer praktisk komponent som lar brukeren dele opp store filer for å ha konsistens og orden i koden.

```
import Vue from "vue";
import Vuex from "vuex";

Vue.use(Vuex);

export default new Vuex.Store({
    state: {
        data: null
    },
    getters: {
        getData: state => {
            return state.data;
        }
    },
    mutations: {
        updateData(state, payload) {
            state.data = payload
        },
    },
    actions: {},
    modules: {}
});
```

Kode utkast 3.

Kode utkast 3 og 4 viser hvordan komponenter samhandler med vuex filen.

```
<script>
export default {
  data: () => ({
  }),
  methods: {
    updateData(data) {
        this.$store.commit("updateData", data)
    }
  },
  computed: {
    getData() {
        return this.$store.getters.getData;
    }
};
</script>
```

Kode utkast 4.

4.1.5 Mixins

En annen nyttig funksjon som ikke er et bibliotek, men en del av vue kjernefunksjonaliteten er mixins. Noen ganger hender det at utviklere bruker lik kode flere steder, noe som kan føre til dårlig utviklingspraksis. Mixins gjør det mulig for utviklere å skrive funksjoner, som kan være f.eks methods eller computed i en .js fil. Disse frittstående funksjonene kan importeres til hvilke som helst vue komponent eller visning. Kode utkast 5 er et eksempel på hvordan innhold i mixins-fil kan se ut.

```
export default {
    methods: {
        addOne(count) {
            return count++
        }
    }
};
```

Kode utkast 5.

Kode utkast 6 er et eksempel på hvordan en importerer mixins-funksjoner til en vue komponent. Etter importen kan man kalle addOne funksjonen innenfor vue komponenten.

```
<script>
import func from "@/mixins/someFunc.js";

export default {
    mixins: [func],
};
</script>
```

Kode utkast 6.

4.1.6 Firebase

Firebase[5] er en utviklerplattform for nett- og mobilapplikasjoner som er ervervet av Google. De tilbyr mange forskjellige funksjoner som lagring av filer, hosting, database, autentisering og mye mer. Med bred språkstøtte (SDK's) og god dokumentasjon er firebase et enkelt valg for både erfarne og juniorutviklere. Firebase har forskjellige betalingsplaner og gir nok ressurser til prototyping og småskala applikasjoner. De har til og med to forskjellige typer databaser, som er sanntids database og cloud firestore. Disse har forskjellige prissystemer.

4.1.7 Python

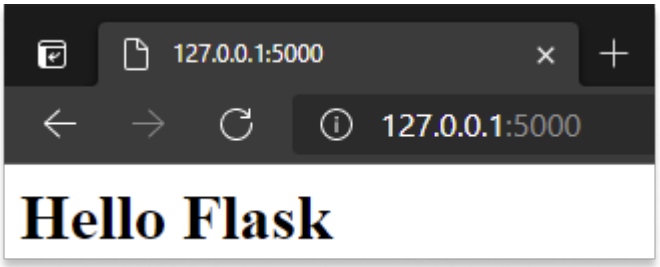
Python[6] er et objektorientert programmeringsspråk (OOP) som har eksistert i over 30 år. Med enkel syntaks er det et perfekt programmeringsspråk for nybegynnere. Biblioteker som numpy og pandas gjør python til et av de beste når det gjelder dataanalyse og datavisualisering. Ulempen til python er treghet, sammenlignet med språk som C++ eller Java. Dette gjør det mindre ønskelig for applikasjoner der hastighet er viktig for eksempel 3D spillutvikling. Ved hjelp av biblioteker som Flask, er python også et godt webutviklingsverktøy.

4.1.8 Flask

Er et rammeverk som tillater webutvikling med python. Det kan brukes til å bygge komplette webapplikasjoner eller kun backend delen. Flask[7] egner seg bra for å bygging av API-er, med noen få linjer er det mulig å sette opp en fungerende flask applikasjon. Kode utkast 7 viser hvor enkelt det er å sette opp HTTP Request metoden GET.

```python
from flask import Flask

app = Flask(__name__)

@app.route('/', methods=['GET'])
def home():
    return "<h1>Hello Flask</h1>"

if __name__ == '__main__':
    app.run()
```

```
* Running on http://127.0.0.1:5000/
```



Kode utkast 7.

4.1.9 Ngrok

Ngrok[8] er et verktøy som tillater eksponering av lokale webservere (for eks. localhost:5000) til nettet med egen statisk urlbane. Dette er nyttig for prosjekter der behovet for å eie server er nødvendig, i stedet for å bruke skybaserte tjenester som Heroku. Med offentlig IP som stadig endres av internettleverandøren, er det vanskelig å opprette en tunnel uten avtale med internettleverandøren. Ngrok gjør nøyaktig dette for å løse utfordringen. Med en gratis plan, tillater de kundene å bruke opptil en HTTP / TCP tunnel på tilfeldig url med opptil 40 tilkoblinger per minutt.

4.2 Datakommunikasjoner

4.2.1 HTTP

Hypertext Transfer Protocol[9] (HTTP) er en formløs protokoll som brukes til å levere data som HTML-filer, bilder, database dokumenter osv. Den bruker TCP-tilkobling med bekreftelse protokoll for å sikre at data blir levert i enveisrettet kanaler (unidirectional), selv om det kan oppstå feil. HTTP er kjent som grunnlaget for datakommunikasjonen til World Wide Web

eller som de fleste kjenner det, internett. I dag bruker nesten alle nettsider HTTPS, som er en sikrere versjon av HTTP og som er kryptert via Secure Socket Layer (SSL).

4.2.2 Websocket

Websocket[10] (ws) er en kommunikasjonsprotokoll som bruker en enkel TCP tilkobling for å gi toveis kommunikasjonskanaler (full-dupleks) mellom klient og server. Plassert i 7. lag av OSI-modellen sammen med HTTP-protokoll, ble ideen om websocket født ut av HTTP sine begrensninger. Selv om disse protokollene er ganske forskjellige, er websocket designet for å fungere over HTTP porter 443 og 80, samt for å støtte HTTP proxytjener og andre mellomledd, og dermed gjøre websocket kompatibel med HTTP protokollen. På lik linje med HTTP/S finnes det en sikrere spesifikasjon av websocket som er Websocket Secure (wss). Fig. 6 viser hvordan HTTP og WebSocket er forskjellige fra hverandre.

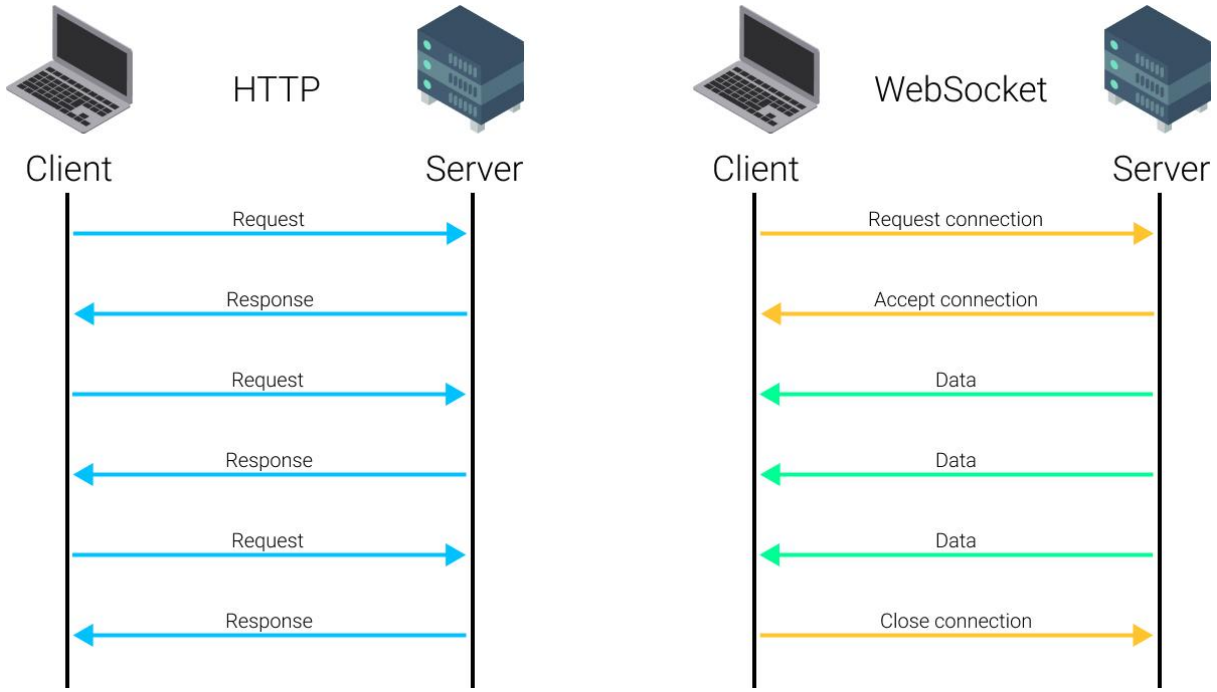


Fig. 6. Sammenligning mellom HTTP og Websocket

4.2.3 Socket.io

Socket.io[11] ble opprinnelig utviklet som et JavaScript-bibliotek for sanntids, toveiskommunikasjon mellom klient og server. Det finnes også flere klient implementeringer på andre språk, som vedlikeholdes av utviklere i forskjellige programmeringsnettsamfunn.

Socket.io oppretter en “long-polling” forbindelse som vil si at forbindelse skal vare lengst mulig. Når forbindelsen er opprettet, bytter den til best tilgjengelig tilkoblingsmetode, som i de fleste tilfeller er Websocket tilkobling. Socket.io er populært for bruk med applikasjoner som chatterom, nettcasinoer og mange andre applikasjoner som krever sanntid klient til server interaksjoner.

4.2.4 SignalR

Signalr[12] er enda et sanntids kommunikasjonsbibliotek basert på websocket, hovedsakelig for ASP.NET-miljøet. Sanntidsoppdatering fra Easee enheter kommer gjennom signalr til både mobilapplikasjon og webportal. Easee sin backend er skrevet i C#, som kan være årsaken til at signalr ble valgt.

4.2.5 Rest API

Selv om Web API er ekstremt populært innen webutvikling, har definisjonen API (Application Programming Interface) eksistert i lang tid. En av de enkleste eksemplene på API bruken er, sannsynligvis heis. Når en person velger ønsket etasje, kommer heisen til den spesifikke etasjen uten at personen trenger å vite hvordan. De samme prinsippene gjelder for Web API, dersom utviklere setter opp sluttpunkt med logikk kan alle som har tilgang til API sende forespørsler uten å måtte vite hvordan det fungerer. Web API bruker HTTP forespørsel/respons for å overføre data fra den ene enden til den andre. Sammen med respons vil det alltid være en statuskode som lar klienten vite hvordan forespørselen gikk. En av de mest populære måtene å integrere API på, er sannsynligvis Rest API[13]. Rest API er definert av sin arkitektoniske stil der metoder som GET, POST, PUT, DELETE bestemmer hvilken type handling forespørselen dreie seg om. Headeren brukes til å spesifisere forskjellige spesifikasjoner,, som f.eks at datatypen skal være i JSON- eller XML-format.

4.2.6 Serielt grensesnitt

Serielt grensesnitt[14] Er et fysisk grensesnitt mellom to digitale systemer. Kommunikasjonsgrensesnittet overfører spenningsserier gjennom ledninger der “1” representerer høy og “0” representerer lav, som egentlig er bits. Ved mottakende ende er dekoderen i

stand til å dekode data fra pulserende serier. I dette prosjektet ble det brukt en forbindelse mellom arduino nano og Raspberry pi via USB, som har støtte for serielt grensesnitt.

4.3 Elektronikk

4.3.1 Solcelle

Målet med solcellen i dette prosjektet var ikke å generere tilstrekkelig mengde kraft, men i stedet bruke den for simuleringsformål. Istedenfor å kun bruke tilfeldig genererte tall, er solcellen med på å gi et faktisk solmønster når solen er der. Solcellen som ble brukt i prosjektet (Fig. 7) er på størrelse med en mobiltelefon og er spesifisert til å generere opp til 2V 200mA. Dette passer perfekt til direkte tilkobling på en analog inngang på arduino nano. Det er også mulig å samle data fra solcellen for bruk i fremtidige forskning og utvikling.

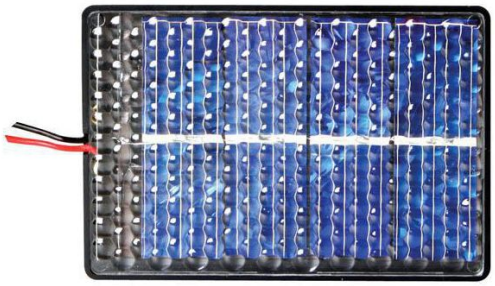


Fig. 7. Solcelle

4.3.2 Raspberry Pi

Raspberry Pi er en datamaskin som har omtrent samme størrelse som et bankkort og som er bygget på et enkelt kretskort. Enheten bruker 5V strømforsyning og har relativt lite strømforbruk, men er kraftig nok for mange applikasjoner. Hele ideen med RPi var å tilby skoler og institusjoner rimelig datamaskin til læringsformål. Den nyeste modellen (Fig. 8) kan spesifiseres med så mye som 8 GB ram og 1,5 GHz Quad Core-prosessor, noe som gjør den til en kraftig liten maskin. Med innebygd WiFi, Bluetooth og konfigurerbare GPIO er disse veldig populære for bruk innen IoT og andre automatiseringsprosjekter.

Fig. 8. Raspberry Pi 4

4.3.3 Arduino Nano

Arduino nano er en liten mikrokontroller bygget på AVR mikroprosessorarkitektur. I likhet med raspberry pi har arduino sitt maskinvaredesign og programvaren åpen lisens. Arduino er ikke så kraftig som rpi og kan heller ikke starte til et operativsystem. Den passer best for mindre oppgaver som for eksempel måling, beregning og datatransmisjon. Den har også noe som rpi ikke har, som er innebygd ADC (Analog til digital omformer). Med dette kan en måle spenninger opp til 5V som analoge signaler og med 10 bits oppløsning (0-1023). Det er mulig å kjøpe ADC moduler spesielt laget for tilkobling til Raspberry pi GPIO pinner. Den eneste grunnen til at denne metoden ble brukt er på grunn av tilgjengeligheten på det tidspunktet.

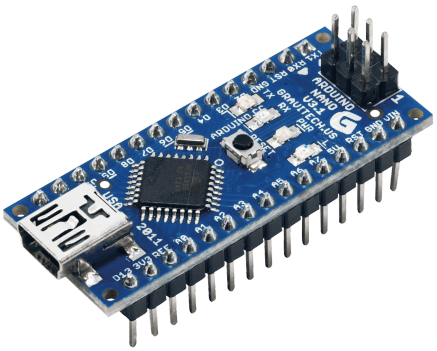


Fig. 9. Arduino nano

4.4 Annet

4.4.1 Vuetify

Vuetify[15] er et brukergrensesnittrammeverk bygget med Vue. Hensikten med dette rammeverket er å gi utviklere muligheten til å skape rik og engasjerende brukeropplevelse. Med et stort antall komponenter som navigasjonsfelt, knapper, tekstfelt osv,, er det veldig enkelt å bygge et konsekvent og moderne brukergrensesnitt. Hver komponent kan akseptere flere egenskaper for å tilpasse stilen og oppførselen.

4.4.2 vue-css-donut-chart

vue-css-donut-chart[16] er et lettvekter bibliotek som lar utviklere lage smultringdiagram raskt og effektivt. Med reaktiv data og stilendring gir denne pakken god brukeropplevelse samt sanntidspresentasjon. Dette biblioteket er laget spesifikt for vue rammeverket og i dette prosjektet representerte 0 til 100 datatilstand som et måleverktøy.

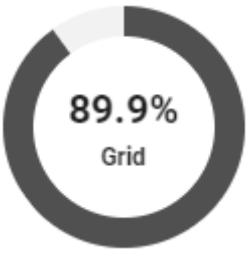


Fig. 10. smultringdiagram laget med vue-css-donut-chart

4.4.3 ApexCharts

ApexCharts[17] er et javascript bibliotek som lar utviklere lage dynamiske eller statiske grafer. Den har støtte for flere js rammeverk som gjør det lettere å integrere i et prosjekt med funksjoner og livskroker (lifecycle hooks). Det er mange diagramstiler å velge mellom, og i dette prosjektet ble arealdiagram uten utfylling brukt.

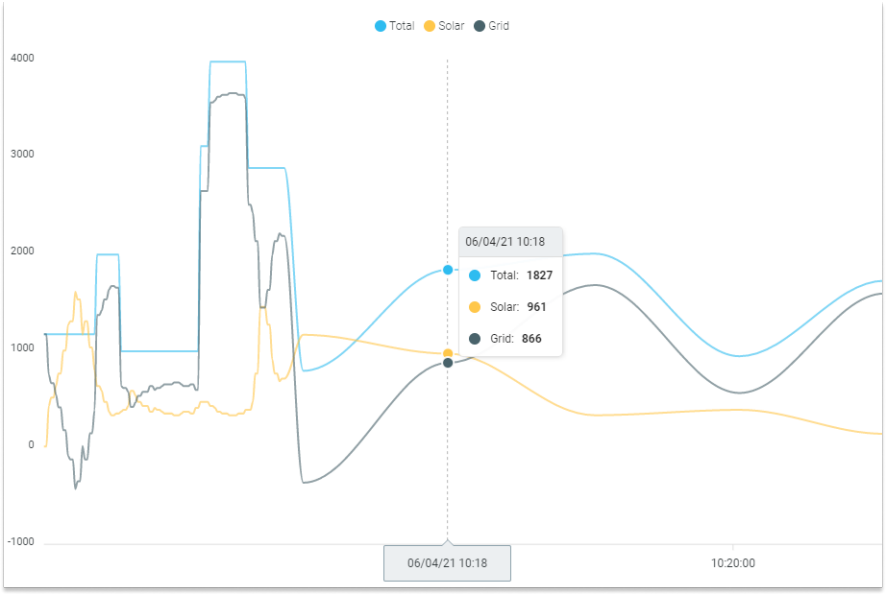


Fig. 11. Graf laget med ApexCharts

5. Utførelsen

Hovedmålet med dette kapittelet er å gi leseren en god oversikt over hvordan prosjektet ble utført og årsakene bak sentrale beslutninger. Dette kapittelet består av noen grundige tekniske forklaringer og noen lett forklarte scenarier med litt kode og diagrammer.

5.1 Planlegging

Planleggingen besto av kategorisering av arbeidsoppgaver, valg av teknologier, bestilling av deler og kalkulasjoner. Det var spesielt viktig å dele opp oppgavene i mindre underkategorier for å ikke miste oversikten samtidig som det så mer overkommelige ut. Diverse verktøy ble brukt i prosessen, men et av de som gjorde en av den størst nytten var Trello[18] tavlen (Fig. 12). Hovedkategorier som var oppdelt i mindre oppgaver, hjalp å holde fokuset kun på de tingene som måtte gjøres. Oppgaver har forskjellige fargekoder basert på oppgavens status.

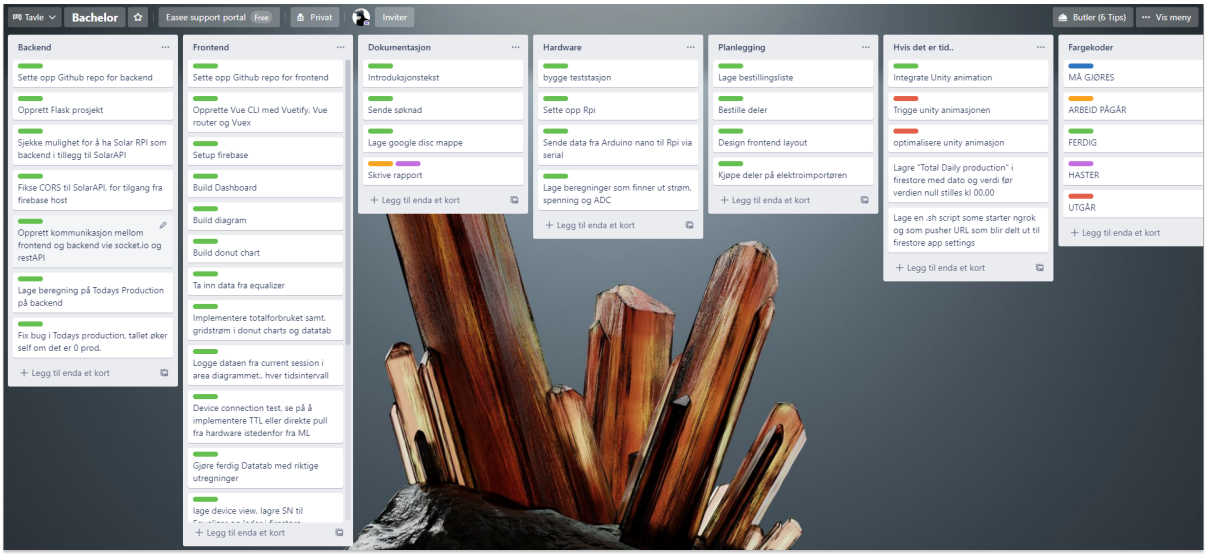


Fig. 12. Trello Tavle

5.2 Design

Med hovedfokuset på funksjonalitet ble utgangspunktet for design minimalistisk og ryddig tilnærming med modernt preg. Programmet som ble brukt til å lage design prototyper er Figma[19], som er et av de mest populære UI/UX verktøyene i industrien. Figma kan brukes til både visualisering og prototyping av dynamiske websider og mobilapper, men også til å lage vektorgrafikk. Selv om sluttresultatet ble noe annerledes, så hjalp det veldig å kunne eksperimentere i Figma før selve utviklingen ble gjennomført, både når det gjaldt tiden og det praktiske. En av mange nyttige funksjoner i Figma er muligheten til å hente ut CSS-kode som kan brukes i produksjonsmiljøet og kan bidra til økt tidsbesparelse. Nesten alle komponentene som har blitt brukt i prosjektet er hentet fra [4.4.1 Vuetify] tillegget, og som i bunn består av Material Design spesifikasjonene.

```
CSS

/* Rectangle 4 */

position: absolute;
width: 3358px;
height: 1894px;

background: #FFFFFF;


/* g10 */

position: absolute;
width: 332px;
height: 61px;
```

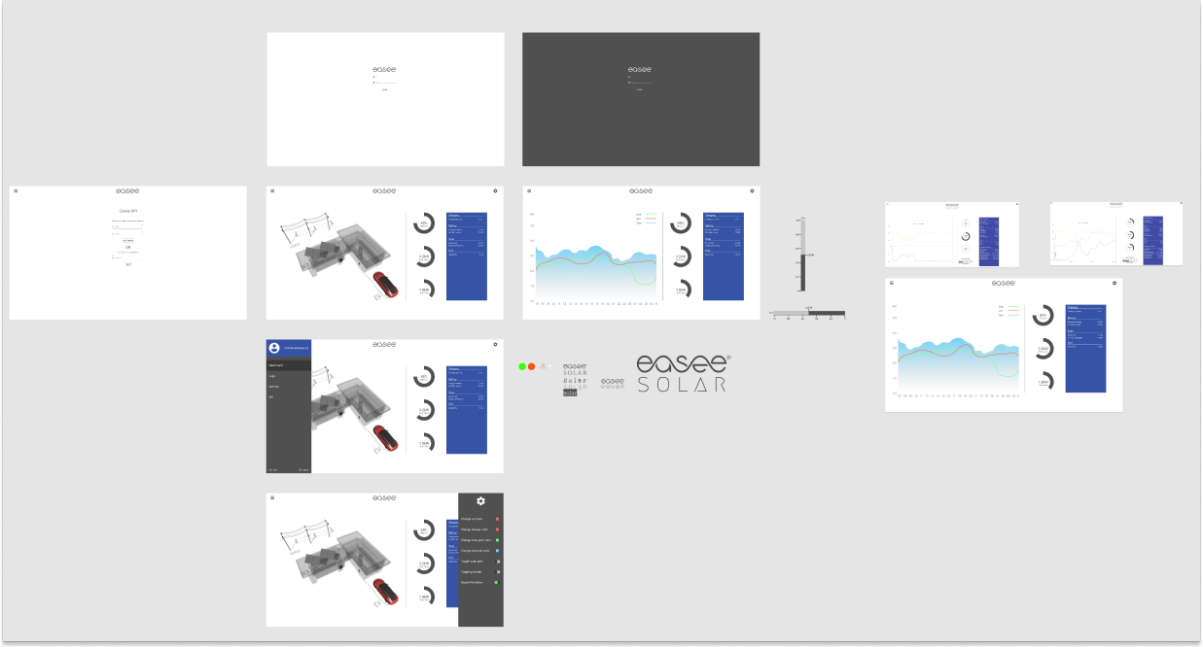
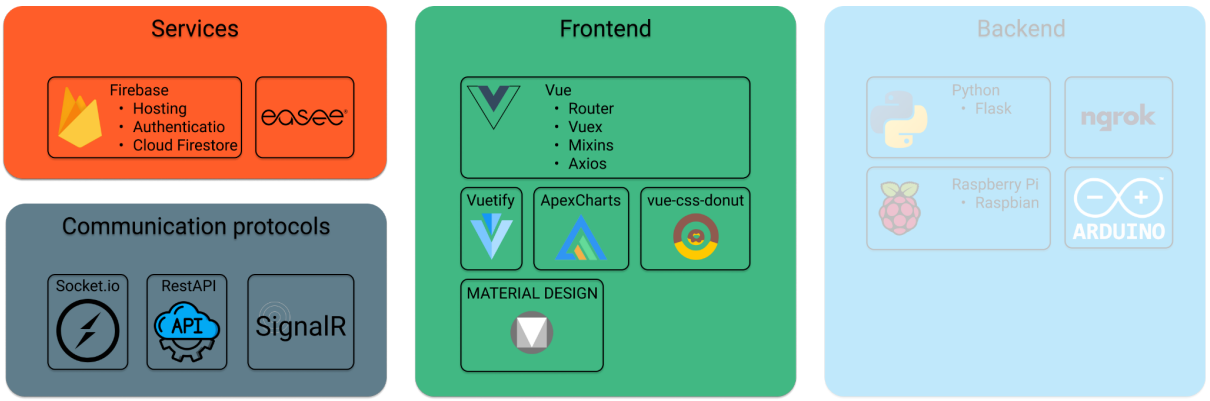


Fig. 13. Design utkast fra Figma

5.3 Frontend



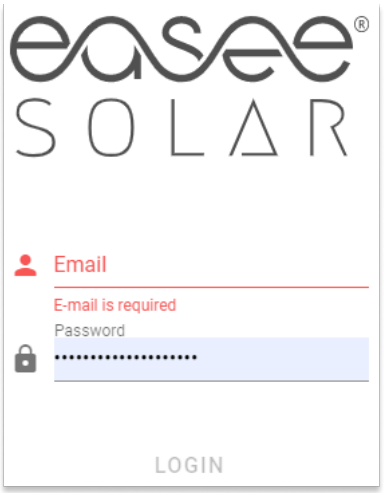
For å gi brukeren en god og brukervennlig opplevelse med applikasjonen, ble visualisering av programmet gjort via en nettportal. Årsaken til en nettbasert løsning og ikke egen OS-applikasjon var å gjøre applikasjonen allsidig for hver bruker uavhengig av arbeidsmiljøet. Selv om dette gjorde mange ting lettere å håndtere, var det andre utfordringer som måtte håndteres. Disse var tilstandshåndtering og kommunikasjon mellom de forskjellige leddene for å nevne noen. Frontend rammeverk som ble brukt i dette prosjektet er **VueJS**. Valget ble hovedsakelig gjort på grunn av tidligere erfaringer med rammeverket og enorm popularitet som ofte kan bety god utviklerstøtte i fremtiden.

Etter å ha opprettet et nytt Vue prosjekt og fjernet unødvendige deler som var automatisk generert var frontend miljøet klart oppsatt. Forskjellige visninger og komponenter ble lagt til basert på design prototype fra planleggingsdelen. Visninger ble også importert til *router*-filen, og med deres respektable stier ble ruting opprettet. Nå som det var stier å navigere til, var det behov for begrensninger som skulle forhindre uautoriserte brukere i å angi begrenset innhold.

5.3.1 Adgangskontroll

Den første visningen som blir servert til enhver ny bruker er påloggingsvisningen. På dette tidspunktet vil det ikke være mulig å opprette ny konto via web portal. Brukere som vil bli akseptert for deltakelse vil få tilgang via brukernavn og passord generert manuelt av administrator. Brukergrensesnitt for innlogging visningen er enkel, der inntastingsfeltet er utstyrt med kontrollfunksjoner som sjekker gyldighet. E-post feltet kontroller at e-post som er oppgitt av brukeren skal inneholde "@", den sjekker også for noen kjente tegn som normalt

ikke er tillatt å bruke i e-post domenenavn. Passordfeltet kontrollerer bare at minimal passord lengde ikke er mindre enn seks tegn. Seks tegn er ganske dårlige når man vurderer passord lengder, og den eneste grunnen til dette nummeret ble brukt var fordi det er den minste lengde som kreves av autentiseringstjenesten (firebase-auth). Når noen av disse reglene blir brutt, forblir Login knappen deaktivert, og tekstfeltet blir rødt. Selvfølgelig siden det er administratoren som oppretter brukerlegitimasjon, vil dette aldri være et problem. Normalt brukes slike funksjoner med innsendte funksjoner (submit forms) som sjekker data før de sender videre til backend. Uansett ble det i dette tilfellet brukt mer som brukergrensesnitt. Etter innsending sendes legitimasjon til firebase autentiseringstjeneste som sjekker om brukeren er legitim eller ikke.




```
login() {
    firebase
        .auth()
        .signInWithEmailAndPassword(this.email, this.password)
        .then(user => {
            this.$router.replace("/dashboard");
        })
        .catch(err => {
            alert("The email address or password that you've entered doesn't match any
account.");
        });
}
```

Kode utkast 8.

Hvis firebase godtar forespørsel, vil brukeren bli omdirigert til dashbordet, eller vil brukeren forbli på innloggingssiden og få beskjed om hva som gikk galt. Når brukeren blir akseptert og omdirigert til dashbordsiden, vil router guard (nevnt i 4.1.3 Routing) fortsette å sjekke om brukeren får riktig tilgang til forskjellige baner.

```
logout() {
    firebase
        .auth()
        .signOut()
        .then(() => {
            this.$router.go();
        });
}
```

Kode utkast 9.

En annen nyttig firebase funksjon er *onAuthStateChanged( )* Kode utkast 10 som kjører fra main.js filen og er "push back watcher". Med andre ord er det en tilstand som er hurtigbufret i brukerens nettleser slik at hver gang siden åpnes på nytt eller oppdateres, vil webportalen huske siste tilstand og dermed ikke omdirigere brukeren til påloggingssiden hvis ikke *logout( )* Kode utkast 9 funksjonen ble kalt.

```
firebase.auth().onAuthStateChanged(_ => {
    if (!app) {
        app = new Vue({
            router,
            vuetify,
            store,
            render: h => h(App)
        }).$mount("#app");
    }
});
```

Kode utkast 10.

5.3.2 Dashboard

Etter at adgangskontroll delen var ferdig, var det på tide å bygge brukergrensesnitt for dashbordet. Tanken var å lage en side med graf, datafelt, målere, navigasjonsfelt og alternativer. Siden navigasjonsfeltet skulle bli en del av nesten alle visninger i applikasjonen, var dette det første elementet som ble opprettet. Navigasjonsfeltet ville også gjøre det enklere å navigere mellom visninger i utviklingsprosessen.

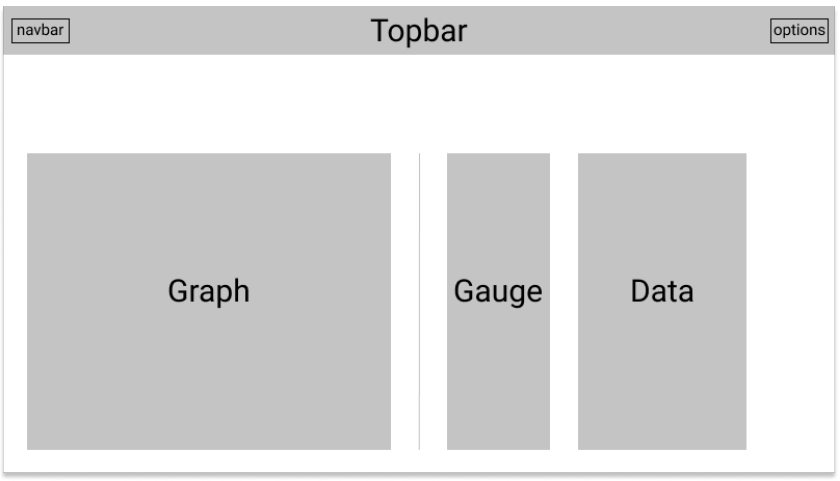


Fig. 15. Dashboard layout

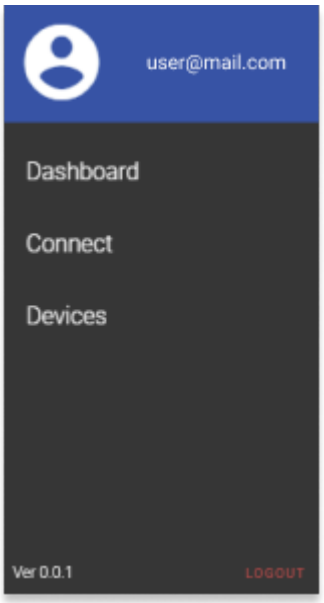
5.3.2.1 Topbar

Topbar i seg selv var ikke noe annet enn en div med tre hovedkomponenter inni. Fra venstre til høyre var det en knapp for navigasjonslinje skuff, easee logo og knapp for alternativer/verktøy.

Fig. 16. Navbar/Topbar

Navigasjonsskuff er en vuetify komponent og er en av mange komponenter som ble brukt i dette prosjektet. Det som er bra med å bruke disse komponentene, er at mye mekanismer er ferdig laget slik at utvikleren kan fokusere kun på problemløsning.

```
<v-list-item @click="goToDashboard">
    <v-list-item-title>Dashboard</v-list-item-title>
</v-list-item>

<v-list-item @click="goToConnect">
    <v-list-item-title>Connect</v-list-item-title>
</v-list-item>

<v-list-item @click="goToEaseeDevices">
    <v-list-item-title>Devices</v-list-item-title>
</v-list-item>
```




```
goToDashboard() {
    this.$router.push("/dashboard").catch(err => {});
},
goToConnect() {
    this.$router.push("/connect").catch(err => {});
},
goToEaseeDevices() {
    this.$router.push("/devices").catch(err => {});
}
```

Kode utkast 11.

Selv om det meste av estetikken og utseendet var ferdig, måtte navigasjonsfelt utformes for å passe inn i prosjektet. Knapper for forskjellige visninger ble bundet til funksjoner som ville utløse ruteren når de ble brukt.

Øverst til høyre vises informasjon om brukeren som er logget inn i applikasjonen. Denne informasjonen hentes fra firebase.

```
<span id="user-mail">{{ user.email }}</span>
```

```
data: () => ({
    user: firebase.auth().currentUser
})
```

Kode utkast 12.

Navigasjonsskuff og navigasjonsfeltet fungerer ikke bare som navigasjonsverktøy, men disse har også brukergrensesnitt funksjoner som viser brukeren når tilgangstoken til Easee kontoen er utløpt.

Fig. 17. tilkoblingsstatus

Disse visuelle effektene slettes automatisk når brukeren oppdaterer tilgangstokenet.

5.3.2.2 Graf

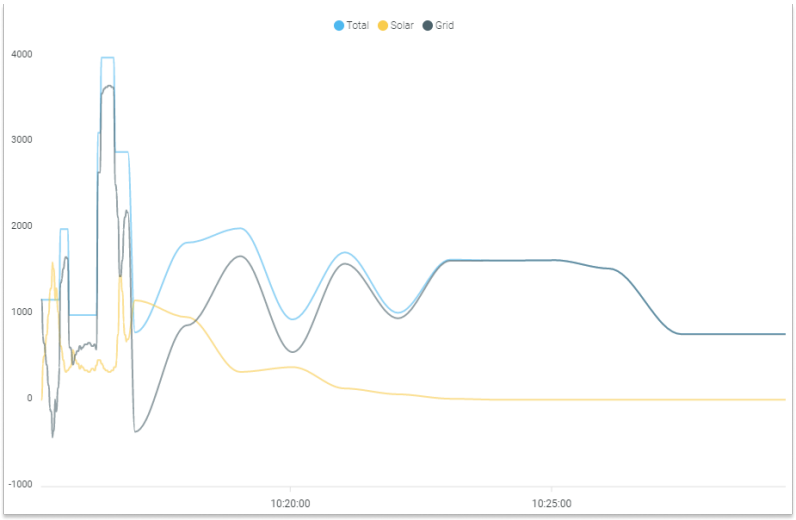


Fig. 18. Dashboard graf

Grafen gir en visuell oversikt over hele prosessen. Data som total strømforbruk, produsert solenergi og strøm importert fra nettet er vist i grafen. Når siden er lastet, begynner plotting med 1 sek intervall. Etter at datasettet når lengden på 100 datapunkter, reduseres plott intervallet til 60 sek. På den måten får resultatet mye jevnere graf, spesielt over en lengre periode.

```
mounted() {
        var self = this;
        this.interval = setInterval(() => {
            self.plotData();
        }, 1000);
    }
```

Kode utkast 13.

```
methods: {
        plotData() {
            // Garbage collector
            if (this.chartOptions.xaxis.categories.length > 500) {
                this.chartOptions.xaxis.categories.shift();
                this.series[0].data.shift();
                this.series[1].data.shift();
                this.series[2].data.shift();
            }

            if (this.$store.getters.getActivePowerImport !== 0) {
                const date = new Date();
                this.chartOptions.xaxis.categories.push(date.toISOString());
                this.series[0].data.push(
                    parseFloat(
                        this.$store.getters.getActivePowerImport * 1000
                    ).toFixed(0)
                );
                this.series[1].data.push(
                    parseFloat(this.$store.getters.getSolarValueWatts).toFixed(
                        0
                    )
                );
                this.series[2].data.push(
                    parseFloat(this.$store.getters.getPowerDifference).toFixed(
                        0
                    )
                );

                // Helping function to update chart
                window.dispatchEvent(new Event("resize"));

                // Storing data in Vuex
                this.$store.commit(
                    "updateChartTimestamp",
                    this.chartOptions.xaxis.categories
                );
                this.$store.commit("updateChartTotal", this.series[0].data);
                this.$store.commit("updateChartSolar", this.series[1].data);
                this.$store.commit("updateChartGrid", this.series[2].data);

                // Reduce update interval
                if (this.chartOptions.xaxis.categories.length > 100) {
                    clearInterval(this.interval);
```

```
                var self = this;
                this.interval = setInterval(() => {
                    self.plotData();
                }, 60000);
            }
        }
    }
}
```

Kode utkast 14.

Det er også implementert logikk som fungerer som en type søppeloppsamler (garbage collector). Det betyr at når matrise blir lenger enn 500 punkter, vil logikken slette det første elementet fra matrisen. Dette tallet kan selvfølgelig økes til mye større verdi uten at det vil gi noen ytelsesproblemer, men det gjør at grafen ser generelt jevnere ut når det ikke er for mange datapunkter.

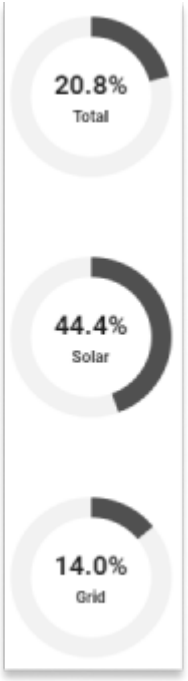
Normalt sett når brukeren navigerer til forskjellige visninger i applikasjonen, blir alt som er lagret lokalt i hver visning eller komponent, nullstilt. For å la brukeren fortsette med sin økt lagres data fra grafen inni sentrallagringsenheten, 4.1.4 Vuex. Med vue *created()* blir data lastet fra vuex i det sekundet dashbordet lastes inn.

```
created() {
    // Loading data from Vuex
    if (this.$store.getters.getChartTimestamp.length !== 0) {
        if (this.chartOptions.xaxis.categories.length === 0) {
            this.chartOptions.xaxis.categories = this.$store.getters.getChartTimestamp;
            this.series[0].data = this.$store.getters.getChartTotal;
            this.series[1].data = this.$store.getters.getChartSolar;
            this.series[2].data = this.$store.getters.getChartGrid;
        }
    }
}
```

Kode utkast 15.

Det oppstod et problem med automatisk grafoppdatering når datasettet ble oppdatert, men "resize" hendelse løste det.

5.3.2.3 Instrumentpanel



Smultringdiagram er et brukergrensesnitt som gir brukeren informasjon om verdier innenfor den øvre og nedre grensen. Diagrammet øverst (Total) viser det totale energiforbruket i en husholdning. Denne informasjonen blir hentet fra Easee equalizer gjennom en 4.2.2 Websocket tilkobling og er basert på hovedsikringsverdien og forbruket. Diagrammet i midten (Solar) representerer produsert solenergi og data hentes fra backend via en annen websocket tilkobling. Diagrammet nederst (Grid) representerer forskjellen mellom energiforbruk og produsert energi. Ved hjelp av vue computed, oppdateres diagrammer i sanntid basert på oppdateringer fra enheter.

```
computed: {
    solarDataUpdated: function() {
        return (this.solarPower[0].value =
this.$store.getters.getSolarValueWattsPersentage).toFixed(1);
    },
    totalConsumptionUpdate: function() {
            return (this.totalPower[0].value =
this.$store.getters.getTotalAvailableEffectPersentage).toFixed(1);
    },
    powerConsumedDifferenceUpdate: function() {
        if (this.$store.getters.getPowerDifferencePersentage <= 0) {
            return (this.gridPower[0].value = 0.0).toFixed(1);
        } else {
            return (this.gridPower[0].value =
this.$store.getters.getPowerDifferencePersentage).toFixed(1);
        }
    }
}
```

Kode utkast 16.

Pakken vue-css-donut-chart tar inn forskjellige egenskaper (props) for å tilpasse figurene og utseendet på diagrammene.

```
<vc-donut class="donut"
  foreground="#f2f2f2"
  :size="donutSize"
  unit="px"
  :thickness="25"
  :sections="totalPower"
  :total="100"
  :start-angle="0"
  :auto-adjust-text-size="true">
    <h1 v-bind:style="{ fontSize: dataFontSize + 'px' }">
        {{ totalConsumptionUpdate }}%</h1>
    <h4 v-bind:style="{ fontSize: legendFontSize + 'px' }">
        {{ totalPower[0].label }}</h4>
</vc-donut>
```

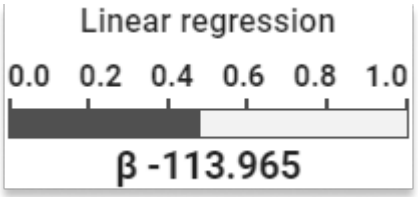
Kode utkast 17.

Selv om smultringdiagrammene fungerte etter hensikten, var måten diagrammet ble lastet inn i oppstarten lite intuitivt. De startet på null og etter at data ble hentet inn i appen begynte animasjonen å sparke inn. For å fikse denne kjedelige introen ble det lagt til logikk som startet animasjon som ruller fra null til hundre og tilbake til null igjen. Dette ligner på hvordan audi gjør i dashbord på bilene sine.

```
spinUpAnimation(start = 0, end = 100, duration = 500) {
    let startTimestamp = null;
    const step = timestamp => {
        if (!startTimestamp) startTimestamp = timestamp;
            const progress = Math.min(
              (timestamp - startTimestamp) / duration, 1);
            this.totalPower[0].value = Math.floor(
              progress * (end - start) + start);
            this.solarPower[0].value = Math.floor(
              progress * (end - start) + start);
            this.gridPower[0].value = Math.floor(
              progress * (end - start) + start);
            if (progress < 1) {
                window.requestAnimationFrame(step);
            }
            if (progress === 1) {
                this.rollBackAnimation();
            }
    };
    window.requestAnimationFrame(step);
}
```

Kode utkast 18.

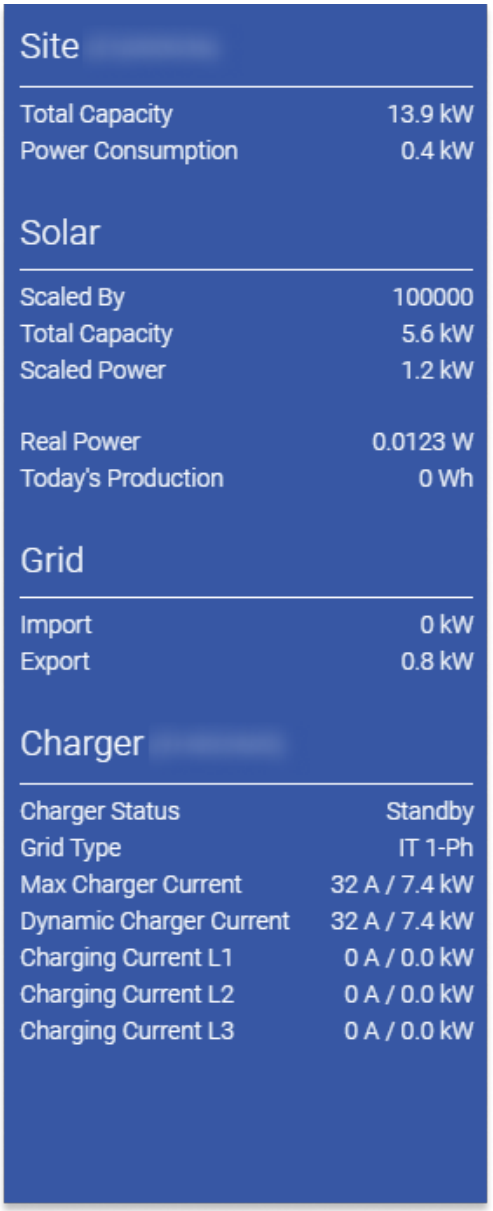
Koden for tilbakerullingsfunksjonen er nokså lik Kode utkast 18. Den eneste forskjellen er at dersom det finnes verdi i vuex, ruller animasjonen til denne verdien, og dersom det ikke er noen verdi, ruller den tilbake til null. Begge funksjonene kalles på, i vue created. Siden størrelsen på smultringer styres gjennom egenskaper (props), ble det lagt til logikk som sjekker vindusstørrelse og endrer smultringstørrelsen i henhold til logikken.



Nederst på instrumentpanelet er det et liggende stolpediagram som representerer resultatet fra lineær regresjonsmodell. Denne måleren er skreddersydd uten tredjepartspakker og reaktiviteten kommer av DOM manipulering med v-bind attributtet. Full forhåndsvisning av koden finnes inne i ScaleGraph.vue filen. Mer om lineær regresjon i 5.3.3.1 Regulering av laderen

5.3.2.4 Datatab

Data seksjonen består for det meste av enhetsindikasjoner. Data seksjonen inneholder inngående informasjon om solsystemer, for eksempel energiprodusert før og etter skalering, daglig produksjon osv. Den inkluderer også informasjon om lader og forskjellige strømmer som er satt i forskjellige faser, samt ladestatus. All denne informasjonen oppdateres i sanntid gjennom forskjellige websockets tilkoblinger. Data seksjonen er modulbasert og kan enkelt endres i programvaren.



Det meste av dataene som leveres fra laderen eller equalizer kommer ufiltrert, som betyr mange desimaler. For å presentere de nødvendige dataene på en god og intuitiv måte, måtte de fikses. I vue computed er det en rekke funksjoner som tar inn kilde data og formaterer det i et mer brukervennlig format.

Selv om data som er hentet fra laderen og equalizer gir utvikleren nok informasjon om anlegget, må fortsatt noe data beregnes ut fra gitte verdier. Eksempel på dette vil være total kapasitet. Equalizer supplerer applikasjon med data som nettype, antall faser tilkoblet og hovedsikringsverdi. Å kombinere all informasjon er nok til å kunne beregne hva som er maksimal tilgjengelig strøm for husholdningen. (Fig. 19) viser logikk som beregner total kapasitet på et anlegg.

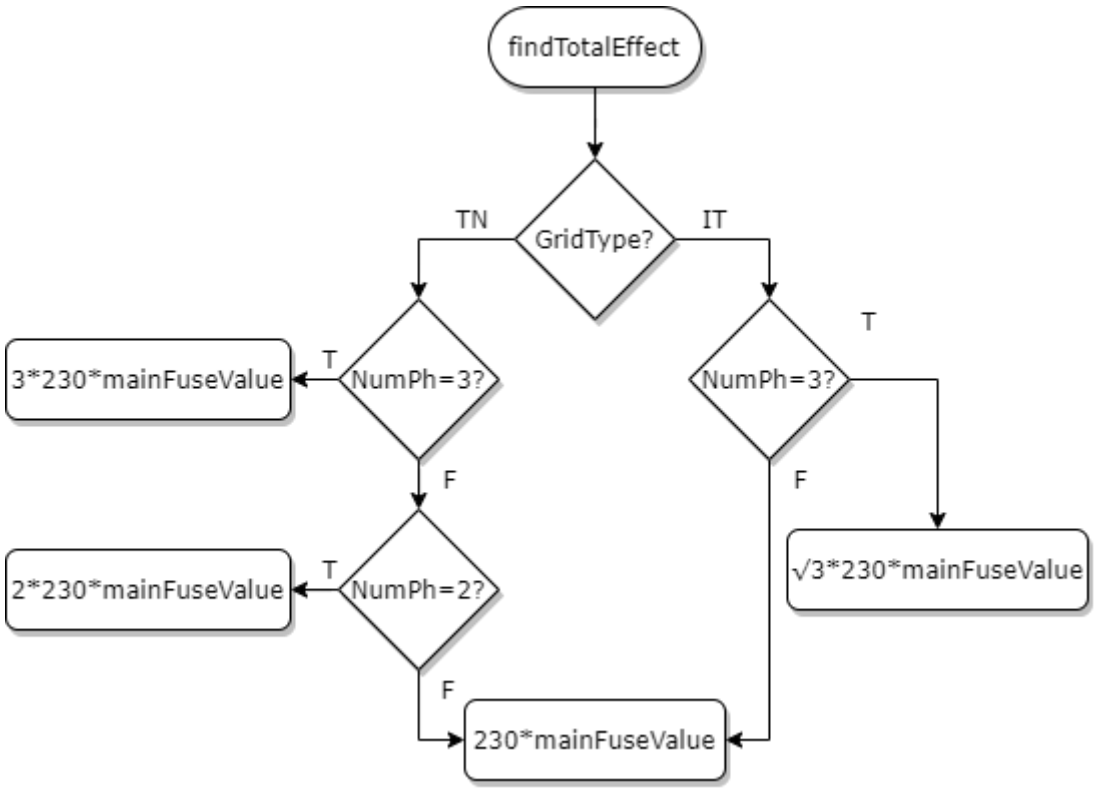


Fig. 19. Total kapasitet på et anlegg flytskjema

Som nevnt tidligere i dokumentasjonen, lagres alle innkommende data i vuex før de kan nås med getters fra andre komponenter. Data seksjon er ikke noe unntak. Fordi data fra vuex kan brukes av flere komponenter, er det lurt å holde dem i originalformat og heller formatere dem lokalt i forskjellige komponenter. Kode utkast 19 er et av eksemplene på formatering av data på en slik måte. Funksjonen sjekker først om data eksisterer, dersom data ikke eksisterer returneres funksjonen 0. Dette er for å unngå å vise brukeren "NaN" som er JavaScript egenskapen "Not-a-Number" og indikerer at verdien ikke er et lovlig nummer. Etter det sjekker logikken om verdien er innenfor det aksepterte området, dersom det ikke er det returnerer funksjonen 0. Og til slutt dersom verdien passerer alle kontrollene, blir den formatert med fire desimaler og videre til data seksjonen.


```
solarRealPower: function() {
    if (!this.$store.getters.getSolarData.realPower) {
        return 0.0;
    } else {
        if (this.$store.getters.getSolarData.realPower <= 0.00009) {
            return 0.0;
        } else {
```

```
            return parseFloat(this.$store.getters.getSolarData.realPower).toFixed(4);
        }
    }
}
```

Kode utkast 19.

5.3.3 Options

Selv om denne delen tilhører dashbordet, består den av mange viktige deler som former hele prosjektet, dermed virket det mest hensiktsmessig å dele det opp til et eget avsnitt. Ved å klikke på tannhjulikonet i hjørne øverst til høyre vises en meny med mange forskjellige alternativer og funksjoner.

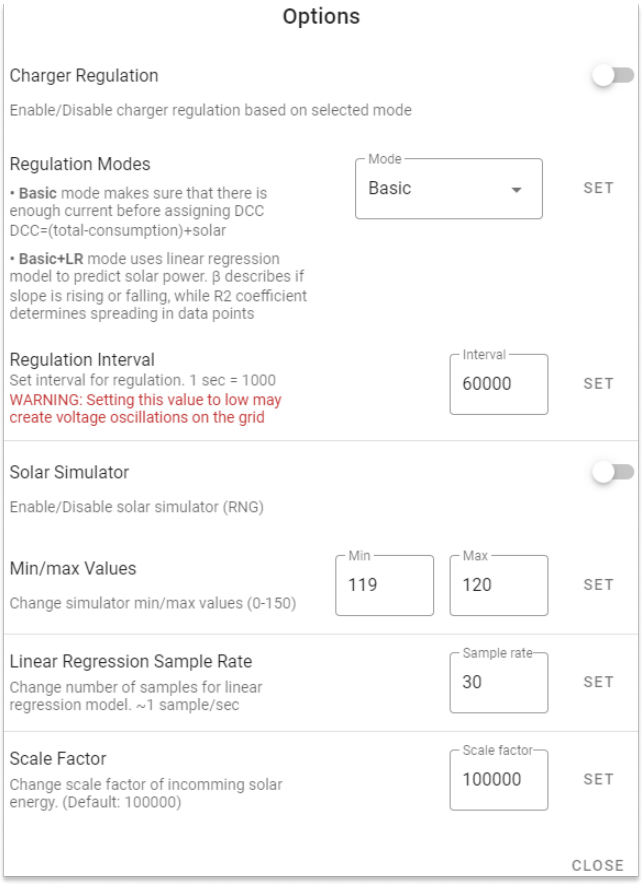


Fig. 20. Options oppsett

5.3.3.1 Regulering av laderen

Ved siden av å lage et verktøy for å utvikle og teste ut solenergi integrasjon, var laderegulering et av hovedmålene for dette prosjektet. For øyeblikket består programvaren av to reguleringsmoduser, **Basic** og **Basic + LR**. Hovedfunksjonaliteten til Basic modus er å

beregne resterende energi før den tilordnes laderen. Hvis overskuddsenergi er større enn maksimal ladestrøm (MCC), vil dynamisk ladestrøm (DCC) settes til lik maksimal ladestrøm. Fordi Easee bruker strøm (A) til å tildele laderne energi, må resultatene fra beregningene konverteres til ampere før de kan sendes videre til laderen.

$$I = \frac{P}{numPh * 230} \leftrightarrow P = numPh * 230 * I$$

```
currentToEffectConverter(current) {
      var gridType = this.$store.getters.getChargerGridType;

      // TN_3_PH
      if (gridType === "1") {
        return 3 * 230 * current;
        // TN_2_PH
      } else if (gridType === "2") {
        return 2 * 230 * current;
        // TN_1_PH OR IT_1_PH
      } else if (gridType === "3" || gridType === "5") {
        return 230 * current;
        //  IT_3_PH
      } else if (gridType === "4") {
        return 230 * Math.sqrt(3) * current;
      } else {
        return 0;
      }
    },
    effectToCurrentConverter(effect) {
      var gridType = this.$store.getters.getChargerGridType;
      // TN_3_PH
      if (gridType === "1") {
        return effect / (3 * 230);
        // TN_2_PH
      } else if (gridType === "2") {
        return effect / (2 * 230);
        // TN_1_PH OR IT_1_PH
      } else if (gridType === "3" || gridType === "5") {
        return effect / 230;
        //  IT_3_PH
      } else if (gridType === "4") {
        return effect / (230 * Math.sqrt(3));
      } else {
        return 0;
      }
    }
```

Kode utkast 20.

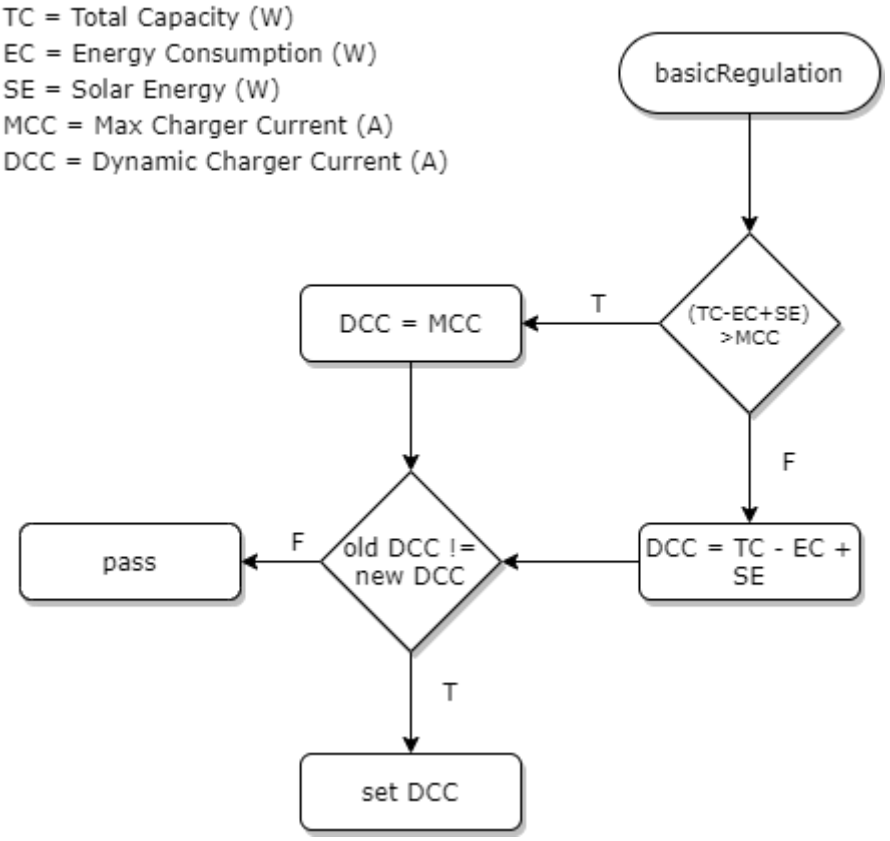
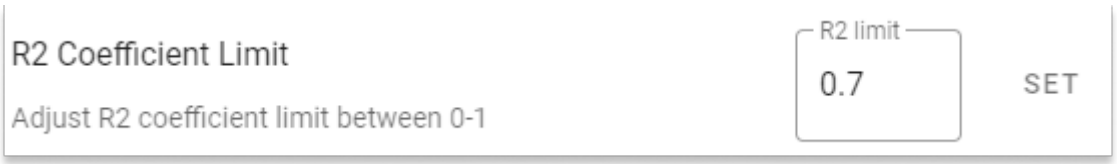


Fig. 21. Basic mode flytskjema

På grunn av måten mange elbiler fungerer, spesielt når de mottar lav ladestrøm, implementerte Easee en terskel på 6 ampere. Det betyr at når laderen mottar en verdi som er mindre enn 6 ampere, vil ladingen pauses. Det er fordi noen elbiler vil nekte oppstart av lading etter å ha mottatt for lav strømverdi, og må dermed kobles fra og til manuelt for å starte en ny ladesesjon. Hvis ladingen er midlertidig stoppet, vil bilen kunne starte ladingen når høyere strømverdier er tilgjengelig. Basic + LR gjør det samme som Basic-modus, men i tillegg prøver den å holde mest mulig ladetid. Forskjellen på Basic-modus og Basic +LR er når strømmen som er til overs er under 6A, altså DCC < 6A, sjekker programmet koeffisienten β for å avgjøre om solenergien stiger eller forfaller. Hvis trenden er stigende, sjekkes regresjonsmodellens godhet R^2 og grenseverdien til denne kan justeres i applikasjonen.



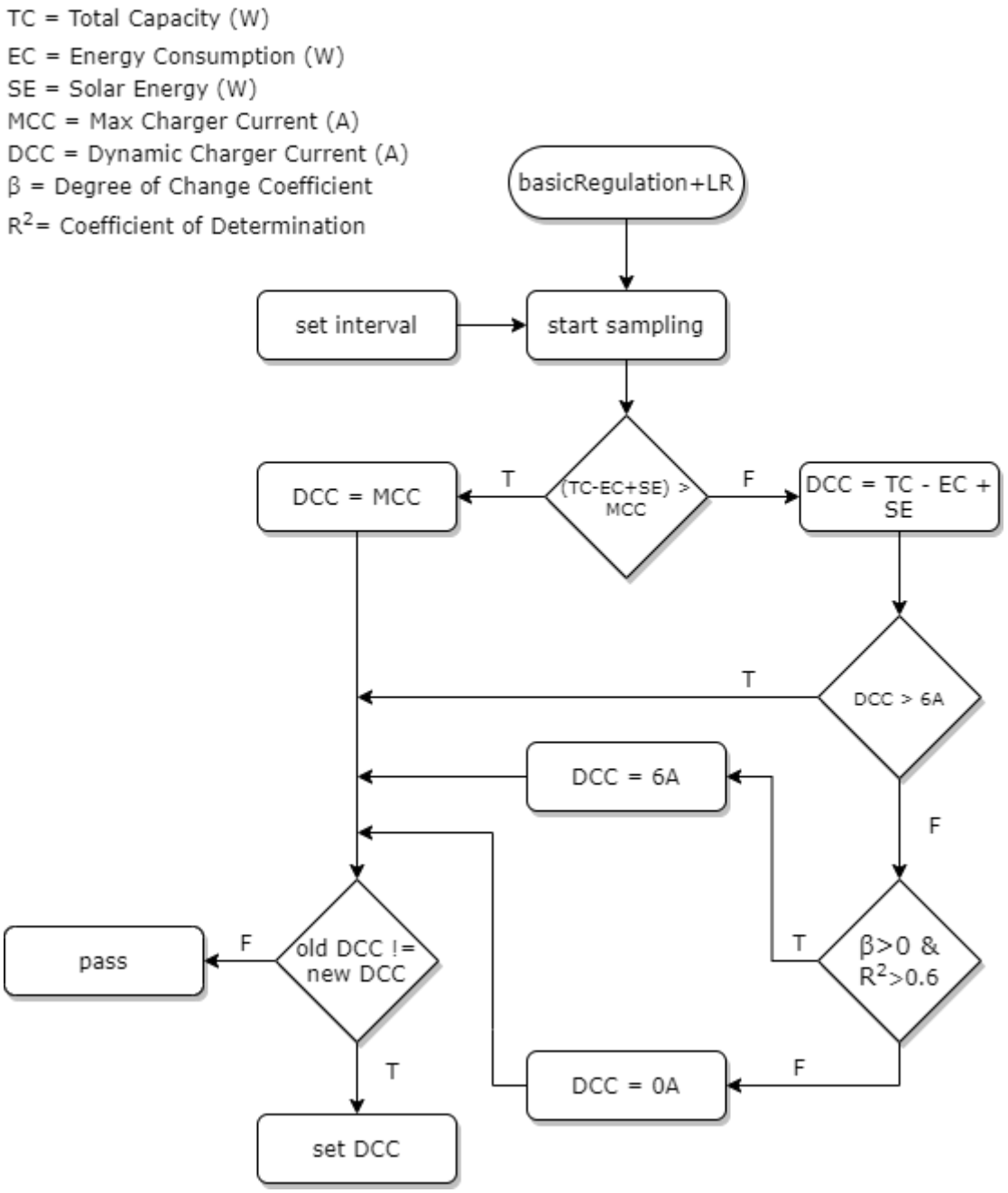
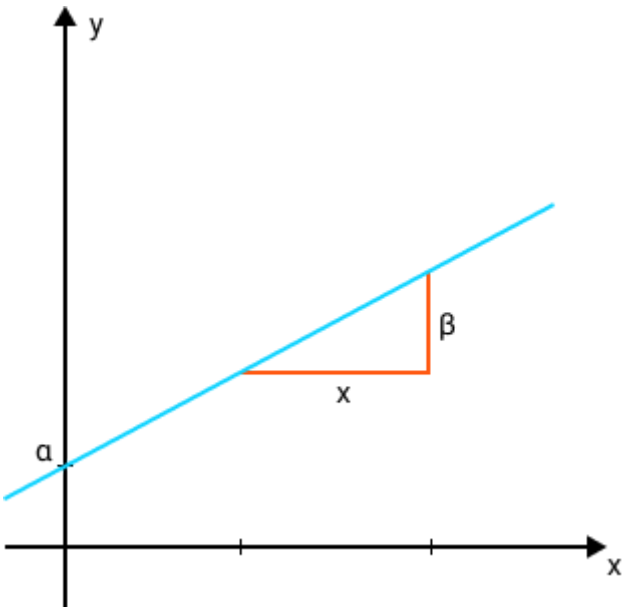


Fig. 22. Basic + LR flytskjema

For å bygge en regresjonsanalyse modell må det være to variabler for å kunne fastslå om det er en lineær sammenheng mellom disse. I dette tilfellet er x aksen tidsbasert, og tar prøver hvert sekund basert på angitt intervall, hvis f.eks intervall er satt til 30 vil x aksen se ut som dette x = [1, 2, 3, ..., 30]. Y aksen består av solenergimålinger som er i watt. Konstanten α antyder hvor linjen krysser y-aksen, mens konstanten β er linjens stigningstall. For å få en bedre forståelse av hvordan lineær regresjonsmodell er bygget opp, kan det demonstreres med et eksempel.



$$y = \alpha + \beta x$$

Basert på et reelt 9.1 Datasett er det mulig å bryte ned hver del av denne modellen, og ta utgangspunkt i følgende verdier:

$$\Sigma_{i=1}^{n}(x_i - \bar{x})^2 = 2247,5$$

$$\Sigma_{i=1}^{n}(y_i - \bar{y})^2 = 0,0000032112$$

$$\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = 0,07213$$

$$\beta = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\Sigma_{i=1}^{n}(x_i - \bar{x})^2} = 3.2111 * 10^{-5}$$

$$\alpha = \bar{y} - \beta\bar{x} = 0.01152$$

For å avgjøre om trenden stiger eller synker, er det bare behov for β. Selv om β er veldig liten, er det tydelig å se at trenden er stigende.

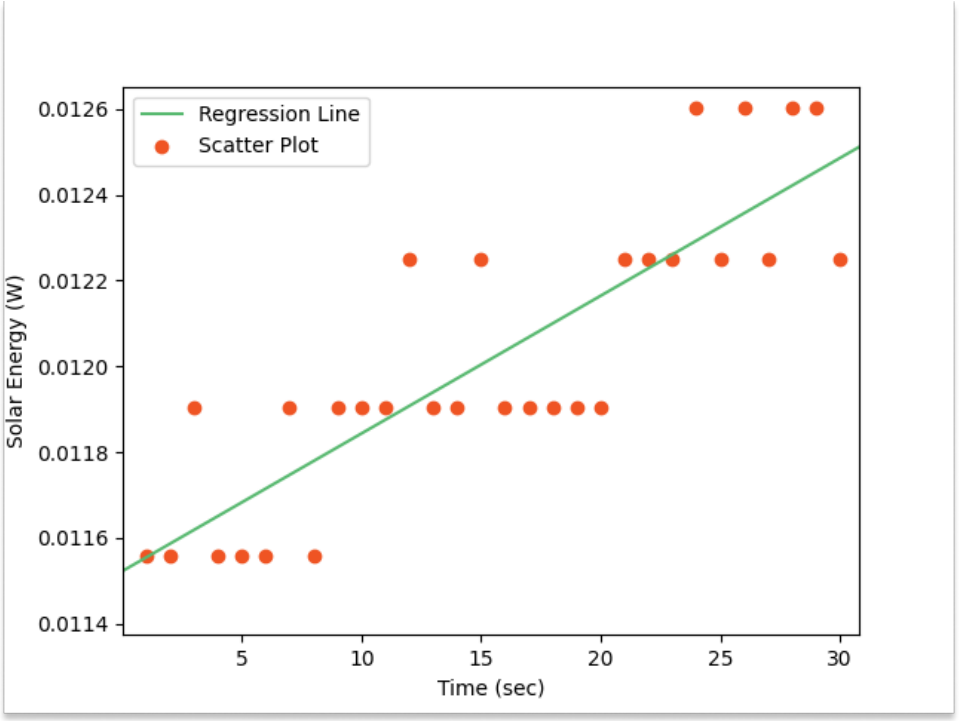


Fig. 23. 9.1 Dataset lineær regresjon output

Det som gjensto var å kontrollere troverdigheten til resultatene. Dette ble gjort ved å sjekke hvor stor variasjon det var mellom datapunktene. R^2 gir uttrykk for hvor stor del regresjonen forklarer av total variasjon. $0 \leq R^2 \leq 1$

$$r = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2} * \sqrt{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}} = 0.8490$$

$$r^2 = 0.7208$$

Normalt anses verdier for $R^2 \geq 0,75$som veldig bra, applikasjonens terskel er satt på $R^2 \geq 0,70$, og dermed tyder dette på at regresjonslinjen passer godt til datasettet. Det er også mulig å teste stigningstallet β og se om den er forskjellig fra null ved å ta en 95% konfidensintervall test.

$$\beta \pm t_{0.025} * SE(\beta) = [2.436 * 10^{-5}, 3.982 * 10^{-5}]$$

```
leastSquareRegression(y) {
      var n = this.$store.getters.getNumberOfSamples;
      var x = [];
      for (var i = 1; i < n + 1; i++) {
        x.push(i);
      }
      if (x.length === y.length) {
        var sum_x = x.reduce(function (a, b) {
          return a + b;
        }, 0);

        var sum_y = y.reduce(function (a, b) {
          return a + b;
        }, 0);

        var mean_x = sum_x / x.length;
        var mean_y = sum_y / y.length;
        var numer = 0;
        var denom = 0;

        for (var i in x) {
          numer += (x[i] - mean_x) * (y[i] - mean_y);
          denom += (x[i] - mean_x) ** 2;
        }
        var m = numer / denom;
        var c = mean_y - m * mean_x;
        var rmse = 0;
        var y_pred = 0;

        for (var i in x) {
          y_pred = c + m * x[i];
          rmse += (y[i] - y_pred) ** 2;
        }

        rmse = (rmse / n) ** 2;
        var ss_tot = 0;
        var ss_res = 0;

        for (var i in x) {
          y_pred = c + m * x[i];
          ss_tot += (y[i] - mean_y) ** 2;
          ss_res += (y[i] - y_pred) ** 2;
        }

        var r2 = 0;

        if (ss_res > 0) {
          r2 = 1 - ss_res / ss_tot;
        } else {
          r2 = 0;
        }

        var r = Math.sqrt(r2);
        var output = {
          m: m,
          c: c,
          rmse: rmse,
          r2: r2,
          r: r,
        };
        this.$store.commit("updateLinearRegressionOutput", output);
      }
    }
```

Kode utkast 21.

Reguleringsintervallfeltet bestemmer hvor ofte reguleringsalgoritmen skal kjøres. Inngangen er ikke filtrert, men merket med advarsel om at regulering som skjer for ofte kan føre til svingninger på strømnettet, og at brukeren bør være forsiktig når verdien settes for lavt.

Disse reguleringsalgoritmene er på ingen måte perfekte, men det er en god start. Herfra er det mulig å implementere andre funksjoner som f.eks å hente data fra en vær api, eller sjekke strømpriser via tjenester som Nordpol eller Tibber. Å kombinere mange forskjellige variabler vil trolig gi bedre og mer presis reguleringsmetode. Regulerings koden er inkludert 9.2 chargerController.js

5.3.3.2 Solenergisimulator

Siden det er ingen garanti for sol på vestlandet, virket det som en god idé å implementere solenergisimulator. Det er i utgangspunktet tilfeldige genererte tall (RNG) mellom brukerdefinerte øvre og nedre grenser. Selv om solenergisimulator gjør en god jobb med å simulere verdier, er det fortsatt bare tilfeldige verdier og antageligvis ikke så bra å basere en reguleringsalgoritme på.

```
randomNumberGenerator() {
    var min = this.$store.getters.getSimulationMinValue();
    var max = this.$store.getters.getSimulationMaxValue();
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

Kode utkast 22.

5.3.3.3 Skalerigsfaktor

Solcellepanel som ble brukt i dette prosjektet er omtrent like stor som en mobiltelefon, og energien som produseres er veldig liten, ca 2V * 0.2A = 0.4W under optimale forhold. For å sikre at solenergi kunne konkurrere med hverdagslig energiforbruk, trengte det å bli oppskalert, mye (x100000). Denne verdien behøver ikke å endres, men er tilgjengelig for brukerne i tilfelle feilsøking. Betaverdien fra reguleringsmetode Basic + LR blir også oppskalert.

5.3.4 Connect

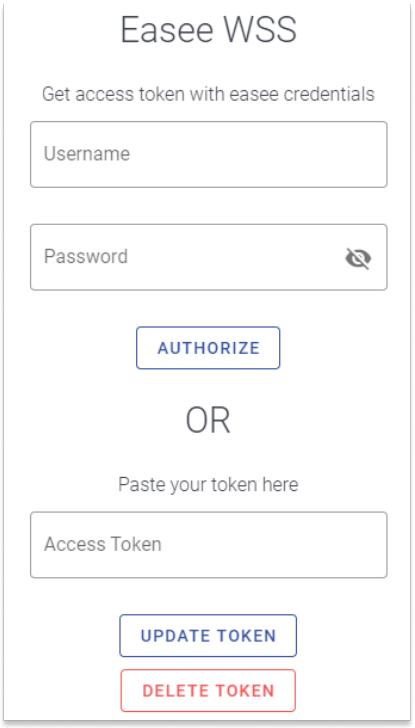


Fig. 24. Connect oppsett

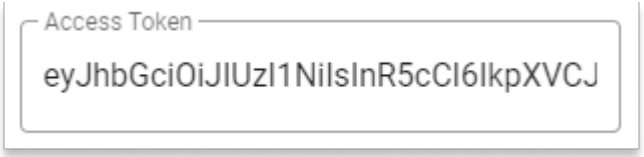
Connect siden gir brukeren tilkoblingsgrensesnitt til Easee backend ved enten å generere nytt tilgangstoken via easee.cloud brukerkontoen, eller ved å lime inn allerede generert tilgangstoken. Nylig genererte eller oppdaterte token blir lagret i brukerprofilen i firebase databasen, firestore. Siden noen brukere kan ha token med adminrettigheter, kan det være en god praksis å sltte token etter bruk for å ivareta sikkerheten. Dette kan oppnås ved å klikke på "DELETE TOKEN" knappen.

Etter å ha fått inn data fra input feltet genereres tilgangstoken ved å sende en POST forespørsel til *https://api.easee.cloud/api/accounts/token* med brukerinformasjon innpakket i JSON format. Axios er en pakke som lar programmet sende API forespørsler fra nettleseren. Hvis serveren svarer med statuskode 200, blir tilgangstoken ekstrahert fra svaret og lagret både i vuex og i brukerdatabasen. Deretter blir visuelle vedlikeholdseffekter nullstilt og signalr-tilkobling fornyes når tilgangstoken er gyldig.

```
loginWithCredentials() {
    const jsonData = JSON.stringify({
        userName: this.username,
        password: this.password
    });

    axios.post("https://api.easee.cloud/api/accounts/token", jsonData, {
        headers: {"Content-Type": "application/json"}
    }).then(resp => {
        if (resp.status === 200) {
             this.username = "";
             this.password = "";
             this.$store.commit("loadEaseeAccessToken", resp.data.accessToken);
             this.$store.commit("updateMaintenance", false);
             this.$store.commit("updateEaseeAccessTokenMaintenance", false);
             this.saveToken();
        }
    })
}
```

Kode utkast 23.

Oppdatering av tilgangstoken funksjonen har et filter som sender en GET forespørsel til Easee api for å sjekke om tokenet er gyldig eller ikke. Hvis den returnerer statuskode 200, lagres token på samme måte som generert token gjør.



Når token er lagret i databasen, trenger ikke brukeren å gjør noe før den utløper. Når brukeren besøker Connect siden, vil token som for øyeblikket brukes, være synlig i skjemaet. Skjemaene i vue som fungerer både som input felt, men også som output felt, må håndteres på en riktig måte for å fungere etter hensikten. Data objektet må ha sin egen getters og setters inne i vue computed.

```
<v-text-field v-model="accessTokenValue" label="Access Token"></v-text-field>
<v-btn class="btn" outlined @click="updateToken">Update Token</v-btn>
```

```
data: () => ({token: null}),
```

```
computed: {
    accessTokenValue: {
        get() {
            return this.$store.getters.getEaseeAccessToken;
        },
        set(value) {
            this.token = value;
        }
    }
}
```

```
methods: {
    updateToken() {
        if (this.token !== null) {
            this.$store.commit("loadEaseeAccessToken", this.token);
        }
    }
}
```

Kode utkast 24.

5.3.5 Devices

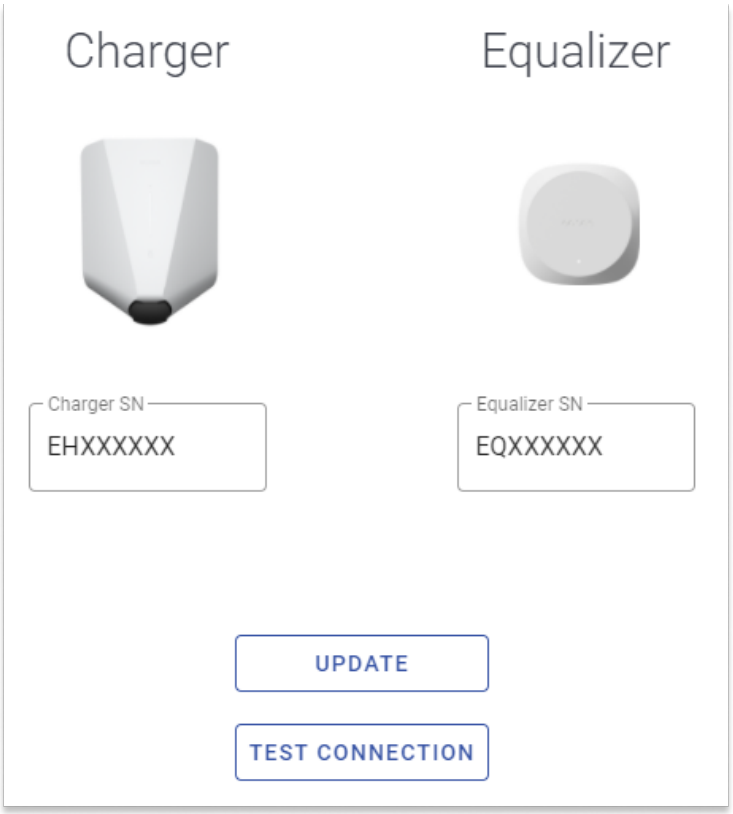


Fig. 25. Device oppsett

Forskjellige API forespørsler og signalr-datastrøm krever enhetens serienummer for å vite hvilken enhet du skal koble til. Device siden er et brukergrensesnitt som skal gjøre det visuelt bedre for brukeren å koble enhetene sine til applikasjonen. Dette prosjektet er bygget opp rundt Easee sine produkter som for tiden er Easee laderobot og equalizer.

Siden består hovedsakelig av to funksjoner, den ene oppdaterer serienumre, og den andre tester forbindelse med enhetene. Oppdateringsfunksjonen sjekker om inputfeltet er faktisk data og ikke udefinert (undefined). Deretter oppdateres dokumentfeltet i brukerdatabasen

med serienumre fra inputfeltet. Hvis brukeren er pålogget for første gang og ikke har et dokument i databasen, vil logikk opprette et nytt felt i stedet. Dette er nødvendig for å kunne håndtere "not-found" respons fra firebase.

```
if (this.equalizerSerialN !== null) {
  this.$store.commit("updateEqualizerSN", this.equalizerSerialN);
}

if (typeof this.$store.getters.getEqualizerSN != "undefined") {
  db.collection("users")
    .doc(firebase.auth().currentUser.uid)
    .update({ equalizerSN: this.$store.getters.getEqualizerSN })
    .then(() => {
      this.$store.commit("snackbarNotify", "Equalizer SN updated!");
      this.$store.commit("updateEaseeWSSConnectionStatus", false);
    })
    .catch((error) => {
      if (error.code === "not-found") {
        db.collection("users")
          .doc(firebase.auth().currentUser.uid)
          .set({ equalizerSN: this.$store.getters.getEqualizerSN })
          .then(() => {
            this.$store.commit("snackbarNotify", "Equalizer SN added!");
          })
          .catch((_) => {
            this.$store.commit(
              "snackbarNotify",
              "Unable to add Equalizer SN. Try again"
            );
          });
      } else {
        this.$store.commit(
          "snackbarNotify",
          "Unable to update Equalizer SN. Try again"
        );
      }
    });
}
```

Kode utkast 25.

“Test connection” er en funksjon som tester forbindelsen mellom klient og server ved å sende GET forespørsel til Easee api med enhetens serienummer og ber om enhetens siste tilstand. Siden Easee API ikke snakker direkte med enheter, men gjennom en MQTT-broker, vil den siste tilstanden ikke alltid bli oppdatert, men hurtigbufret. Dette er grunnen til at signalr datastrøm brukes til sanntids dataoppdatering. På samme måte som med andre API-kall, vil en grønn hake legges til enheten dersom svaret er statuskode 200, eller vil et rødt kryss være synlig hvis testen mislykkes.

Fig. 26. Tilkoblingstest utfall

```
<div class="img">
  <v-img src="@/assets/charger_white.png" alt="charger_white.png"></v-img>
  <v-icon
    v-if="testDevice && chargerTest === 'OK'"
    class="test-icon-charger"
    style="color:rgb(34, 230, 34)">
    check
  </v-icon>
  <v-icon
    v-if="testDevice && chargerTest === 'FAILED'"
    class="test-icon-charger"
    style="color:rgb(255, 30, 30)">
    close
  </v-icon>
</div>;
```

Kode utkast 26.

v-if avgjøre hvilken icon som skal brukes basert på utfallet.

```
this.testDevice = true;
// Charger connection test
axios
  .get(
    "https://api.easee.cloud/api/chargers/" +
      this.$store.getters.getChargerSN +
      "/state",
    {
      headers: {
        Authorization: "Bearer " + this.$store.getters.getEaseeAccessToken,
        "Content-Type": "application/json",
      },
    }
  ).then((resp) => {
    if (resp.status === 200) {
      if (resp.data.isOnline) {
        this.chargerTest = "OK";
      }
    }
  }).catch((error) => {
    this.chargerTest = "FAILED";
    this.$store.commit(
      "snackbarNotify",
      "Charger connection test error: " + error.response.status
    );
  });
```

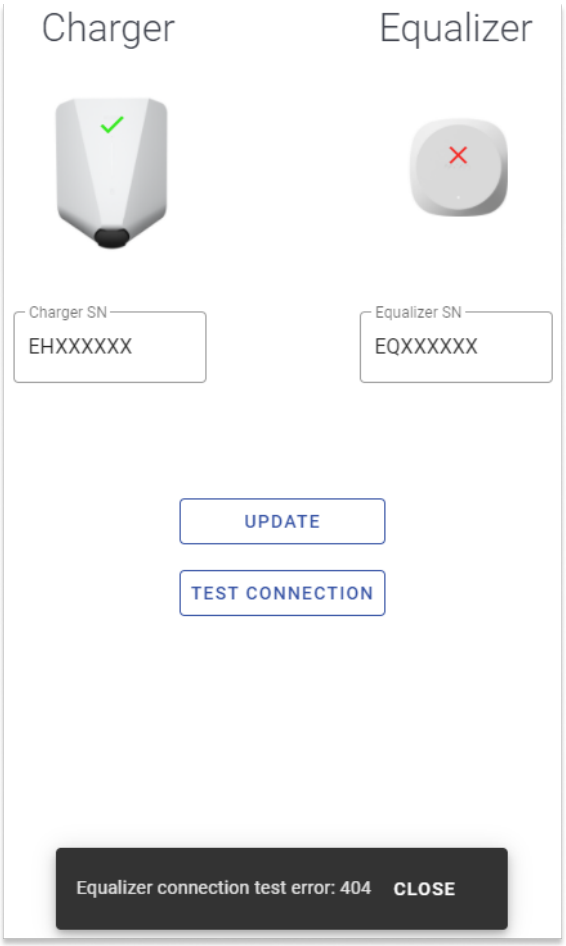
Kode utkast 27.

5.3.6 Snackbar

En nyttig vuetify komponent som er ment til visuell indikasjon og god brukeropplevelse. Ved å pakke den inn i sin egen komponent er det mulig å bruke den overalt i applikasjonen. Ulike funksjoner bruker dette for å informere brukeren om resultatet.

```
<template>
    <v-snackbar
        v-model="$store.state.snackbarActive"
        :timeout="2000"
        :bottom="'bottom'"
        >{{ $store.state.snackbarText }}
        <template>
            <v-btn
                color="white"
                text
                @click="hideSnackbar">
                Close
            </v-btn>
        </template>
    </v-snackbar>
</template>


export default {
    name: "snackbar",
    data: () => ({}),
    methods: {
        hideSnackbar() {
            this.$store.state.snackbarActive =
false;
        }
    }
};
```

Kode utkast 28.



Det er hovedsakelig to måter å implementere dette på. Den første er å bruke “promps” for å overføre forskjellige verdier som beskjed, varighet, farge osv. Den andre metoden er å bruke vuex, og ha en global tilstand for å aktivere komponenten. Disse metodene fungerer nesten på samme måte. Det som skiller vuex metoden fra “promps” er at etter den blir aktivert, henter snackbarkomponenten data fra vuex. Etter aktivering starter en timer som deaktivere snackbarkomponenten automatisk når gitt tid er utløpt.

5.3.7 Tjenester

For å lykkes med dette prosjektet ble forskjellige tjenester benyttet. Easee cloud, solcelle API og Firebase har stått sentralt gjennom hele prosjektet. Disse tjenestene bruker asynkrone long-polling tilkoblinger (websocket) for å opprette toveis sanntidskommunikasjon. RestApi blir brukt for kommandosending og datapolling.

5.3.7.1 Easee cloud

Tilkobling til easee cloud er satt opp via signalr og egen JavaScript utvidelse som er utviklet av Microsoft. Tilkoblingen starter i det siden lastes inn, deretter tar signalr klassen *HubConnectionBuilder()* inn url og tilgangstoken som parametere, og oppretter ny forbindelse. Etter at tilkoblinger er opprettet og etter en nødvendig forsinkelse, abonneres tilkoblingen på listen over gitte enheter. Vue mounted overvåker tilkoblingen, og hver gang en av enhetene sender ut nye dataoppdateringer, vil "ProductUpdate" fange opp disse og oppdatere global vuex datasettet.

```
created() {
    this.connection = new signalR.HubConnectionBuilder()
      .withUrl(this.easeeURL, {
        accessTokenFactory: () => this.$store.state.easeeAccessToken,
      })
      .build();

    this.connection
      .start()
      .then(
        // Added delay between connection.start() and .invoke()
        setTimeout(() => this.subToDevices(), 1000),
        //Disable maintenance icons when connected
        this.$store.commit("clearMaintenance")
      )
      .catch((_) => {
        this.$store.commit("updateEaseeWSSConnectionStatus", false);
        //Enable maintenance icons when not connected
        this.$store.commit("updateMaintenance", true);
        this.$store.commit("updateEaseeAccessTokenMaintenance", true);
        // Notify user
        this.$store.commit("snackbarNotify", "Update your Easee access token.");
      });
  },
  mounted() {
    this.connection.on("ProductUpdate", (update) => {
      if (update.mid === this.$store.getters.getEqualizerSN) {
        this.$store.commit("updateEqualizerData", update);
      } else if (update.mid === this.$store.getters.getChargerSN) {
        this.$store.commit("updateChargerData", update);
      }
    });
  }
```

Kode utkast 29.

Abonnering på enheter skjer i en egen funksjon som går gjennom listen over tilgjengelige enheter. Dette programmet vil normalt sett ha to enheter, Easee lader og equalizer. Navigering frem og tilbake i applikasjonen vil utløse denne logikken flere ganger, noe som betyr at abonnering også blir utløst flere ganger. For å forhindre dette ble det lagt til en kontrollør med global tilkoblingstilstand som kontrollerer om forbindelsen har blitt opprettet eller ikke.

```
subToDevices() {
      if (!this.$store.getters.getEaseeWSSConnectionStatus) {
        var devices = [
          this.$store.getters.getEqualizerSN,
          this.$store.getters.getChargerSN,
        ];
        var count = 0;
        for (var device in devices) {
          this.connection
            .invoke("SubscribeWithCurrentState", devices[device], true)
            .then((_) => {
              // Updating connection status to prevent from subscribing if already subscribed.
              console.log("Subscribing to: " + devices[count]);
              this.$store.commit("updateEaseeWSSConnectionStatus", true);
              this.$store.commit("clearMaintenance");
              count++;
            })
            .catch((_) => {
              if (this.connection.state === "Connected") {
                this.$store.commit(
                  "snackbarNotify",
                  "Failed to subscribe to Equalizer. Check connection"
                );
                this.$store.commit("updateMaintenance", true);
              }
            });
        }
      }
```

Kode utkast 30.

Dersom tilkoblingen blir opprettet på riktig måte sørger programmet for at alle visuelle indikatorer blir fjernet. Dersom det oppstår feil i tilkoblingen sørger applikasjonen for å varsle brukeren om hva som gikk galt, til en viss grad.

5.3.7.2 Firebase

Som nevnt tidligere blir Firebase brukt mye i dette prosjektet som autentisering, lagring og hostingtjeneste. Etter at brukeren har logget inn, vil brukerdata som serienumre, tilgangstoken og globale innstillinger som addressen til backend serveren lastes inn. Noe av informasjonen kan i stedet lagres i brukerens hurtigbuffer, men i dette tilfellet ble database brukt. På grunn av at denne logikken blir brukt av forskjellige sider, ble det implementert som en mixins kode utkast 31.

```
created() {
  db.collection("users").doc(firebase.auth().currentUser.uid).get()
    .then((doc) => {
      if (doc.exists) {
        this.$store.commit(
          "loadEaseeAccessToken",
          doc.data().easeeAccessToken
        );
        this.$store.commit("updateChargerSN", doc.data().chargerSN);
        this.$store.commit("updateEqualizerSN", doc.data().equalizerSN);
      }
    }).catch((_) => {
      this.$store.commit(
        "snackbarNotify",
        "Failed to load access token from database."
      );
    });
  db.collection("app").doc("settings").get()
    .then((doc) => {
      if (doc.exists) {
        this.$store.commit("loadSolarAPIURL", doc.data().ngrokURL);
      }
    }).catch((_) => {
      this.$store.commit(
        "snackbarNotify",
        "Failed to load access token from database."
      );
    });
}
```

Kode utkast 31.

I tillegg til å lese fra database, er det flere funksjoner som skriver til database. Når en tillater brukeren interaksjon med database, er det viktig å sette regler for å gjøre det sikrere. Heldigvis er dette noe som kan konfigureres på Firebase sin side. Regler for hva som skal være tillatt eller forbudt kan enkelt administreres av en administrator.

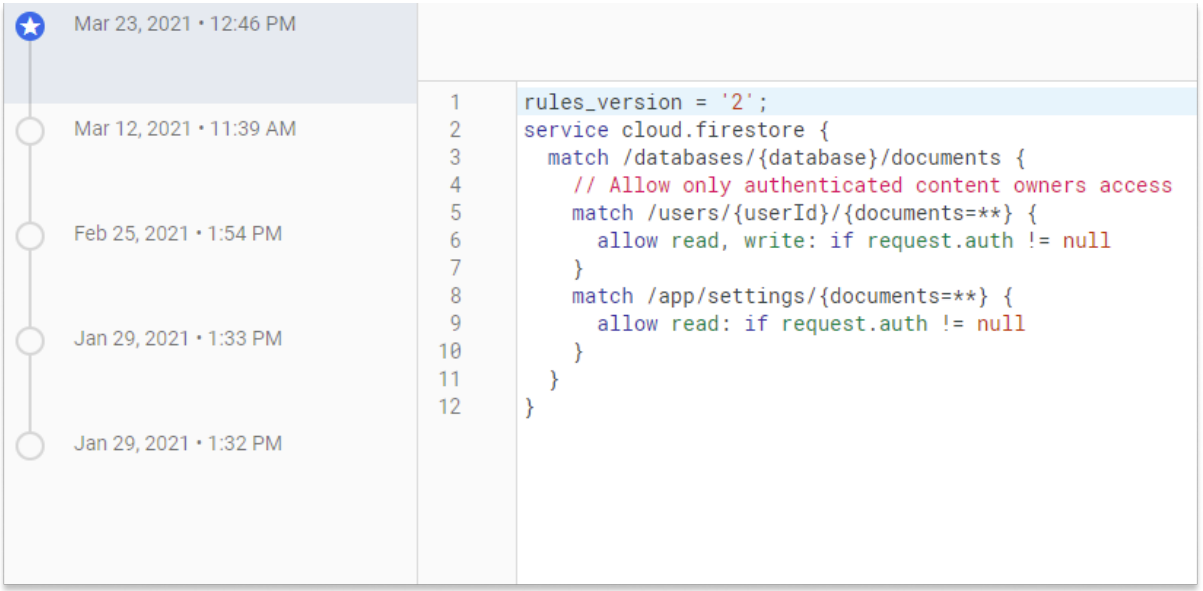


Fig. 27. Firebase firestore database tilpassede regler.

Cloud firestore som er databasetypen som brukes i dette prosjektet er dokumentorientert database (NoSQL). (Fig. 27) viser regler som ble satt for to av dokumentene som denne applikasjonen bruker. Begge krever at brukeren er autorisert gjennom autorisasjonstjenesten i firebase. Dokument */app/settings/* er skrivebeskyttet (read-only), fordi det for øyeblikket ikke er grunn til å tillate vanlige brukere skriving til det.

5.3.7.3 Solcelle API

Sol komponenten bruker også en Websocket forbindelse for å kommunisere med backend server, via socket.io pakken. I likhet med easee cloud starter forbindelsen ved sideinnlasting og deretter abonnerer på datastrøm. Denne komponenten gjør mer enn bare å motta og lagre data, det sjekker også om simulator modus er aktivert slik at data fra backend kan erstattes av RNG verdier og etterligne mottaksintervall.

```
created() {
    this.solarPanelWSS = io.connect(this.$store.getters.getSolarAPIURL);
  },
  destroyed() {
    this.solarPanelWSS.off("solarDataStream");
    this.solarPanelWSS.off("connect");
    this.$store.commit("updateSolarAPIConnection", false);
    console.log("Disconnected from solar API");
  },
  mounted() {
    this.solarPanelWSS.on("solarDataStream", (data) => {
      var jsonData = JSON.parse(data);
      if (this.$store.getters.getSimulationModeStatus) {
        var rnd = this.randomNumberGenerator();
        var simData = {
          adc: rnd,
          realPower: this.adcToWattConverter(rnd),
          dailyProd: 0,
        };
        this.$store.commit("updateSolarData", simData);
        this.regressionCalculation(simData.realPower);
      } else {
        this.$store.commit("updateSolarData", jsonData);
        this.regressionCalculation(jsonData.realPower);
      }
    });

    this.solarPanelWSS.on("connect", (_) => {
      console.log("Connected to solar API");
      this.$store.commit("updateSolarAPIConnection", true);
    });
  }
```

Kode utkast 32.

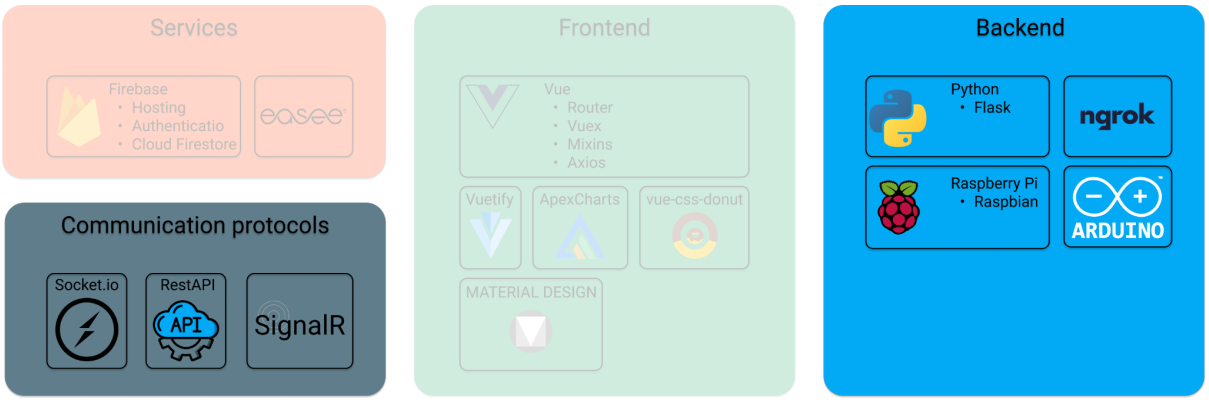
For å forhindre at flere tilkoblinger starter, bruker denne komponenten *destroyed()* for å avslutte abonnementet og lukke forbindelsen hver gang brukeren forlater dashbordsiden der

solar wss hovedsakelig brukes. I tillegg blir lineær regresjonsfunksjon kalt med data fra hver tick. Funksjonen *adcToWattConverter()* oversetter analogt signal ADC til energi (W)

```
adcToWattConverter(adc) {
    var voltage = (adc * 5.12) / 1024;
    var current = voltage / 10;
    return voltage * current;
},

regressionCalculation(energy) {
    if (this.energyData.length !== this.$store.getters.getNumberOfSamples) {
        this.energyData.push(energy);
        if (this.energyData.length > this.$store.getters.getNumberOfSamples) {
            this.energyData = [];
        }
    } else {
        this.leastSquareRegression(this.energyData);
        this.energyData = [];
    }
}
```

Kode utkast 33.

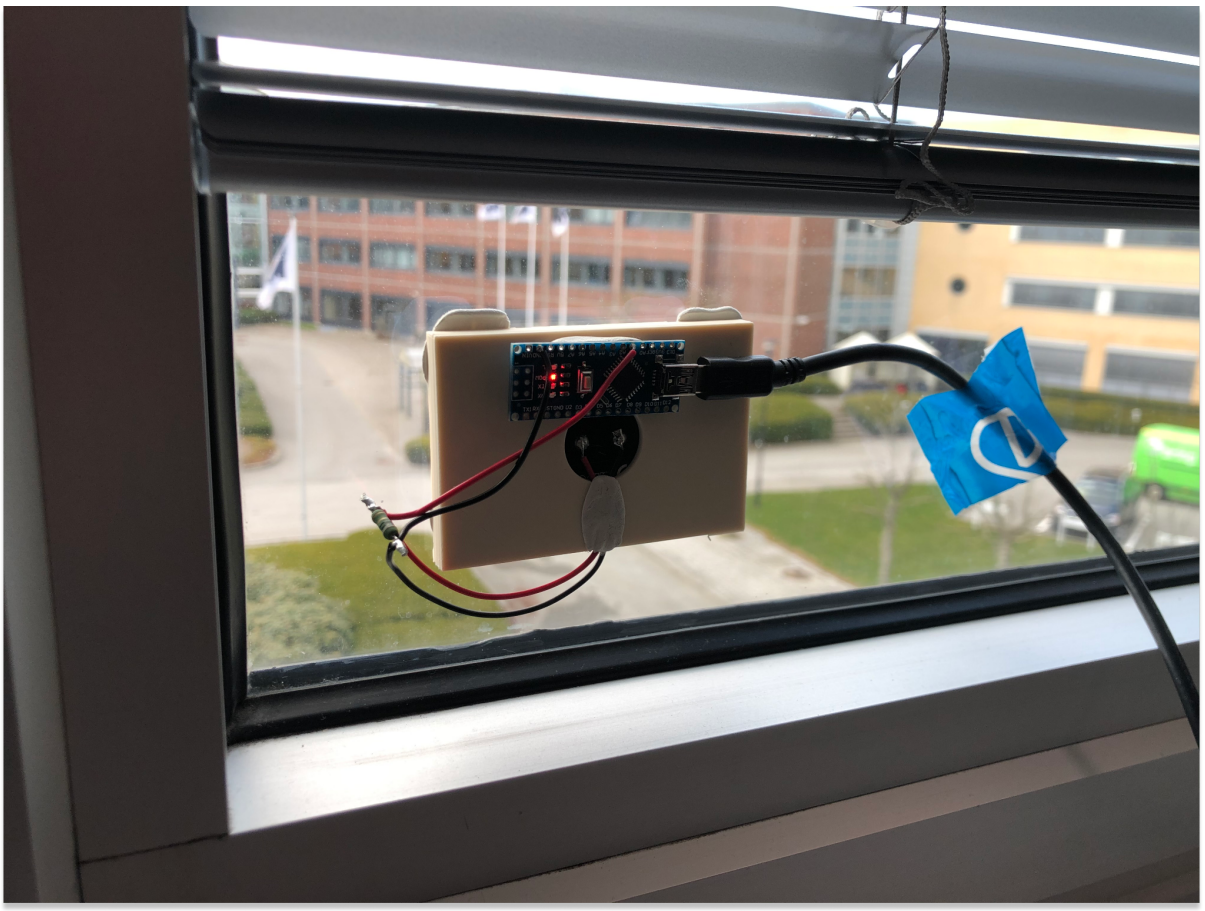
5.4 Backend



Backend server har som oppgave å lese kontinuerlig data strøm fra solcellepanel og sende det videre til webapplikasjon klientene. Serveren er konfigurert på en raspberry pi 4 og serverer sanntidsdata 24/7, basert på beskrivelsen av tjenestekvalitet (Quality of service), der målet er å ha 99,96% oppetid. Serveren bruker Raspbian OS som er basert på Debian linux med optimaliseringer for Rpi maskinvare.

5.4.1 Fjernstyring

Backend serveren er plassert nær et vindu på Easee kontorlokale. Det kan fjernstyres gjennom TeamViewer og på den måten kan serveren startes på nytt hvis noe er galt. Selv om det er mulig å skrive kode gjennom TeamViewer tilkobling, “lagger” det en del, og er derfor ikke en optimal langsiktig løsning. En måte å etablere responsiv tilkobling på er å lage tcp tunnel over port 22 og bruke den med VSCode Remote - SSH utvidelsen.



5.4.2 HTTP Tunnel

Tunneling krever som kjent en statisk offentlig ip adresse, som ikke er en enkel oppgave å få fra en internettleverandør. Heldigvis er det tjenester som tilbyr denne type funksjoner, Ngrok er en av mange leverandører og brukes i dette prosjektet. Ngrok bruker forskjellige prissystemer for å tilby sine kunder forskjellige type tjenester. Dette prosjektet er basert på en kostnadsfri plan. Ngrok tilbyr tunneling via HTTP/ TCP tilkoblinger, men en får kun lov å opprette en tunnel tjeneste om gangen dersom det skal være kostnadsfritt.
Flask-applikasjonen kjøres på localhost:5000 og med Ngrok tunneling får denne forbindelsen en url som kan brukes av offentlige klienter for å opprette en forbindelse.

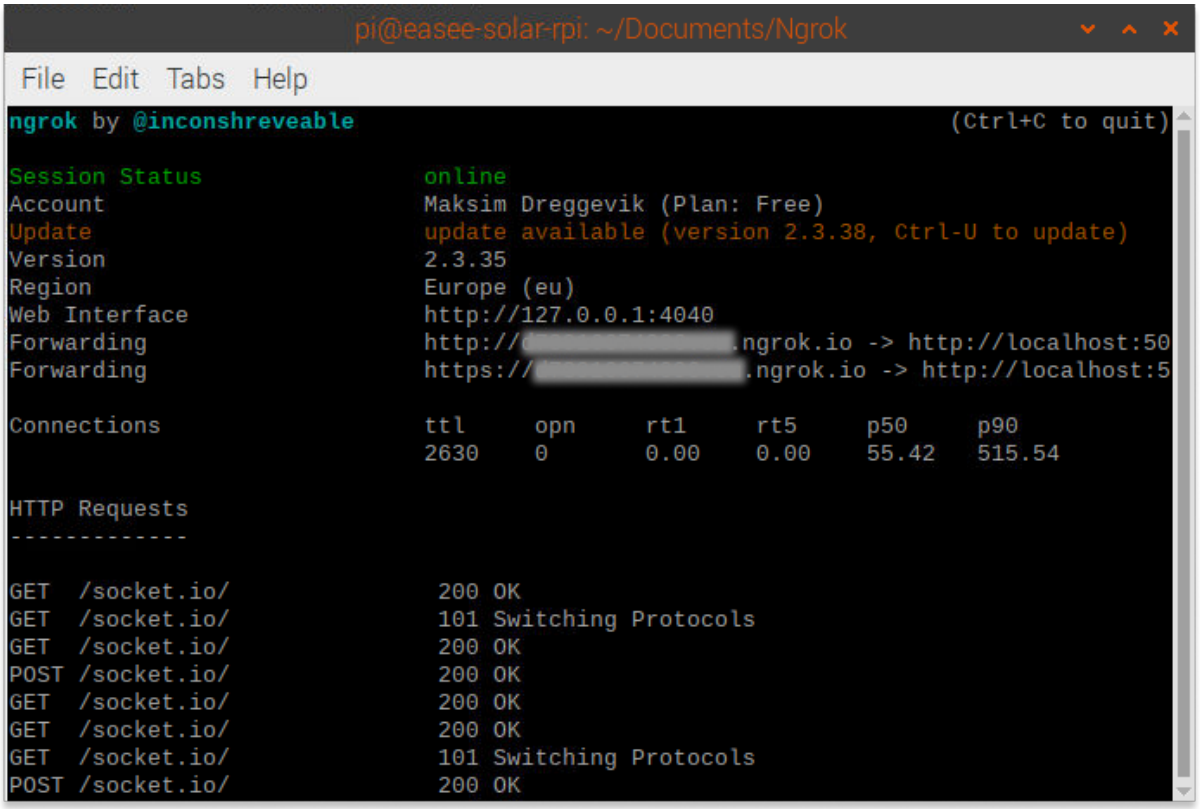


Fig. 28. Ngrok konsoll

5.4.3 Programvare

Programmet er bygget med Flask og socket.io. Den har to hovedoppgaver der den ene er å lytte på seriellport for oppdateringer fra Arduino og den andre er å beregne daglig solproduksjon. Daglig solproduksjon funksjonen har foreløpig ingen brukstilfeller, men kan potensielt bli brukt til bl.a statistikk senere. De to hovedoppgavene kjøres i egne separate tråder opprettet av socket.io funksjonen start_bakgrunn_oppgave () som bruker gevent coroutines.

```
@socketio.on('connect')
def start_get_data_thread():
    global socketio_thread, daily_prod_thread
    with thread_lock:
        if socketio_thread is None:
            socketio_thread = socketio.start_background_task(target=emit_solar_data)
    with thread_lock:
        if daily_prod_thread is None:
            daily_prod_thread = socketio.start_background_task(target=daily_production_logger)
```

Kode utkast 34.

Begge trådene starter når klienten blir koblet til. Dette er egentlig ikke den beste måten å gjøre det på, fordi den daglige produksjonsfunksjonen ikke begynner å logge før dette skjer. Rent praktisk spiller det ikke så stor rolle, fordi serveren startes på nytt av en administrator, og når admin tester klientsiden, vil serveren starte og kjøre for "alltid". Funksjonen *emit_solar_data ()* kjøres i en endeløs løkke, som leser data fra Arduino og sjekker om data er ødelagt. Dersom alt går bra, blir alt pakket inn i en json fil og sendt ut. Det er en liten men nødvendig forsinkelse på enden av sløyfen, som forhindre at pakker kolliderer på vei ut, og reduserer antall feil til et minimum.

```
def emit_solar_data():
    global solar_data
    while True:
        try:
            serial_data = serial.Serial('/dev/ttyUSB0',115200).readline().decode('utf-8',
errors='ignore')
            if serial_data != None:
                solar_data = float(serial_data)
                solar_data_real_power = adc_to_watts_converter(float(serial_data))
                data = {
                    "adc": solar_data,
                    "realPower": solar_data_real_power,
                    "dailyProd": solar_daily_prod,
                }
                socketio.emit('solarDataStream', json.dumps(data), broadcast=True)
                socketio.sleep(0.1)
        except:
            print("Failed to receive or send data")
```

Kode utkast 35.

Et av datapunktene som går inn i json objektet er daglig solproduksjon som får sin verdi fra en global variabel. Som nevnt tidligere, starter programmet to tråder der den ene har som oppgave å kalkulere daglig solproduksjon.

```
def daily_production_logger():
    global solar_daily_prod, prod_buffer, hour_data
    minute_timer = time.time()
    current_date = time.localtime().tm_mday
    while True:
```

```
if time.time() - minute_timer > 59:
    minute_timer = time.time()
    hour_data.append(adc_to_watts_converter(solar_data))
    solar_daily_prod = (sum(hour_data)*len(hour_data)/60)+prod_buffer
    if len(hour_data) > 59:
        prod_buffer += solar_daily_prod
        hour_data = []
if time.localtime().tm_mday != current_date:
    current_date = time.localtime().tm_mday
    minute_timer = time.time()
    hour_data = []
    solar_daily_prod = 0
    prod_buffer = 0

socketio.sleep(0.1)
```

Kode utkast 36.

Formelen for å finne produsert energi per time er:

$$Energy = Power * Time$$

For bedre brukeropplevelse oppdateres daglig produksjon hvert minutt i stedet for hver time, noe som fører til mer logikk og buffere som mellomlagrer all ekstra data.

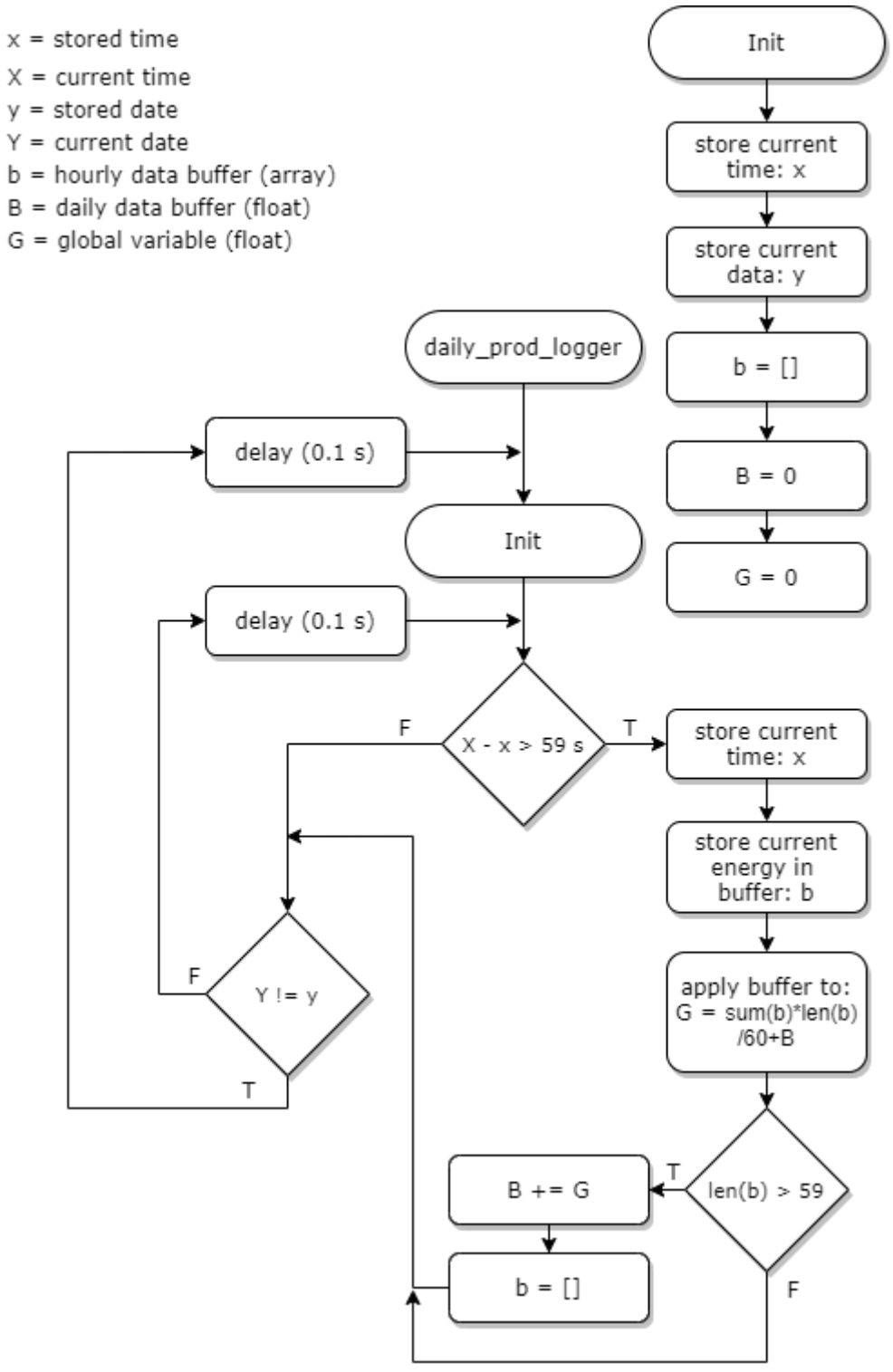


Fig. 29. Solenergi produksjonslogger flytskjema

Programmet inneholder også funksjonen *adc_to_watt_converter()* som konverterer ADC signaler om til effekt(W) produsert av solcellen.

```
def adc_to_watts_converter(adc):
    voltage = (adc*5.12)/1024
    current = voltage/10
    return voltage*current
```

Kode utkast 37.

5.4.4 Maskinvare

Arduino nano ble brukt som omformer, siden Raspberry pi ikke har ADC. Kommunikasjon mellom Rpi og Arduino er kablet med USB A - USB mini B med referansespenning på 5V. Etter noen målinger ble ref.spenningen justert til 5.12V i programmet. Dette for å få mest mulig nøyaktige målinger. Motstand på 10-ohm ble tilsatt slik at det kunne være lettere å beregne effekten. ADC på Arduino har 10-bits oppløsning, som betyr at den kan detektere $1024(10^{2})$ forskjellige analoge nivåer.

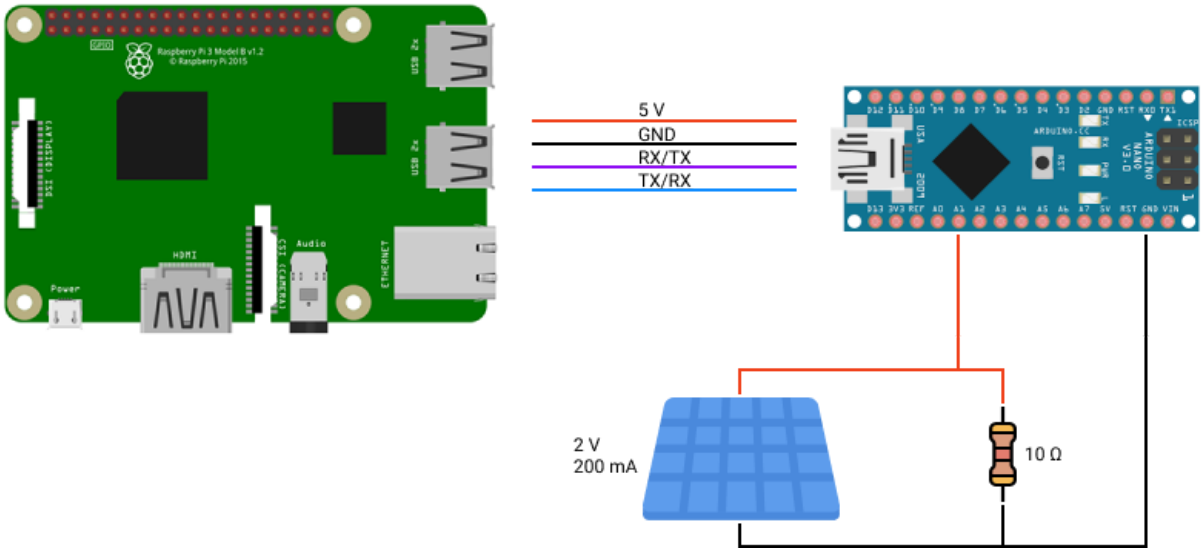


Fig. 30. Koblingsskjema for maskinvaren

Med 2V nominell spenning, vil ADC avlesningen være rundt ~400 ved perfekte forhold.

$$ADC = \frac{Vin * 1024}{Vref} = \frac{2 * 1024}{5.12} = 400$$

Solcellepanelet ble plassert i et vindu som gir en enorm reduksjon pga. feil vinkel og uv filteret som de fleste vinduer har. Målet var aldri å ha en perfekt posisjon for solcellepanelet, men heller finne ut hva som er maks kapasitet på nåværende posisjon. Etter utallige målinger med solfylte forhold var ADC verdien aldri over 150, noe som ble utgangspunktet for 100%.

$$Vin = \frac{150*5,12}{1024} = 0,75V$$

$$I = \frac{0,75V}{10\Omega} = 0,075A$$

$$P = 0,75V * 0,075A = 0,05625W$$

Dermed blir referansen 0,05625 W for 100% kapasitet, og med oppskalering på klientsiden vil dette tilsvare 5625 W (5.3.3.3 Skalerigsfaktor).

6. Konklusjon

Prosjektet ble vellykket og de fleste mål ble oppnådd. Det er alltid rom for forbedringer, spesielt for reguleringsalgoritmen, mer om dette i 6.1 Veien videre. De fleste av oppgavene gikk som planlagt, noen endringer ble gjort underveis som for eksempel frontend design og backend arkitektur. Opprinnelig var backend planlagt hostet hos Heroku og rapportering av soldata skulle skje ved hjelp av en mikrokontroller. Denne løsningen ville på sikt vært dyrere og mindre fleksibel særlig når Heroku tar betalt basert på dataoverføringsmengden.

Fra begynnelsen var det også planlagt å bruke store solcellepanel med en “on-grid” inverter fra Fronius. Disse blir levert med innebygd api som ville gjøre det mulig å lage datastrømmer på samme måte som solcellepanel som brukes i prosjektet. Forskjellen ville naturligvis være at med store nok solcellepanel trenger ikke produsert kraft å oppskaleres. Denne typen installasjon ville ha medført mye mer kostnader og planlegging, og fordi Easee på denne tiden flyttet kontorlokaler, ble den mindre og enklere løsningen foretrukket. Når det med tiden blir nødvendig vil migrasjon til større solsystem ikke være noe problem.

Vue og Flask fungerer bra sammen, men det er usikkert hvor godt dette vil skalere. For små og mellomstore prosjekter ser det ut til å fungere meget bra. Selv om TeamViewer er helt ok til administrering av backend serveren, er det fortsatt rom for forbedringer spesielt når det gjelder fjernutvikling. Applikasjonen ble delt internt med Easee ansatte for testformål, som viste seg å være positivt fordi en del bugs ble avdekket og fikset.

6.1 Veien videre

Selv om applikasjonen fungerer etter hensikten, er det noen andre forbedringer / funksjoner som kan gjøre den enda bedre. Som nevnt på slutten av kapittel 5.3.3.1 Regulering av laderen, er det mulig å forbedre reguleringsalgoritmer ved å koble applikasjonen opp mot Nordpol eller Tibber API for å få live strømpriser. Hensikten med dette er å vite når man skal bruke strøm fra nettet og hvis det er mulig, når man bare skal bruke egenprodusert energi.

En annen faktor som kan være lurt å sjekke er værvarsling fra en eller annen vær API og basert på resultatene tilføre regresjonsmodellen noen ekstra faktorer som forhåpentligvis vil bidra til å øke nøyaktigheten til modellen.

For tiden er systemet sterkt avhengig av strømmåler, equalizer og målt forbruk. Implementering av virtuell equalizer vil gi brukerne en måte å simulere strømforbruk, samt endre total effektkapasitet som vil gi enda større frihet til å simulere forskjellige scenarier.

Med økt kompleksitet, kan disse funksjonene potensielt gi mer fleksibilitet og smarthet til hele systemet.

6.2 Ekstra

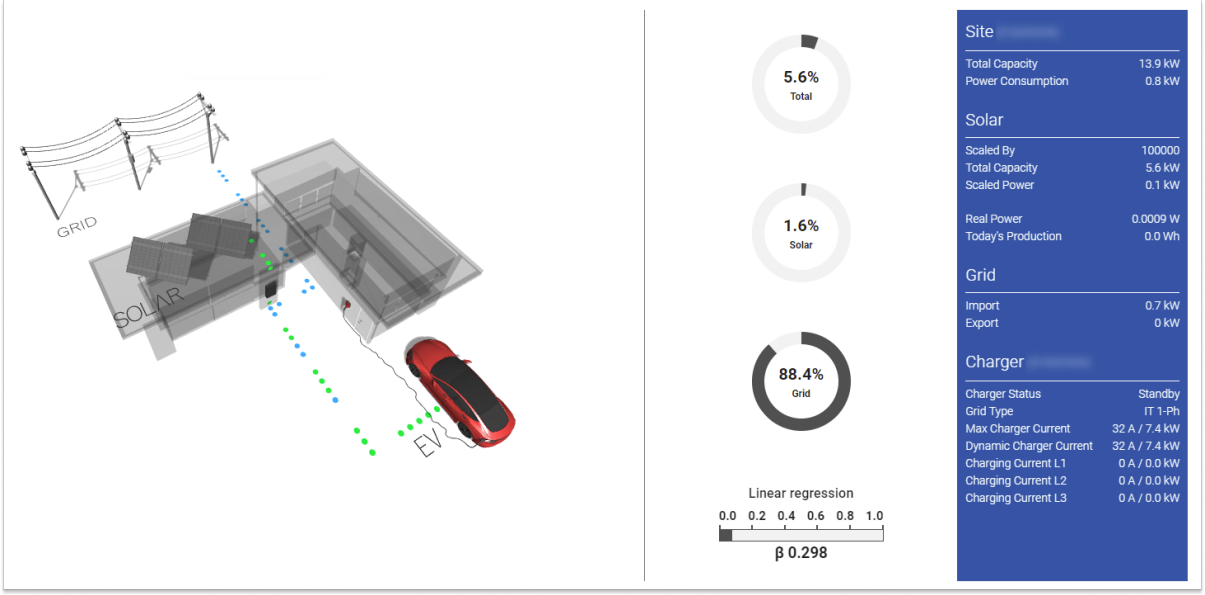


Fig. 31. Animasjonsvisning

I tillegg til arbeid som dette dokumentet allerede viser, var det fra starten av en plan om å integrere dashbordanimasjon. Dette ville ha vært et alternativ der brukeren kunne velge mellom graf- eller animasjonsvisning. Animasjonen ville ha representert anlegget som skulle endres dynamisk når kraft produseres og forbrukes. Noen av gjenstandene ble laget i Blender, mens andre ble lastet ned fra internett. Animasjon ble laget med Unity og har en del API'er som lar frontend trigge de forskjellige tilstandene. Det er for eksempel API for å bytte farge på bilen og laderen, trigger pulsering av prikkstien med en viss hastighet og kan endre fargen på disse prikkene.

Programmeringsspråk som ble brukt i Unity er C#, Kode utkast 38 viser logikken som trigger prikkstien (grønne) fra solcellen og videre til bilen. Hver prikk er en grafisk komponent som

kalles for “sprite”, i koden blir disse aktivert og deaktivert i en viss sekvens med en forsinkelse mellom.

```
using System.Collections;
using UnityEngine;

// Trigger Sequence by calling ToggleSolarSequence(float speed)
// In WebGL should be triggered on event/data update

public class SolarSequence : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            ToggleSolarSequence(0.03f);
        }

        if (Input.GetKeyDown(KeyCode.Alpha0))
        {
            ToggleSolarPath();

        }
    }

    void ToggleSolarPath()
    {
        foreach (SpriteRenderer sprite in GetComponentsInChildren<SpriteRenderer>())
        {
            sprite.enabled = !sprite.enabled;
        }
    }

    void ToggleSolarSequence(float speed = 0.03f)
    {
        StartCoroutine(CounterCoroutine(speed));
    }

    private IEnumerator CounterCoroutine(float timeDelay)
    {
        SpriteRenderer[] sprite = GetComponentsInChildren<SpriteRenderer>();
        int pathSize = GetComponentsInChildren<SpriteRenderer>().Length;
        int index = 0;
        for (int i = 0; i < pathSize / 3; i++)
        {
            if (index < 3)
            {
                sprite[index].enabled = true;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 1].enabled = true;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 2].enabled = true;
                yield return new WaitForSeconds(timeDelay);
            }
            if (index < 27)
            {
                sprite[index + 3].enabled = true;
                sprite[index].enabled = false;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 4].enabled = true;
                sprite[index + 1].enabled = false;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 5].enabled = true;
                sprite[index + 2].enabled = false;
                yield return new WaitForSeconds(timeDelay);
            }
```

```
            if (index > 29)
            {
                sprite[index-3].enabled = false;
                sprite[index].enabled = true;
                yield return new WaitForSeconds(timeDelay);
                sprite[index - 2].enabled = false;
                sprite[index + 1].enabled = true;
                yield return new WaitForSeconds(timeDelay);
                sprite[index - 1].enabled = false;
                sprite[index + 2].enabled = true;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 3].enabled = true;
                yield return new WaitForSeconds(timeDelay);

                sprite[index].enabled = false;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 1].enabled = false;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 2].enabled = false;
                yield return new WaitForSeconds(timeDelay);
                sprite[index + 3].enabled = false;
                yield return new WaitForSeconds(timeDelay);
            }
            index += 3;
        }
    }
}
```

Kode utkast 38.

Etter at animasjon ble konvertert til WebGL og integrert i applikasjonen, brukte den desverre så mye prosessorkraft at nettleseren ble ubrukelig treg. Bytting tilbake til grafen hjalp ikke med ytelsesproblemer. Den eneste løsningen var å oppdatere nettleseren som var i strid med hele single page konseptet. En av årsakene til ytelsesproblemet kan være at animasjon er en virkelig 3D-visning selv om den ser ut som 2D. Skyggen rundt gjenstandene er også dynamisk og ville ha endret seg med kameravinkelen. (Fig. 32) viser forskjellen på ressursbruket med og uten animasjonen.

Fig. 32. CPU bruk med og uten animasjon

Fordi dette var en funksjon som ikke hadde noen praktisk betydning, var det ikke tid til å finne ut hvordan man kunne forbedre dette, men heller forlate funksjonen inntil videre. Faktisk er animasjonen fortsatt inkludert i frontend, men den er deaktivert.

7. Referanser

[1] **JavaScript** (u. d) JavaScript, Language of the Web
Hentet: 25.03.21 Link: https://www.javascript.com/

[2] **VueJS** (u. d) The Progressive Javascript Framework
Hentet: 25.03.21 Link: https://vuejs.org/

[3] **Vue Router**(u. d) Component Based Router
Hentet: 25.03.21 Link: https://router.vuejs.org/

[4] **Vuex** (u. d) State Management Pattern
Hentet: 25.03.21 Link: https://vuex.vuejs.org/

[5] **Firebase** (u. d) Developer tool
Hentet: 30.03.21 Link: https://firebase.google.com/

[6] **Python** (u. d) Object-oriented programming language
Hentet: 30.03.21 Link: https://www.python.org/

[7] **Flask** (u. d) Python Web Development Framework
Hentet: 30.03.21 Link: https://flask.palletsprojects.com/en/1.1.x/

[8] **Ngrok** (u. d) Secure Tunnels for Local Services
Hentet: 30.03.21 Link: https://ngrok.com/

[9] **MDN Web Docs** (u. d) An overview of HTTP
Hentet: 03.04.21 Link: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[10] **Sookicheff, K.** (2019, 4. april) How Do Websockets Work?
Hentet: 30.03.21 Link: https://sookocheff.com/post/networking/how-do-websockets-work/

[11] **Lewington, G.** (u. d) Socketio
Hentet: 30.03.21 Link: https://ably.com/topic/socketio

[12] **Fletcher, P.** (2014, 10. juni) Introduction to SignalR
Hentet: 03.04.21 Link:
https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

[13] **Grinberg, M.** (2013, 20. mai) Designing a RESTful API with Python and Flask
Hentet: 03.04.21 Link:
https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

[14] **Lemmon, M.** (2009, 1. februar) What is a serial interface?
Hentet: 07.04.21 Link:
https://www3.nd.edu/~lemmon/courses/ee224/web-manual/web-manual/lab9/node4.html

[15] **Vuetify** (u. d) Material Design Framework
Hentet: 30.03.21 Link: https://vuetifyjs.com/en/

[16] **Dumptyd** (u. d) Lightweight Vue component for drawing pure CSS donut charts
Hentet: 07.04.21 Link: https://dumptyd.github.io/vue-css-donut-chart/

[17] **FusionCharts** (u. d) Modern & Interactive Open-source Charts
Hentet: 07.04.21 Link: https://apexcharts.com/

[18] **Atlassian** (u. d) Collaboration planning tool
Hentet: 23.04.21 Link: https://trello.com/nb

[19] **Figma, Inc.** (u. d) Web-Design tool
Hentet: 23.04.21 Link: https://www.figma.com

[20] **Karaman, I.** (2020, 11. august) Step counter code
Hentet: 08.04.21 Link:
https://stackoverflow.com/questions/63366884/problem-multiples-counter-number-same-page

[21] **Løvås, G. G.** (2018) Statistikk for Universiteter og Høgskoler (4. utg.).
Oslo: Universitetsforlaget

8. Figurer & Modeller

Smashicons (u. d) Api
Hentet: 29.03.21 Link:
https://www.flaticon.com/free-icon/api_2164832?term=api&page=1&position=35&page=1&position=35&related_id=2164832&origin=search

Smashicons (u. d) Server
Hentet: 02.04.21 Link:
https://www.flaticon.com/free-icon/server_742282?term=server&page=1&position=26&page=1&position=26&related_id=742282&origin=search

Freepik (u. d) Laptop
Hentet: 02.04.21 Link:
https://www.flaticon.com/free-icon/laptop_413837?term=laptop&page=1&position=61&page=1&position=61&related_id=413837&origin=search

surang (u. d) Resistor
Hentet: 18.04.21 Link:
https://www.flaticon.com/free-icon/resistor_2399651?term=resistor&page=1&position=30&page=1&position=30&related_id=2399651&origin=tag

J5 (u. d) Arduino nano && Rpi
Hentet: 18.04.21 Link: http://johnny-five.io/platform-support/#arduino-nano

edson (u. d) Glass House Project
Hentet: 02.04.21 Link:
https://free3d.com/3d-model/glass-house-project-1733.html?__cf_chl_jschl_tk__=121f0c151450ddc4e96590f9e44fb95b279145af-1620932158-0-AUzAaCHBAm8uKaIb3GWSJavCWnM_LBACFS4MBRqn0N_c_NDBJJ_XwRFwZUl4wGn8kmqQW-RJO1YpsKIfv4-rt4l5VlzxSLY-7c2_EEqQVNT4wf5ZWSwshpjyN4M6KGT6f19kAMnE8TZlBH-9XXO6PXp0sSAx9YeJrX1bhj7rE1n9TO-XOdMu-yUowIVHhW7xDqelDNoYeu9n5L7ryqqgiHxzPLxPEqzdlDnI7fjyZgT_vGo8yiv4-oNhYj-6P8BxE6mQSlRCSQYFamF2hczBa4M4PN1JaiEvhyLYcOgf6e1fJmnQgARwVGmKiNaiDt1RMbN_kwOEk7mSvsEsIgYid_D0cvZ4pHoaxTVoRACbrTPcKWe-wP4g6L55ppsSgU8-F2-j2yCD-_-HXHMSg0YAhKYrxWwMP0J3fIaR3GOOlph-VdlBHHr_aj1KAGInfwVZXA

CGTRICKS (u. d) Tesla Model-S
Hentet: 02.04.21 Link:
https://cgtricks.com/download-free-3d-tesla-model-s-blender-cc-studio/

printable_models (u. d) Solar Panels
Hentet: 02.04.21 Link:
https://free3d.com/3d-model/solar-panels-v1--376601.html

9. Vedlegg

9.1 Datasett

x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(y - \bar{y})^2$	$(x - \bar{x})(y - \bar{y})$
1	0,01156	−14,5000	−0,00046	210,25	0,0000002116	0,00667
2	0,01156	−13,5000	−0,00046	182,25	0,0000002116	0,00621
3	0,01190	−12,5000	−0,00012	156,25	0,0000000144	0,0015
4	0,01156	−11,5000	−0,00046	132,25	0,0000002116	0,00529
5	0,01156	−10,5000	−0,00046	110,25	0,0000002116	0,00483
6	0,01156	−9,5000	−0,00046	90,25	0,0000002116	0,00437
7	0,01190	−8,5000	−0,00012	72,25	0,0000000144	0,00102
8	0,01156	−7,5000	−0,00046	56,25	0,0000002116	0,00345
9	0,01190	−6,5000	−0,00012	42,25	0,0000000144	0,00078
10	0,01190	−5,5000	−0,00012	30,25	0,0000000144	0,00066
11	0,01190	−4,5000	−0,00012	20,25	0,0000000144	0,00054
12	0,01225	−3,5000	0,00023	12,25	0,0000000529	−0,000805
13	0,01190	−2,5000	−0,00012	6,25	0,0000000144	0,0003
14	0,01190	−1,5000	−0,00012	2,25	0,0000000144	0,00018
15	0,01225	−0,5000	0,00023	0,25	0,0000000529	−0,000115
16	0,01190	0,5000	−0,00012	0,25	0,0000000144	−0,00006
17	0,01190	1,5000	−0,00012	2,25	0,0000000144	−0,00018
18	0,01190	2,5000	−0,00012	6,25	0,0000000144	−0,0003
19	0,01190	3,5000	−0,00012	12,25	0,0000000144	−0,00042
20	0,01190	4,5000	−0,00012	20,25	0,0000000144	−0,00054
21	0,01225	5,5000	0,00023	30,25	0,0000000529	0,001265
22	0,01225	6,5000	0,00023	42,25	0,0000000529	0,001495
23	0,01225	7,5000	0,00023	56,25	0,0000000529	0,001725
24	0,01260	8,5000	0,00058	72,25	0,0000003364	0,00493
25	0,01225	9,5000	0,00023	90,25	0,0000000529	0,002185
26	0,01260	10,5000	0,00058	110,25	0,0000003364	0,00609
27	0,01225	11,5000	0,00023	132,25	0,0000000529	0,002645
28	0,01260	12,5000	0,00058	156,25	0,0000003364	0,00725
29	0,01260	13,5000	0,00058	182,25	0,0000003364	0,00783
30	0,01225	14,5000	0,00023	210,25	0,0000000529	0,003335

9.2 chargerController.js

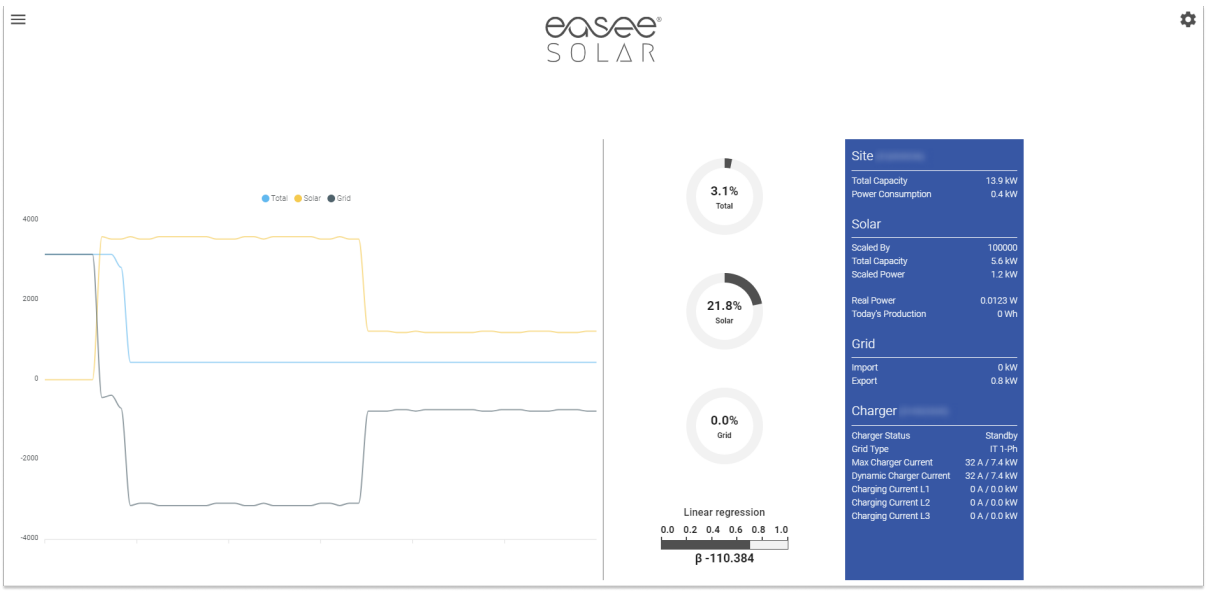
```
import axios from "axios";
import gridCalc from "@/mixins/gridCalculator.js";

export default {
  mixins: [gridCalc],
  data: () => ({}),
  methods: {
    startChargerRegulation() {
      var self = this;
      if (this.$store.getters.getRegulationModeState) {
        this.$store.commit("updateRegulationModeState");
        clearInterval(this.$store.getters.getRegulationLoopCache);
      } else {
        this.$store.commit("updateRegulationModeState");
        this.$store.commit(
          "updateRegulationLoopCache",
          setInterval(() => {
            self.basicRegulationMode();
          }, self.$store.getters.getRegulationInterval)
        );
      }
    },
    changeRegulationInterval() {
      clearInterval(this.$store.getters.getRegulationLoopCache);
      var self = this;
      if (this.$store.getters.getRegulationModeState) {
        this.$store.commit(
          "updateRegulationLoopCache",
          setInterval(() => {
            self.basicRegulationMode();
          }, self.$store.getters.getRegulationInterval)
        );
      }
    },
    basicRegulationMode() {
      var totalEnergyAvailable = parseInt(
        this.$store.getters.getTotalAvailableEffect
      );
      var consumedEnergy = parseInt(
        this.$store.getters.getActivePowerImport * 1000
      );
      var solarEnergyProdused = parseInt(
        this.$store.getters.getSolarValueWatts
      );
      var chargerMaxEffect = this.currentToEffectConverter(
        this.$store.getters.getChargerMaxCurrent
      );
      var chargerMaxCurrent = this.$store.getters.getChargerMaxCurrent;

      var newDCC = 0;

      if (
        totalEnergyAvailable - consumedEnergy + solarEnergyProdused >
        chargerMaxEffect
      ) {
        newDCC = parseInt(chargerMaxCurrent);
      } else {
        newDCC = parseInt(
          this.effectToCurrentConverter(
            totalEnergyAvailable - consumedEnergy + solarEnergyProdused
          )
        );
        if (this.$store.getters.getCurrentRegulationMode === "Basic+LR") {
          if (newDCC < 6) {
            newDCC += this.linearRegressionCheck(newDCC);
          }
```

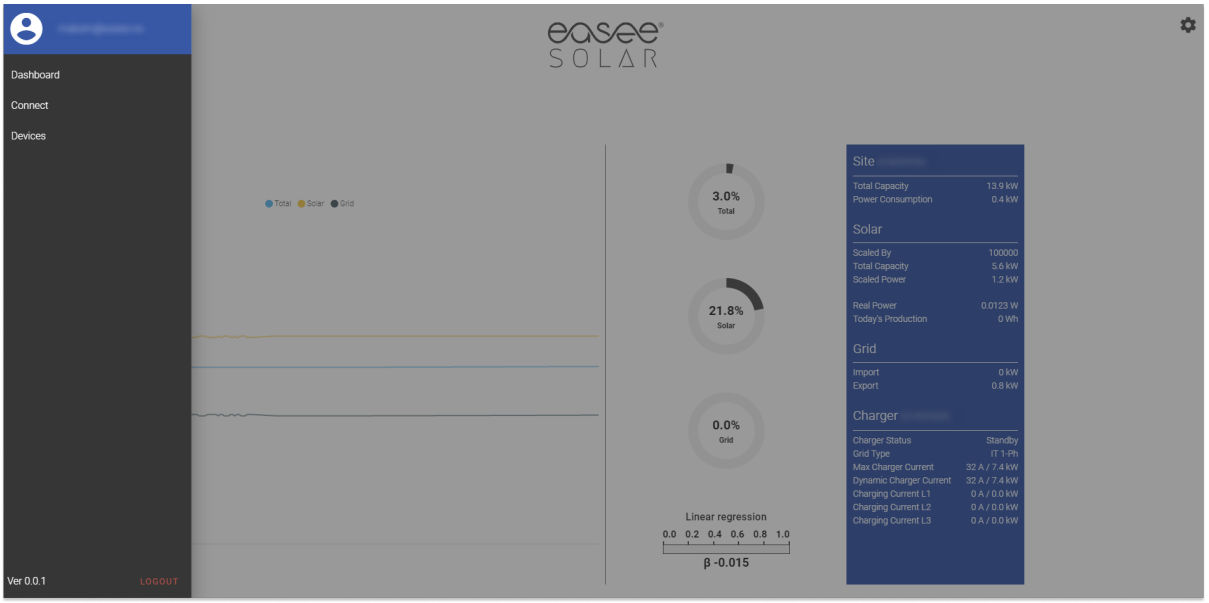
```
        }
      }

      newDCC = parseInt(newDCC);

      if (this.$store.getters.getDynamicChargerCurrentBuffer !== newDCC) {
        this.setDynamicChargerCurrent(newDCC);
        this.$store.commit("updateDynamicChargerCurrentBuffer", newDCC);
      }
    },
    linearRegressionCheck(dcc) {
      var B = this.$store.getters.getLinearRegressionOutput.m;
      var R2 = this.$store.getters.getLinearRegressionOutput.r2;
      var diff = 6 - dcc;

      if (B > 0) {
        if (R2 > this.$store.getters.getR2CoefficientLimit) {
          return diff;
        } else {
          return 0;
        }
      } else {
        return 0;
      }
    },
    setDynamicChargerCurrent(currentVal) {
      const data = JSON.stringify({
        dynamicChargerCurrent: parseInt(currentVal),
      });

      axios
        .post(
          "https://api.easee.cloud/api/chargers/" +
            this.$store.getters.getChargerSN +
            "/settings",
          data,
          {
            headers: {
              Authorization:
                "Bearer " + this.$store.getters.getEaseeAccessToken,
              "Content-Type": "application/json",
            },
          }
        )
        .then((resp) => {
          if (resp.status === 202) {
            // Can't send same value twice
            console.log("NEW DCC IS SET");
          }
        })
        .catch((error) => {
          console.log("API FAILED" + error);
        });
    },
  },
};
```

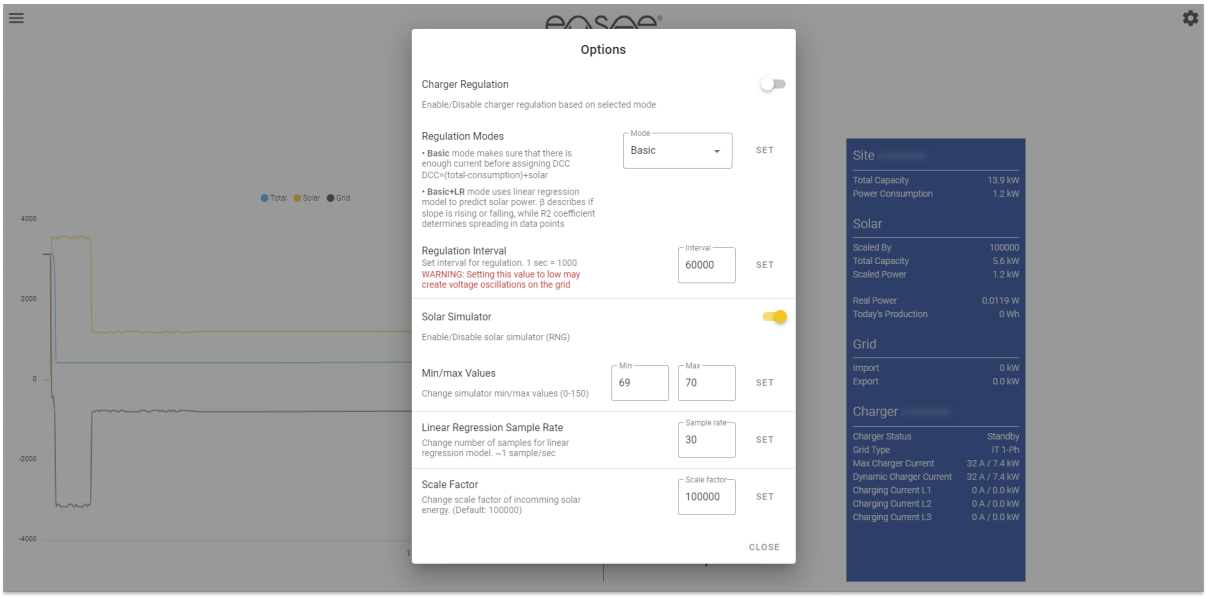
9.3 Github

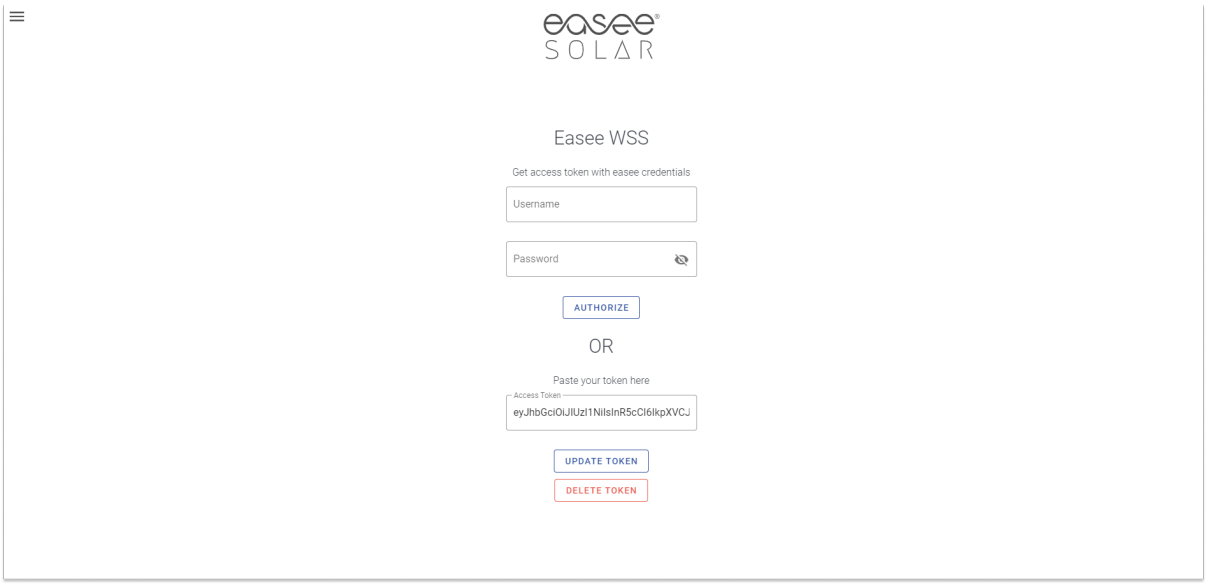
https://github.com/Herant/Easee-Solar

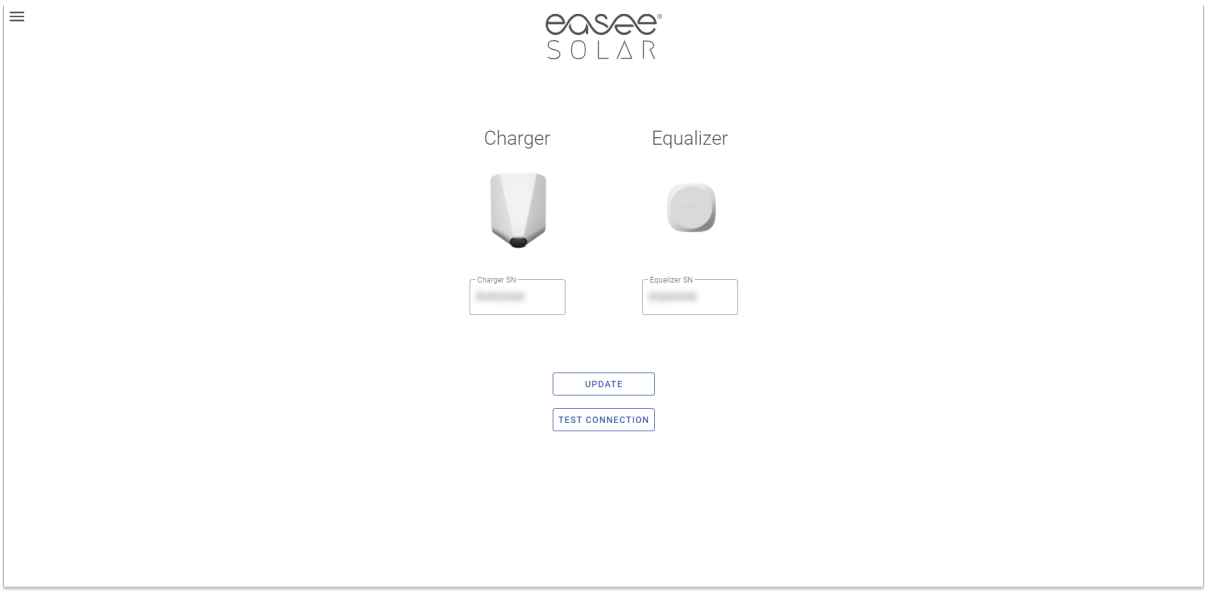
9.4 Skjermbilder webportalen






Options
Charger Regulation
Enable/Disable charger regulation based on selected mode
Regulation Modes
• Basic mode makes sure that there is enough current before assigning DCC DCC=(total-consumption)+solar
• Basic+LR mode uses linear regression model to predict solar power. β describes if slope is rising or falling, while R2 coefficient determines spreading in data points
Mode
Basic
SET
Regulation Interval
Set interval for regulation. 1 sec = 1000
WARNING: Setting this value to low may create voltage oscillations on the grid
Interval
60000
SET
Solar Simulator
Enable/Disable solar simulator (RNG)
Min/max Values
Change simulator min/max values (0-150)
Min
69
Max
70
SET
Linear Regression Sample Rate
Change number of samples for linear regression model. ~1 sample/sec
Sample rate
30
SET
Scale Factor
Change scale factor of incomming solar energy. (Default: 100000)
Scale factor
100000
SET
CLOSE
Total
Solar
Grid
4000
2000
0
-2000
-4000
Site
Total Capacity 13.9 kW
Power Consumption 1.2 kW
Solar
Scaled By 100000
Total Capacity 5.6 kW
Scaled Power 1.2 kW
Real Power 0.0119 W
Today's Production 0 Wh
Grid
Import 0 kW
Export 0.0 kW
Charger
Charger Status Standby
Grid Type IT 1-Ph
Max Charger Current 32 A / 7.4 kW
Dynamic Charger Current 32 A / 7.4 kW
Charging Current L1 0 A / 0.0 kW
Charging Current L2 0 A / 0.0 kW
Charging Current L3 0 A / 0.0 kW


easee SOLAR
Easee WSS
Get access token with easee credentials
Username
Password
AUTHORIZE
OR
Paste your token here
Access Token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ
UPDATE TOKEN
DELETE TOKEN


easee SOLAR
Charger
Equalizer
Charger SN
Equalizer SN
UPDATE
TEST CONNECTION