

## **Prosjektinnlevering – Gruppe 9**

Emil Berglund

Andreas B. Olaussen

Khalid H. Osman

Sebastian W. Thomsen

Ida K. Tollaksen

Høgskolen i Østfold

Ord: 9935

Emnekode: ITF20319

## Innholdsfortegnelse

<b>INTRODUKSJON.....</b>	<b>4</b>
<b>PROBLEMSTILLING .....</b>	<b>5</b>
<b>LØSNINGSBESKRIVELSE.....</b>	<b>6</b>
<b>KJERNEFUNKSJONALITET: .....</b>	<b>6</b>
<b>ÅPEN OG FLEKSIBEL PLATTFORM .....</b>	<b>6</b>
<b>DATA OG TILGANG .....</b>	<b>6</b>
<b>EKSPANDERBARHET OG ANTAKELSER .....</b>	<b>6</b>
<b>BEGRENSNINGER .....</b>	<b>7</b>
<b>FUNKSJONELLE KRAV .....</b>	<b>9</b>
IDENTIFISERING AV NØDVENDIGE FUNKSJONER .....	9
FUNKSJONER RELATERT TIL BRUKEREN.....	10
GRUPPEKRAV BASERT PÅ FUNKSJONELLE OMRÅDER .....	12
FORKLARING AV KRAV SOM DEKKES I PROTOTYPEN .....	12
<b>IKKE-FUNKSJONELLE KRAV:.....</b>	<b>13</b>
YTELSE .....	13
SIKKERHET .....	14
TILGJENGELIGHET.....	14
BRUKERVENNLIGHET.....	14
SKALERBARHET.....	14
EKSTERNE AVHENGIGHETER .....	15
TESTING .....	15
OPPSUMMERING .....	15
<b>BRUKERE AV SYSTEMET .....</b>	<b>15</b>
JULIE STRØM.....	16
MARKUS JOHANSEN .....	18
<b>SYSTEMARKITEKTUR.....</b>	<b>20</b>
GITHUB:.....	20
MAPPESTRUKTUR .....	21
KODESTRUKTUR .....	21
<b>PROTOTYPEDOKUMENTASJON .....</b>	<b>23</b>
<b>TESTING.....</b>	<b>27</b>
<b>UTVIKLINGSPROSESS .....</b>	<b>30</b>

SPRINTS .....	30
<i>Sprint 1: Design og konseptutvikling</i> .....	30
<i>Sprint 2: Programmering og evaluering</i> .....	30
<i>Sprint 3: Fullføring og dokumentasjon</i> .....	31
ERFARINGER .....	31
KODEPRINSIPPER .....	31
I UTVIKLINGEN AV SYSTEMET FULGTE VI NOEN DEFINERTE KODEPRINSIPPER FOR Å SIKRE AT KODEN VAR STRUKTURERT, SAMTIDIG SOM DET VAR VEDLIKE HOLDBART. ....	31
AVHENGIGHETER .....	32
VALG UNDERVEIS .....	32
<b>INSTRUKSJONER FOR BRUK.....</b>	<b>34</b>
FRONT-END.....	34
<i>Introduksjon til Vue og Quasar</i> .....	34
<i>Filstruktur</i> .....	34
<i>Kjøring og testing av front-end</i> .....	35
BACK-END .....	36
<i>ElectricityPrices</i> .....	38
<i>GPS</i> .....	39
<i>Notifications</i> .....	40
<i>Users</i> .....	43
<b>KONKLUSJON OG DISKUSJON RUNDT VIDERE ARBEID .....</b>	<b>44</b>
KONKLUSJON .....	44
VIDERE ARBEID .....	44
<b>MODELLER OG VEDLEGG.....</b>	<b>46</b>
<b>RELEVANTE LENKER .....</b>	<b>46</b>

# Introduksjon

I dagens samfunn ser vi en stadig økende interesse for smarte løsninger som gjør hverdagen enklere og mer effektiv. For oppstartsbedrifter som ønsker å tilby innovative produkter, kan det imidlertid være utfordrende å utvikle løsninger som både er brukervennlige og tilpasset målgruppens behov, samtidig som løsningen differensierer seg selv på markedet fra andre aktører.

Denne dokumentasjonen tar utgangspunkt i en oppstartsbedrift som ønsker å utvikle et smart system for strømvarsling og optimal lading av elbiler. Målet er å lage en løsning som er enkel å bruke for folk med begrenset IT-kompetanse, samtidig som den leverer avanserte funksjoner som strømprisanalyse, varslinger, og anbefalinger om når det lønner seg å lade bilen.

Dokumentet viser hele utviklingsprosessen fra kravspesifikasjon til ferdig prototype. Videre beskriver det kjernesystemet og funksjonene som er nødvendig for at kunden skal forstå hvordan produktet fungerer, og hvordan man kan bygge videre på det. Det inneholder detaljer om systemets funksjonalitet, tekniske krav, arkitektur, testing og hvordan vi har gått fram for å komme fram til en løsning som tilfredsstiller kundens behov.

Denne dokumentasjonen har derfor to formål: Å demonstrere hvordan et teknologisk konsept kan gå fra idé til prototype, og å vise hvordan et solid fundament kan legges for videreutvikling. Det er vår forhåpning at dette arbeidet ikke bare dekker oppdragsgivers behov, men også inspirerer til videre innovasjon innen smart energistyring og bærekraftig teknologi.

## Problemstilling

Problemstillingen vi sto ovenfor som ledet oss til idéen vår, handlet mye om alminnelighet og åpenhet. I dag er det ikke uvanlig at strømkunder har egne applikasjoner og tjenester for sine strømkunder. I disse applikasjonene finner man som regel funksjoner som smart-lading og andre muligheter for assistanse rundt lading. Disse er derimot veldig lukkede løsninger og krever ofte ekstra tilbehør eller fysiske produkter fra aktøren direkte, eller fra produsenter som støttes av strømselskapet. Ordene «lukkede løsninger» refererer her til at funksjonene som man noen ganger finner i slike applikasjoner, ikke er tilgjengelig for alle.

Vi tenkte derfor: «Hvordan kan vi utvikle en åpen og brukervennlig løsning for elbillading som eliminerer avhengigheten av proprietære løsninger og spesifikke strømselskaper, samtidig som den tilbyr avanserte funksjoner som optimalisert lading og strømprisanalyse?»

# Løsningsbeskrivelse

For å møte utfordringen i dagens marked med lukkede løsninger for elbillading, er det valgt å tilrettelegge for en åpen og brukervennlig tjeneste som er uavhengig av spesifikke strømselskaper eller fysiske tilbehør. Hovedmålet er å tilby en fleksibel løsning som tilpasser seg brukerens behov, uten tekniske barrierer, samtidig som den gjør lading enklere.

## Kjernefunksjonalitet:

Produktets kjernefunksjonalitet er å varsle brukeren når bilen bør lades. Dette kan gjøres på flere måter, men hovedsakelig basert på disse verdiene:

- Tidligere, gjeldene og fremtidige strømpriser
- Prognoser for prisutvikling
- Gjenværende batteriprosent på bilen

## Åpen og fleksibel plattform

I motsetning til eksisterende løsninger fra strømselskapene, Vibb og Tibber, skal ikke dette systemet kreve noen tilknytning til en spesifikk strømleverandør eller innkjøp av kompatible produkter. Dette gjør løsningen tilgjengelig for alle, uavhengig av hvilket system eller leverandør de bruker.

## Data og tilgang

For at tjenesten skal fungere optimalt kreves det visse data og tilganger fra brukeren. Disse er hovedpunktene:

- Posisjonstilgang: For å avgjøre når brukeren ankommer hjemmet.
- Strømdata: For nåværende og estimerte priser.
- Batteriprosent: For å sikre at varsler er relevante og ikke sendes unødvendig.

## Ekspanderbarhet og antakelser

Produktet bør kunne utvikles videre basert på brukerbehov og tilbakemeldinger. Det er her viktig å ikke overtenke løsningen under utviklingen. Eksempler på fremtidige utvidelser kan derimot være som følger:

- Smarte varslinger: Tilpassede varsler, basert på kjørevaner og batterinivå.
- Fysisk produkt: En skjerm i garasjen, som aktiveres når bilen parkeres som presenterer brukeren med sanntidsdata, og gir anbefaling visuelt.

## Begrensninger

Ettersom løsningens hovedmål er enkelhet og åpenhet, vil det være klare og tydelige begrensninger for den. Dette er noen typiske begrensninger som denne løsningen kan støte på:

1. Begrenset tilgang til data.
  - a. Strømprisdata: Nøyaktige og oppdaterte strømpriser kan være vanskelige å få tak i, dersom strømleverandøren ikke deler denne informasjonen via åpne API-er. Dette kan føre til at man må nøye seg med generaliserte priser fra åpne kilder, som ikke vil være tilpasset brukerens valg av strømselskap.
  - b. Batteriprosent: For enkelte bilmodeller er det nødvendig med spesiell programvare eller tillatelse fra bilprodusent for å hente ut batteridata, noe som kan begrense den universelle kompatibiliteten.
2. Avhengig av brukerens tillatelser
  - a. Løsningen krever tilgang til brukerens posisjon og bilens data. Mange brukere kan være skeptiske til å gi slike tillatelser på grunn av personvern og sikkerhetsbekymringer.
3. Begrenset teknologi hos brukeren
  - a. Ikke alle brukere har smarttelefoner eller apper som er kompatible med moderne API-er og systemer. Brukere med eldre biler eller telefoner uten smart-funksjonalitet kan også møte utfordringer med å bruke løsningen.
4. Kompatibilitet og standarder
  - a. Elektriske biler og ladeløsninger varierer mye mellom produsenter og leverandører. Hvis det ikke finnes en standardisert metode for datainnhenting og kommunikasjon, kan løsningen ha begrenset bruk for noen biltyper eller ladestasjoner.
5. Pålitelige prognoser for strømpris
  - a. Fremtidige strømpriser er ofte uforutsigbare og kan påvirkes av faktorer som vær, etterspørsel, og energimarkedet. Feilaktige prognoser kan føre til at brukeren mottar dårlige anbefalinger, noe som svekker tilliten til løsningen.
6. Irriterende varsler
  - a. Selv med tilpasning kan varsler oppleves som irriterende hvis de er for hyppige eller ikke relevante (f.eks. hvis brukeren bare har kjørt en kort tur). Dette kan føre til at brukeren skruer av funksjonaliteten.
7. Juridiske og regulatoriske barrierer

- a. Tilgang til strømdata og bilinformasjon kan være underlagt nasjonale lover og regler, som kan variere fra land til land. Dette kan hindre en global lansering av løsningen uten tilpasninger.



## Funksjonelle krav

### Identifisering av nødvendige funksjoner

Funksjonelle krav beskriver hvilke funksjoner systemet krever for å oppfylle både brukerens behov og løse oppdragsgivers problemstilling. Tabellen viser en oversikt over kravene, med status, utviklingsomfang og forretningsnytte.

ID	Beskrivelse	Status	Utviklingsomfang	Forretningsnytte
FK1	Bruker skal kunne registrere seg med både e-post og passord	Fullført	Lav	Høy
FK2	Bruker skal kunne logge inn med e-post og passord	Fullført	Lav	Høy
FK3	Systemet skal kunne hente strømpriser fra et åpent API	Fullført	Middels	Høy
FK4	Bruker skal motta varslinger når strømpriser er lav nok til optimal ladning	Fullført	Høy	Høy
FK5	Systemet skal vise nåværende, fremtidige og historiske strømpriser i en graf	Fullført	Middels	Middels
FK6	Systemet skal kunne identifisere brukerens posisjon og registrere når bruker har ankommet hjem	Fullført	Middels	Høy
FK7	Bruker skal kunne gi tillatelser for sin posisjon og varsler ved første gangs bruk	Fullført	Lav	Høy
FK8	Bruker skal kunne både legge til og se informasjon om registrerte biler i et dashboard	Delvis Fullført	Høy	Middels
FK9	Bruker skal kunne se anbefalinger om når det lønner seg for bruker å lade basert på både strømpris og batteri	Delvis Fullført	Høy	Høy

## Funksjoner relatert til brukeren

### 1. Registrering og innlogging (FK1, FK2)

Brukeren skal kunne opprette konto med e-post og passord. Bruker skal også få kunne logge inn for å få tilgang til personlig tilpasset strømdata og varsler. Dette skal videre føre til en sikker tilgang til brukerens data.

**Implementasjon:** Firebase Authentication håndterer autentisering. Funksjonene er testet og sikre.

Det fungerer slik at når bruker registrer seg, lagres e-post og passord i Firebase Authentication. Dette gir en sikker og skalerbar løsning for brukerhåndtering. Når bruker logger inn sjekker Firebase om e-post og passord stemmer overens med eksisterende konto.

Registrering og innloggings funksjonene har blitt testet med ulike brukerkontoer blant gruppen for å sikre at funksjonene fungerer korrekt i alle scenarioer.

### 2. Varslingssystem (FK4, FK7)

Varslingssystemet har i oppgave å sikre at bruker mottar varsler når det er økonomisk nyttig å lade bilen. Dette er for å redusere strømkostnader og optimalisere ladeplan.

**Implementasjon:** Firebase Cloud Messaging brukes for push varsler. Data fra Strømpris - API og posisjonskontroll trigger varsler.

Det blir sendt push-varsler til brukerens mobil, ved hjelp av Firebase Cloud Messaging. Dette trigges av strømprisdata og brukerens posisjon. Eksempelvis, hvis strømprisen faller under et gitt nivå og brukeren er innenfor definerte hjem posisjonen, sendes det varsel til brukeren.

Varslingssystemet har blitt testet ved å simulere ulike strømpriser og posisjoner for å sjekke om varsler kun sendes ut når det er relevant. Ved hjelp av Java tester ble det blant annet testet at varsler blir sendt på riktig tidspunkt og at irrelevant varsler unngås.

### 3. Strømprishåndterings (FK3, FK5)

Denne delen av systemet henter strømpriser og visualiserer disse i en graf. Dette er for å gi bruker en oversikt over nåværende historisk og fremtidige priser.

**Implementasjon:** API-data hentes og behandles av pris tjeneste klasser. Diagram klassen brukes til å vise det grafiske.

Det fungerer slik at API-integrasjon henter priser i sanntid. Deretter lagres prisene midlertidig i en database og visualiseres i diagram. Eksempelvis kan en bruker planlegge lading ved å følge med på grafen som viser hvordan strømprisen utvikler seg over tid.

Strømpris funksjonene er testet ved hjelp av Java klasser som sjekker om prisdata er korrekt.

#### 4. Posisjonsbaserte tjenester (FK6)

GPS delen av systemet går ut på at applikasjonen sjekker om bruker er innenfor en radius definert av bruker som kalles “hjem”. Når bruker er innenfor radiusen sender den ut varsler til bruker. Dette gjøres for å unngå irrelevante varsler når bruker ikke er i den definerte “hjemme” posisjonen.

**Implementasjon:** En utviklet Java klasse “HomeChecker” bruker Haversine-formelen for å bygge avstander.

GPS funksjonen fungerer slik at “HomeChecker” klassen sjekker avstanden mellom brukers nåværende posisjon og de definerte hjemme-koordinatene. Dersom avstanden fra brukerens posisjon er innenfor en radius på 100 meter, anses brukeren som “hjemme”.

Eksempel på scenario kan være som følger: Bruker ankommer hjemmet. Systemet registrer posisjonen og sender ut et varsel om ladning basert på strømpris.

GPS funksjonen er testet i Java klasser som bekrefter at radiusberegningen er korrekt.

Integrasjon mellom posisjon og varsler testes også i Java, for å sikre at systemet reagerer riktig.

#### 5. Dashboard og ladeanbefalinger (FK8, FK9)

Dashboard delen i applikasjonen gir brukeren oversikt over brukerens registrerte biler og deres ladestatus. Brukeren får anbefalinger på lading basert på strømpriser og bilens batteristatus.

**Implementasjon:** Dashboard er implementert i “DashboardPage.vue”. Anbefalinger på lading er delvis ferdig og må videre utvikles.

Denne delen av applikasjonen fungerer slik at bruker legger til biler manuelt i applikasjonens dashboard og kan se detaljer om batteristatus og forbrukshistorikk. Videre beregnes ladeanbefalinger ut ifra strømprisen og bilens gjenværende batteri-nivå.

**Eksempel fra applikasjonens prototype:**

- Dashboard: “Volkswagen ID.3 - Batterinivå 67%”
- Anbefaling: “Det lønner seg å lade senere i kveld.”

Dashbordet er testet for grunnleggende funksjonalitet, som å vise registrerte biler.

Ladeanbefalinger er også testet i Java, hvor en test klasse tester for når det er optimalt å lade.

## Gruppekrav basert på funksjonelle områder

Funksjonelt område	Funksjoner	Status	Beskrivelse
Brukerhåndtering	FK1(Registrering) FK2(Innlogging)	Fullstendig implementert	Firestore Authentication: Sikrer trygg tilgang til data
Datahåndtering	FK3(Strømpriser) FK5(Grafvisning)	Fullstendig implementert	Henting og visualisering av strømpriser fra API
Varslingssystem	FK4(Varsler) FK6(Posisjon) FK7(Tillatelser)	Fullstendig implementert	Sender ut relevante varsler basert på pris og posisjon
Dashbord og biler	FK8(Dashbord) FK9(Ladeanbefalinger)	Fullstendig implementert	Viser batteristatus og gir anbefalinger til lading

## Forklaring av krav som dekkes i prototypen

### Identifisering av krav

Kravene i systemet ble identifisert ut ifra en analyse vi gjorde innad i gruppen. Vi diskuterte blant annet behov for systemet og brukerens forventninger. Målet med selve systemet var å at alle nødvendige funksjoner ble definert for å løse problemstilling vi satt fra starten. Viktige krav som brukerhåndtering, innhenting av strømpris og posisjonskontroll ble prioritert som de viktigste elementene i systemet.

### Prioritering av krav

Allerede i planlegging i første sprint bestemte vi oss får å bruke «t-shirt sizing»-metoden. Dette gjorde det mulig for å gruppere ulike funksjoner etter hvor krevende de ville være å implementere i systemet. Denne prioriteringsprosessen førte til at vi kunne levere maksimal verdi til slutt, samtidig som vi dekket de tekniske kravene.

### **Implementering av prioriterte krav**

FK1 – FK7 er de høyeste prioriterte kravene som ble implementert i prototypen for å sikre at systemet kan demonstrere kjernefunksjonaliteten. Disse er også testet grundig etter implementasjonene, for å sikre at funksjonene oppfører seg som forventet i henhold til de tekniske kravene. Funksjonell testing ble gjort for å validere hver funksjon, samtidig som integrasjon testingen sørget for at komponentene, som varsler og API-data, samhandlet på riktig måte.

### **Delvis implementerte krav**

Hovedtyngden av kravene ble dekket i prototypen, men noen av funksjonene krever fremdeles videreutvikling. FK8 som handler om dashboard og FK9 som går ut på ladeanbefalinger er delvis implementert, men mangler full dataintegrasjon og mer presis algoritme for lade anbefalinger basert på sanntidsdata.

## **Ikke-funksjonelle krav:**

Ikke funksjonelle krav handler om delen av systemet som sikrer at funksjoner fungerer effektivt, sikkert og på en brukervennlig måte under ulike forhold. Disse kravene er viktige for at systemet er lett tilgjengelig for brukere uten teknisk kompetanse, samtidig som at systemet opprettholder en høy grad av ytelse, sikkerhet og stabilitet.

### **Ytelse**

For at applikasjonen skal oppnå optimal brukeropplevelse er det viktig at systemet kan håndtere både brukere og levere raske responstider samtidig, spesielt for tidssensitive funksjoner som automatiske varsler basert på posisjon og strømpris. Systemet er designet for å sikre at API-forespørsler har en rask responstid selv under perioder med høy belastning og at varsler kan sendes ut rask etter oppfylte kriterier. Derfor er det viktig at alle funksjoner er

optimalisert for å redusere responstiden, og at tester blir gjort for at systemet kan håndtere mange brukere uten ytelsestap.

## Sikkerhet

Sikkerheten i applikasjonen er et av de mest sentrale kravene i systemet, i og med at det blir håndtert sensitiv brukerdata, som posisjonsinformasjon og autentisering. Systemet og eksterne tjenesters kommunikasjon er sikkert med kryptering, samtidig som passord lagres i hash format for å beskytte mot uautorisert tilgang. Det har også blitt gjort andre sikkerhetstiltak mot angrep for å minimere risikoen for sikkerhetsbrudd. I tillegg utføres det regelmessige tester for å sikre systemet samtidig som vi eliminerer sårbarheter. Slike tiltak beskytter brukere.

## Tilgjengelighet

At systemet er tilgjengelig, er avgjørende. Det er satt som krav at systemet skal ha oppetid minimum 99% av tiden og fungerer på alle plattformer. Feilmeldingene i systemet er designet for å være veiledende for bruker på en enkel måte. For å oppnå den høyeste tilgjengeligheten som mulig gjøres det regelmessige tester i servere, samtidig som det gjøres sikkerhetskopier så ofte som mulig. Firebase funksjoner er implementert for å identifisere feil raskt for å sikre at funksjonaliteter fungerer på enheten til enhver tid.

## Brukervennlighet

Systemet ble utviklet med tanke på brukere uten noe teknisk bakgrunn. Systemet er bygd slik at det er selvforklarende og intuitivt brukergrensesnitt. Det vil si at varsler og anbefalinger presenteres på en tydelig måte. Navigasjonen er bygd på en måte som er logisk og lett og forstå. Brukervennligheten har blitt evaluert og testet igjennom brukertester med målgruppe, der tilbakemeldingene har videre blitt brukt til å forbedre funksjonalitet og design. Personas og scenarioer har vært sentrale for å identifisere målgrupper og sikre at brukerbehovet er møtt.

## Skalerbarhet

Systemet er bygget for å kunne håndtere fremtidig vekst. Det er et modulært system noe som gjør at det videre kan bygges på og er enkelt for både utvidelse og vedlikeholdene. Systemet skal klare å tåle en økning i brukere uten betydelig tap av ytelse. Derfor er det viktig å gjøre regelmessige ytelsestester for å sikre at systemet kan vokse etter brukernes behov.

## Eksterne avhengigheter

Systemet er bygd på noen eksterne avhengigheter som åpne API-er for strømdata og Firebase for både varsler og autentisering. For å sikre robusthet har vi implementert midlertidige funksjoner i prototypen som simulerer eksterne tjenester, slik at funksjonene kan testes uten faktiske avhengigheter. Senere kan vi erstatte disse simulerte funksjonene med faktiske integrasjoner, noe som gjør systemet til slutt blir et komplett og operativt system. Dette gjør det mulig for systemet å bli testet, samtidig som det videre kan bygges på med faktiske integrasjoner.

## Testing

For å validere at ikke-funksjonelle krav oppfylles, har vi gjennomført testing. Prototypen har blitt testet for stress tester som evaluerer systemets ytelse under belastning. Systemet har også blitt testet for potensielle sikkerhetssvakheter. Tilgjengeligheten har blitt testet ved å simulere ulike feilscenarier. Brukervennligheten har også blitt testet med reelle brukere, som bekreftet at systemet er både enkelt og intuitivt for brukere.

## Oppsummering

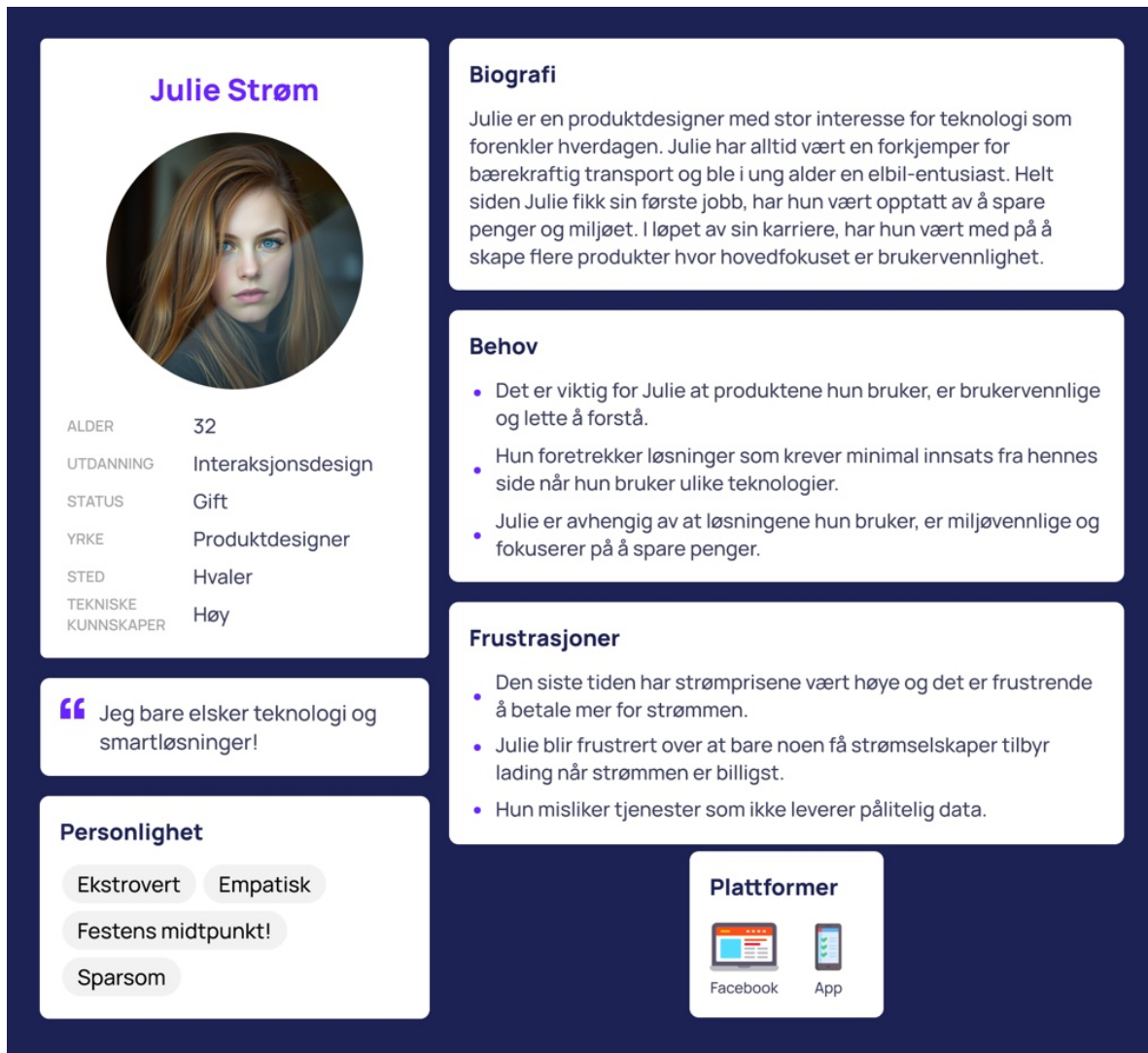
Ved å fokusere på ytelse, sikkerhet, tilgjengelighet, brukervennlighet, skalerbarhet og håndtering av eksterne avhengigheter, sikrer systemet både en robust og pålitelig løsning. Dette er gjort for at brukeren skal ha enkel tilgang til smartteknologi, samtidig som det møter våre mål om å gjøre teknologien tilgjengelig for brukere uten teknisk kompetanse. Derfor er delen med ikke – funksjonelle krav en viktig del av systemet og tiltakene sikrer at systemet er klart for videreutvikling.

## Brukere av systemet


Brukerne av vårt system er hovedsakelig el-bileiere som er opptatt av å spare både miljø og penger gjennom effektive løsninger. De er opptatt av smartteknologi og bærekraft, men har ikke nødvendigvis dyptgående it-kunnskaper. Disse brukerne prioriterer enkle og intuitive løsninger som er visuelt attraktiv og lett å forstå. Produktet vårt skal passe “mannen i gata” og være et verktøy som ikke krever opplæring og som gir god verdi til kundene.

Under er en visuell representasjon av “Julie Strøm” og “Markus Johansen” som er fremtidige brukere av systemet vi ønsker å tilby.

## Julie Strøm



**Julie Strøm**



ALDER	32
UTDANNING	Interaksjonsdesign
STATUS	Gift
YRKE	Produktdesigner
STED	Hvaler
TEKNISKE KUNNSKAPER	Høy

**Biografi**

Julie er en produktdesigner med stor interesse for teknologi som forenkler hverdagen. Julie har alltid vært en forkjemper for bærekraftig transport og ble i ung alder en elbil-entusiast. Helt siden Julie fikk sin første jobb, har hun vært opptatt av å spare penger og miljøet. I løpet av sin karriere, har hun vært med på å skape flere produkter hvor hovedfokuset er brukervennlighet.

**Behov**

- Det er viktig for Julie at produktene hun bruker, er brukervennlige og lette å forstå.
- Hun foretrekker løsninger som krever minimal innsats fra hennes side når hun bruker ulike teknologier.
- Julie er avhengig av at løsningene hun bruker, er miljøvennlige og fokuserer på å spare penger.

**Frustrasjoner**

- Den siste tiden har strømprisene vært høye og det er frustrerende å betale mer for strømmen.
- Julie blir frustrert over at bare noen få strømselskaper tilbyr lading når strømmen er billigst.
- Hun misliker tjenester som ikke leverer pålitelig data.

**Personlighet**

Ekstrovert   Empatisk

Festens midtpunkt!

Sparsom

**Plattformer**

Facebook   App

**Quote:** Jeg bare elsker teknologi og smartløsninger!

## Brukerhistorie 1: Lading når strømmen er billigst

Som en miljøinteressert elbileier ønsker jeg å kunne planlegge lading av bilen min når strømprisen er lavest, slik at jeg kan spare penger og miljøet.

## Scenario 1: Planlegging av lading

Julie kommer hjem fra jobb og har koblet bilen sin til hjemmeladeren sin. Hun henter seg en kopp kaffe, setter seg ned i sofaen og åpner VarsEL-appen for å sjekke dagens strømpriser. Julie ser at strømprisen er lavest mellom kl.17.00-22.00. Julie vet dermed at hun ikke trenger å tenke på å lade bilen før den tid, og kan fokusere på andre.



## **Brukerhistorie 2: Oversiktlig informasjon**

Som elbileier ønsker jeg å kunne se og legge til informasjon om bilen min i et oversiktlig dashboard, slik at jeg kan ha kontroll over lading og informasjon knyttet til bilen min.

### **Scenario 2: Registrere bilen**

Julie har akkurat kjøpt seg ny elbil og klør i fingrene etter å registre den i VarsEL-appen sin. Hun åpner appen, navigerer seg til dashboardet, trykker på “Se alle kjøretøy” og legger til den nye bilen. På Dashboardet har hun nå en oversikt over bilen sin, batteristatus, høyest og lavest strømpris, samt nåværende strømpris og mulighet til å navigere seg til en strømprisgraf.

## **Brukerhistorie 3: Varsling**

Jeg ønsker å ha muligheten til å kunne bli varslet når strømprisen er lavest når jeg ikke har mulighet til å analysere strømprisene selv.

### **Scenario 3: På farten**

Julie har en travel dag og er i full gang med å forberede middag til gjester og deres barn. Hun har ikke tid til å sjekke dagens strømpriser manuelt. Midt i forberedelsene får hun et varsel fra VarsEL-appen om at strømprisen er på sitt laveste. Med denne informasjonen vet hun at det lønner seg å lade bilen, før hun deretter kan fortsette middagsforberedelsene uten å ha det i bakhodet.

## Markus Johansen

### Markus Johansen



ALDER	44 år
UTDANNING	Master (Lektor)
STATUS	Gift
YRKE	Lærer
STED	Fredrikstad
TEKNISK KUNNSKAP	Høy

#### Biografi

Markus Johansen er en mann som bor i en enebolig i Fredrikstad, med sine tre barn og kone. Han har alltid vært opptatt av nyere teknologi og er alltid på utsikt etter noe nytt. 4 år tilbake byttet han over fra bensin bil til EL-bil. Som ungdomsskolelærer pendler han alltid i hverdagene til arbeidsplassen, som ligger sentrum i Fredrikstad. Ellers kjører han ofte familien rundt, i og med at de er en aktiv familie.

#### Behov

- Markus er opptatt av å redusere kostnader, inkludert strømudgifter, spesielt når det kommer til EL-bilen
- Markus ønsker å være mer miljøvennlig, og derfor opptatt av å lade bilen på tider med lavere strømforbruk.
- Markus er opptatt av at ting er lett og effektivt i hverdagen, og derfor verdsetter ekle l

#### Frustrasjoner

- Markus er en travel person, med familie og fritidsaktiviteter, så han setter pris på løsninger som sparer han tid.
- Markus mener det er utfordrende å konstant følge med på strømprisene

#### Personlighet

Ekstrovert
Strukturert

Optimistisk
Planlegger

#### Plattformer


Facebook


App

“ Jeg ønsker rett og slett en lettere hverdag, ved hjelp av nyere teknologi

### Brukerhistorie 1: Registrering i appen

Som en ny bruker av VarsEL-appen ønsker jeg å registrere meg enkelt og gi tillatelser til posisjon og varsler, slik at jeg kan bruke systemet effektivt og motta varslinger om strømpriser.

### Scenario 1: Førstegangsregistrering

Markus åpner VarsEL-appen for første gang og blir møtt av en velkomstmelding. Han trykker på knappen "Kom i gang" og blir guidet gjennom en kort registreringsprosess. Først velger Markus sin strømleverandør fra en liste. Deretter blir han spurt om å tillate varslinger om

strømpriser, som han enkelt huker av. Appen ber ham deretter om å gi tilgang til posisjonstjenester, slik at funksjonene kan tilpasses der han befinner seg.

Til slutt blir Markus gitt to alternativer for registrering: å opprette en konto med e-post og passord, eller å logge inn med Google-kontoen sin. Han velger å bruke Google-kontoen for en rask registrering. Etter noen sekunder kommer han til dashbordet, der han ser en oversikt over strømpriser og får videre tilgang til appens funksjoner.

## **Brukerhistorie 2: Se strømpriser i graf**

Som elbileier ønsker jeg å kunne se sanntidsstrømpriser, fremtidige- og historiske strømpriser via en graf, slik at det blir enklere for meg å se når jeg burde lade bilen.

## **Scenario 2: Bruke strømgraf**

Markus har lunsjpause på jobben og tar frem VarsEL-appen. Han navigerer seg via dashbordet til strømprisgrafen og analyserer gårdsdagens, dagens og morgendagens strømpriser og gjør seg opp en mening på når han burde lade bilen sin.

## **Brukerhistorie 3: Posisjonsbasert varsling**

Som bruker av tjenesten ønsker jeg å at appen automatisk registrerer når jeg er hjemme, slik at jeg får personlige varslinger om lading basert på min posisjon.

## **Scenario 3: Posisjonsvarsling**

Markus kommer hjem fra kjøretur og VarsEL-appen registrerer at han er i nærheten av hjemmet sitt. Han får et varsel fra appen sin om at strømprisen er lav. Markus går dermed ut og plugger i bilen.

# Systemarkitektur

Uansett hvor man jobber, er det alltid lurt med en god struktur, både for sin egen del, men også for andres del. Dette vil føre til bedre oversikt, enklere vedlikehold og mer effektivt samarbeid. En god systemarkitektur legger grunnlaget for å bygge robuste og skalerbare løsninger som kan vokse og tilpasses i takt med virksomhetens behov.

Når man setter opp en systemarkitektur, bør man fokusere på flere hovedprinsipper, hvor disse er noen av dem:

1. Modularitet: Del gjerne opp systemet i mindre, selvstendige moduler/deler. Da blir det som regel enklere å forstå, teste og endre på uten å påvirke hele løsninger.
2. Skalerbarhet: Arkitekturen bør kunne håndtere å bli utvidet. Struktur er derfor særdeles viktig å få på plass fra første stund, slik at man ikke må endre på hele oppsettet ved senere utvidelse.
3. Robusthet: Systemet bør tåle uventende situasjoner, og håndter disse på en god måte.
4. Brukervennlighet: Strukturer og gjør utvikling på en brukervennlig måte. Enkle, tydelige grensesnitt og godt dokumenterte prosesser er essensielt.
5. Sikkerhet: Systemet bør bygges med tanke på datasikkerhet. Klarer man ikke å implementere sikre løsninger, er det bedre å utelate disse.

Under finnes det mer detaljerte forklaringer og beskrivelser til hvordan man bør strukturere ulike arbeidsområder slik at man også følger prinsippene over.

## GitHub:

1. **Bruk av grener (branches)**: Ved å opprette dedikerte grener for ulike deler av prosjektet; eksempelvis ulike funksjoner eller ulike seksjoner av en applikasjon, vil det være enklere å spore endringer, fordele arbeidsoppgaver, og eventuelt gå tilbake, hvis noe ble feil. Det vil også være en sikkerhet for andre medlemmer i prosjektet, at man ikke overskriver hverandre.
2. **Meningsfulle commit-meldinger**: Ved å skrive korte og presise meldinger som beskriver hva som ble endret og hvorfor, er det veldig enkelt for alle å skjønne hvorfor endringene ble gjort. Dette vil gjøre det enklere for alle å lese og forstå i etterkant.

3. Pull requests: Opprett gjerne pull requests og ta imot tilbakemeldinger fra andre medlemmer for å sikre at kvaliteten på koden er til alles standard før den slås sammen med noe annet.

## Mappestruktur

Mappestruktur er ofte prikken over i-en når det kommer til brukervennlighet og organisering av et prosjekt. En godt definert mappestruktur gir utviklere en intuitiv oversikt over prosjektet og gjør det enklere å finne, vedlikeholde og oppdatere kodebasen.

Det er ingen fasit på hvordan mapper og filer skal struktureres, men man bør legge filer med samme formål og overordnet betydning i samme mappe, og differensiere disse fra filer med andre formål. Eksempelvis i front-end bør man ha en egen mappe til «assets», altså en mappe som inneholder bilder og ekstrainnhold, som igjen legges unna andre filer som ikke ansees som «assets». I back-end bør man differensiere på ulike klasser, og avgrense tester fra klasser, slik at disse ikke blandes. Man bør også samle klasser/funksjoner som tilhører samme kategori for bedre oversikt.

Ved å ha en god mappestruktur er det ikke uvanlig at man opplever økt produktivitet, ettersom alle vet hvor man skal legge ulike filer, og alle vet hvor man finner filene. I tillegg vil vedlikehold være enklere. Med en klar inndeling, vil det lede til færre forandringer som minsker risikoen for å ødelegge noe. Til slutt legger en god mappestruktur et godt grunnlag for videre skalerbarhet og utvikling, ettersom strukturen allerede er på plass, og prosjektet kan derfor vokse uten å bli uoversiktlig.

## Kodestruktur

1. Følg prinsipper som DRY og YAGNI: Ved å følge prinsipper som DRY (Don't repeat yourself) og YAGNI (You aren't gonna need it) unngår man duplisert kode og redundant kode.
2. Bruk klare og konsistente navnekonvensjoner: Variabler, funksjoner, klasser og filer bør ha beskrivende navn som sier noe om deres funksjon. Disse skal derimot ikke være forkortet, og ikke for lange. Blir dette tilfellet, tyder det som regel på at de har

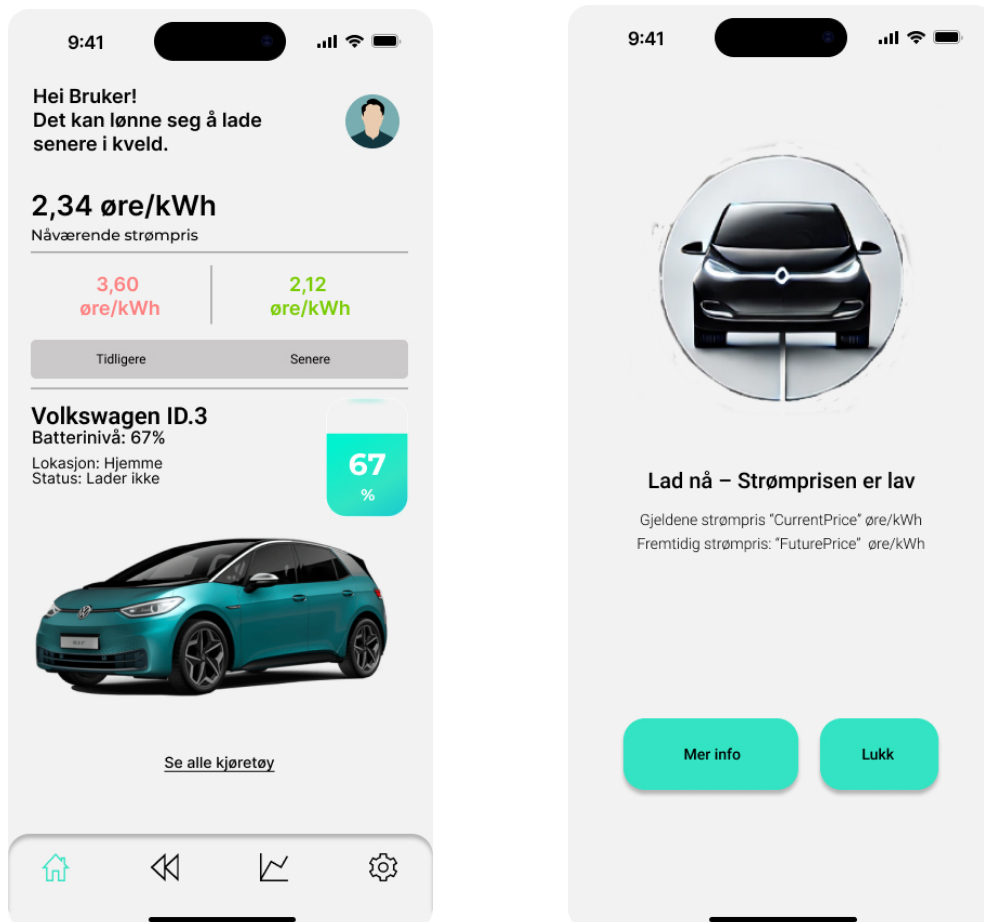
for stort ansvar, og bør fordeles utover. Navnene bør også være spesifikke, som sier akkurat hva som gjøres eller hva som inneholdes.

3. Kommenter når nødvendig: Det kan være greit å legge til kommentarer i koden for å forklare logikk, men unngå unødvendige kommentarer som er lett lesbart ved å se på hva koden gjør. Ender man opp med mye kommentarer, tyder det ofte på at man ikke har god navngivning.

# Prototypedokumentasjon

## VarsEL

Vi har gleden av å introdusere *VarsEL*! En app som gjør det enklere enn noensinne for elbileiere å spare både strøm og miljøet! Appen gir brukerne bedre kontroll over eget strømforbruk når det gjelder lading, samtidig som den gir mulighet til å spare penger i en tid med stadig stigende energipriser. Produktet vårt er en MVP (Minimum Viable Product), altså en tidlig versjon av applikasjonen som illustrerer hvordan sluttproduktet kan se ut. Vi har utviklet flere funksjoner, og gjennom hele prosessen har både kundene og brukerne vært i fokus.



Vår app, VarsEL, har blitt utviklet gjennom flere trinn. Vi startet prosessen med å lage personar, brukerhistorier og scenarioer, som nevnt tidligere. Deretter skisserte vi et interaktivt visuelt design i Figma (et verktøy for å lage wireframes og prototyper). Videre begynte vi

med utvikling og testing av kode i programmeringsspråket Java for å implementere ulike funksjoner som er viktige for systemet vårt. Samtidig har vi også laget en tidlig versjon av appen som kan testes på denne nettsiden: <https://varsel-app.pages.dev/auth/welcome>. I nettleseren kan du selv navigere rundt i appen og få et inntrykk av hvordan vi ser for oss at appen kan bli. Dere er selvfølgelig velkomne til å gjøre de endringene dere ønsker! Hvis du ser på siden i nettleseren, er det lurt å skalere ned vinduet for å få en mer helhetlig opplevelse av appen.

Appen inneholder funksjoner som planlegging av lading, varslinger basert på posisjon og strømpris i sanntid, en oversiktlig strømgraf og en lekker innloggingsside med muligheter for å logge på enten med e-post og passord, eller ved bruk av Google kontoen din!

## **Funksjonalitet**

Det første som møter brukeren når hen åpner VarsEL er velkomstsiden hvor man enkelt kan registrere seg i appen og få tilgang til dashbordet (Hovedmenyen). Når man har logget inn dukker brukerens navn opp i venstre hjørne og sier eksempelvis “Hei Jonathan!” Dette skjer automatisk når man har laget en bruker. Appen er koblet til “Firebase”, som er en database som trygt lagrer mailadresse og passord for innlogging med Google kontoen din.

Fra Dashbordet har brukerne mulighet til å se nåværende strømpris, samt den høyeste og laveste strømprisen for dagen i dag. Videre kan brukeren navigere seg til en side om ladehistorikk (Ingen funksjonalitet enda), en strømprisgraf som ved lansering skal kunne visuelt vise historiske strømpriser, strømprisen for i dag og i morgen. I tillegg kan man navigere seg til “Innstillinger”, hvor man kan bytte til “Darkmode” eller endre språket til engelsk. I VarsEL-appen får man også et varsel om når strømprisen er lavest, slik at brukeren slipper å følge med på dette selv. Vi har laget funksjonen som gjør dette, men ikke implementert det i selve prototypen.

## **Brukergrensesnittet**

Brukergrensesnittet for VarsEL-appen ble først designet i Figma og deretter implementert ved hjelp av Vue.js (som nevnt tidligere). Appen er utviklet med brukeren i fokus, med et enkelt og intuitivt design som sikrer enkel navigering. Målet har vært å gjøre appen lettforståelig og brukervennlig for alle.



## **Teknologi**

I arbeidet vårt har vi brukt programmeringsspråket Java til å lage flere funksjoner som kan tas i bruk med appen. Vi har også utviklet GPS-funksjonalitet som sjekker om brukeren er innenfor en 100 meter radius av huset sitt, hvor tanken er at VarsEL- appen skal kunne registrere når brukeren kommer hjem og i fremtiden vil det være mulig for appen å gi en indikasjon på om brukeren burde lade bilen basert på daværende strømpris. Strømpriser kan bli hentet fra fem forskjellige soner i Norge og i fremtidige versjoner av VarsEL skal prissonene være koblet til posisjonsteknologien slik at dette endrer seg basert på hvor man befinner seg.

I Java har vi laget funksjoner som ikke inngår i prototypen per dags dato, men som kan bli implementert i senere versjoner. I Java har vi laget en egen kobling til et åpent API for å kunne hente sanntidspriser og historiske priser. Dette skal inngå i strømprisgrafen i VarsEL slik at kunden kan ta et informert valg om når hen burde lade bilen, eller hvis vedkommende ønsker å se på tidligere strømpriser.

Selve VarsEL-appen skal vise dagens høyeste og laveste strømpris som nevnt i dashbordet og vi har laget funksjonene som gjør disse kalkulasjonene. For fremtidig bruk kan dere hvis dere ønsker også spesifisere eksakt data for når dere ønsker å se strømpriser til ulike historiske datoer. Koden vi har skrevet kan enkelt bli brukt til det dersom dere ønsker annen funksjonalitet enn hva produktet tilbyr på nåværende tidspunkt. Innloggingen med brukernavn og passord er også laget ferdig med Java og trenger kun å bli implementert i selve appen for at det skal fungere.

## **Begrensninger**

Prototypen vår i nettleseren er ikke ferdig utviklet og mangler flere funksjoner som utregning av hvor mye strøm man har brukt på lading, samt hvor mange kroner det har kostet. Over har jeg nevnt de funksjonene vi har laget i Java og det er behov for at dette produktet videreutvikles før det kan bli lansert til brukere.

## **Hvorfor er VarsEL verdifull for dere?**

VarsEL er utviklet for å møte reelle utfordringer som mange elbileiere står ovenfor i hverdagen. Produktet vårt kan skape verdi for brukere og bedrifter som prioriterer løsninger

som er brukervennlige og fungerer som et enkelt verktøy i hverdagen for å spare miljø og penger. Appen er ment for å automatisere overvåkning av strømpriser og gi brukerne kontroll i egen lading, uten at de trenger å sette seg inn i store tekniske ting.

VarsEL har et stort potensial for vekst og videre utvikling. Med høye drivstoffpriser og et større samfunnsmessig fokus på grønn energi er VarsEL en god og lønnsom løsning for brukere og dere. De ulike funksjonalitetene som vi tilbyr, kan gjenbrukes og dekke en større målgruppe som ønsker smarte ladeløsninger. I fremtidig utvikling av appen kan en god ide være å integrere VarsEL-appen med smart-hjem løsninger som kan gi enda større nytte til brukerne av tjenesten. For brukerne som er ekstra interesserte i strømpriser er appen også perfekt for de som kun ønsker å overvåke strømprisen.

Vi ser frem til å jobbe sammen med dere!

# Testing

Testplanen dekker både funksjonelle og tekniske krav til applikasjonen, med fokus på følgende områder:

1. Kjernefunksjoner:
  - Håndtering av brukere (registrering, innlogging, kryptering av passord).
  - Kontroll av brukerens geografiske posisjon mot en radius.
  - Kalkulasjon og presentasjon av strømpriser basert på eksterne datakilder.
2. Robusthet:
  - Håndtering av feil som ugyldige input, nettverksproblemer og uventet data.
3. Integrasjon:
  - Sikring av korrekt kommunikasjon mellom komponenter som brukerdata, lokasjonskontroll, og HTTP-forespørsler.
4. Ytelse og skalerbarhet:
  - Testing av moduler for effektivitet under forskjellige scenarioer.

Eksempler på automatiserte tester som dekker funksjonelle krav:

1. Brukerhåndtering
  - “UserServiceTests” og “UserRepositoryTests”:
    - Registrering og innlogging med korrekte input.
    - Passord lagres som hash, aldri i klartekst.
    - Feilhåndtering for eksisterende brukere og ugyldige input.
  - “UserTests”:
    - Validerer riktige verdier for e-post og hashed passord.
2. Lokasjonsbaserte funksjoner:
  - “HomeCheckerTest”:
    - Validerer om en bruker er innenfor eller utenfor en definert radius basert på GPS-koordinater.
  - “LocationServiceTest”:
    - Tester korrekthet og robusthet i lokasjonsdata fra GPS-simuleringer.
  - “MainActivityTest”:

- Tester integrasjon mellom lokasjonssjekk (“LocationService”) og radiusberegning (“HomeChecker”).

### 3. Strømprisfunksjonalitet:

- “ElectricityPriceCalculatorTests”:
  - Validerer korrekt beregning av strømpriser basert på ulike input.
- “ElectricityPriceDataTests”:
  - Sikrer korrekt mapping av eksterne data til interne datastrukturer.
- “ElectricityPriceParserTests”:
  - Tester parsing av rådata og håndtering av feilformaterte input.
- “ElectricityPriceUrlBuilderTests”:
  - Sikrer korrekt bygging av URL-er for API-forespørsler.

### 4. HTTP-kommunikasjon:

- “HttpClientTests”:
  - Håndtering av gyldige og ugyldige URL-er.
  - Validerer robusthet mot nettverksfeil som “IOException” og “URISyntaxException”.
- “HttpHandlerTests”:
  - Sikrer korrekt håndtering av HTTP-forespørsler og responser.

### 5. Strømprishenting og visning:

- “GetElectricityPricesTests”:
  - Tester at strømprisdata hentes korrekt fra eksterne kilder og prosesseres for visning.

### 6. Notifikasjon funksjonalitet

- “NotifikasjonTest”:
  - Tester at objekter av “Notifikasjon” blir opprettet på riktig måte og at de inneholder alle de nødvendige parameterene.
- “NotifikasjonServiceTest”
  - Testen at NotifikasjonService genererer de riktige push-notifikasjonene med riktige verdier basert på strømpris og batterinivå.
- “NotifikasjonSenderTest”
  - Tester at NotifikasjonSender sender riktig notifikasjon til Firebase basert på strømpris ved å bruke en mock av NotifikasjonService.

- Mockito brukes for å isolere testen og sikrer at kun PushNotifikasjonSender blir testet.
- “FirebaseInitializerTest”
  - Tester at Firebase blir initialisert riktig og kun én gang.

Hvordan tester viser korrekt oppførsel og feilsituasjoner.

Korrekt oppførsel:

- Brukerhåndtering: Tester som “testRegisterUser\_Success” og “testLoginUser\_Success” bekrefter riktig registrering og innlogging.
- Lokasjonsfunksjoner: “testUserInsideRadius” validerer korrekt deteksjon av brukerens posisjon i forhold til en radius.
- Strømrpiser: Tester som i “ElectricityPriceCalculatorTests” validerer at prisbereninger fungerer som forventet.

Feilsituasjoner:

- Input-feil:
  - “testRegisterUser\_EmptyEmail” og “testLoginUser\_WrongPassword” viser hvordan systemet håndterer ugyldig input.
- Nettverksproblemer:
  - “HttpGetClientTestingIOExceptionGetRequest” demonstrerer robusthet mot nettverksfeil.

# Utviklingsprosess

## Sprints

Prosjektet ble gjennomført i tre sprint-faser, hver med spesifikke mål og oppgaver for å sikre fremdrift og kvalitet.

### Sprint 1: Design og konseptutvikling

- Varighet: 3 uker
- Mål: Utforme plattformens design basert på kravspesifikasjonen.
- Oppgaver:
  - Utarbeide kravspesifikasjon, skrive om produktet og funksjoner.
  - Utvikle personas og scenarios for brukerne.
  - Designe dashboard, varslinger, logo og introduksjonssider.
  - Lage en side for strømpriser og implementere dark mode på hovedsidene.
- Resultat: En konkret designprototype som la grunnlag for videre utvikling.

### Sprint 2: Programmering og evaluering

- Varighet: 4 uker
- Mål: Implementere funksjonalitet basert på designet og kravspesifikasjonen fra første sprint.
- Oppgaver:
  - Evaluere og oppdatere kravspesifikasjonen.
  - Opprette database-tabeller og utvikle backend-klasser med tilhørende tester for:
    - Brukerregistrering, innlogging og lagring av brukerdata.
    - Strømprisoversikt.
    - GPS-funksjonalitet.
  - Bygge en front-end-prototype som viser appens utseende og funksjoner.
  - Implementere et system for varsler/notifikasjoner.
- Resultat: En fungerende prototype, og kode for grunnleggende funksjonalitet basert på kravspesifikasjonen.

## Sprint 3: Fullføring og dokumentasjon

- Varighet: 3 uker
- Mål: Fullføre back-end og front-end og ferdigstille dokumentasjonen.
- Oppgaver:
  - Forbedre funksjonalitet og opprette tester for klassene fra sprint 2.
  - Ferdigstille front-end elementene for en så komplett prototype som mulig.
  - Skrive innleveringsdokument og legge ved nødvendig informasjon.
- Resultat: En gjennomført prototype, back-end som oppfyller kravene fra kravspesifikasjonen, og en omfattende dokumentasjon.

## Erfaringer

Arbeidet med prosjektet har gitt oss verdifulle erfaringer med tanke på å kombinere smidig metodikk og effektiv sprintplanlegging får å utvikle et brukervennlig og funksjonelt system. Vi hadde god oversikt over frister i hver sprint, noe som førte til slutt at vi klarte å utvikle en prototype som oppfylte kravene i kravspesifikasjonen. Vi hadde effektiv kommunikasjon innad i gruppa som var viktig. Vi hadde møter ofte som vi brukte til å avklare mål, fremdrift og fordele oppgaver, noe som sikret at alle i gruppa hadde en klar forståelse av sitt ansvar. Vi brukte verktøy som Trello, som ga oss en oversikt over oppgaver og fremdrift i sprintene. Vi brukte også GitHub som sikret en strukturert og sikker måte å dele samt og gjennomgå kodeendringer på.

## Kodeprinsipper

I utviklingen av systemet fulgte vi noen definerte kodeprinsipper for å sikre at koden var strukturert, samtidig som det var vedlike holdbart.

- Modularitet: Systemet er bygd på moduler som hver har et klart ansvar. Alt fra strømprishåndtering, brukeradministrasjon, og GPS-funksjonalitet. Dette gjør koden lette å tolke, samtidig som det kan testes og videreutvikles uten at endringer i en modul påvirker andre.
- Lesbarhet: All kode er skrevet med beskrivende og klare navn for metoder, klasser og variabler. Dette er for å gi oversikt. Kommentarene i koden har i oppgave å forklare kompleks logikk eller spesifikke valg.
- Redundant kode: Vi har eliminert all redundant kode ved å abstrahere koder med felles funksjonalitet i egne metoder og klasser.

- Enkel – og robusthet: Funksjonene i systemet er bygget for å håndtere både uforventede og forventede situasjoner, som for ugyldig input. Dette sikrer at systemet er robust under ulike forhold.
- Testing: Vi har lagt mye vekt på testing i designet og skrevet tester for å validere at funksjonaliteten i systemet fungerer som forventet. Testene dekker enkeltstående moduler i tillegg til at det dekker integrasjonen mellom dem.

Ved hjelp av disse prinsippene har vi bygget kode i systemet som er robust, oversiktlig og enkel å videre utvikle samtidig som den gir verdi til brukere av systemet.

## Avhengigheter

I prosjektet er det brukt flere tredjepartsbiblioteker og verktøy for å få løsningen vår til å fungere slik vi ønsket. Vi har blant annet benyttet Firebase for push-varslinger og autentisering, som har vært et godt valg for å administrere brukerdata og varslingstjenester. Videre har vi benyttet flere biblioteker for å håndtere GPS-sporing og dynamisk prisberegning for strøm. Ettersom løsningen avhenger av strømdata og posisjonsdata, så vi oss nødt til å implementere disse.

Valget rundt disse avhengighetene var bevisst og gjør at løsningen også krever oppmerksomhet til fremtidig bruk, grunnet eventuelle endringer eller feil i disse verktøyene. Samtidig har det gjort løsningen dynamisk og svart på problemstillingen som vi sto ovenfor. Ettersom tjenestene er gratis i bruk, og åpen for alle, samsvarer de svært godt med våre mål om åpenhet og alminnelighet. I tillegg løser de kjernefunksjonaliteten til løsningen svært bra.

## Valg underveis

### Hvordan vi har jobbet

For å få en god oversikt over oppgavene som skulle utføres, valgte vi å bruke Trello. Dette ga oss mulighet til å fordele oppgaver på en effektiv måte, da vi kunne se hva som var planlagt å få fullført innenfor en gitt sprint, i tillegg til at vi kunne se hvem som holdt på med en spesifikk oppgave for å unngå at flere jobbet med samme ting. Trello var også nyttig for å gjøre sprintene mer transparente for hele teamet, noe som bidro til bedre kommunikasjon og samarbeid. For versjonskontroll og samarbeid på kodebenken benyttet vi GitHub, som også gjorde det enklere å holde styr på endringer og kodeansvarsområder.

### Oversikt over arbeidsfordeling og timebruk



Gjennom verktøy som Trello og GitHub har vi som sagt hatt god kommunikasjon og fordelt arbeidsoppgaver godt. For å få en full oversikt, har vi lagt ved timelistene våre i et Excel-dokument. I dokumentet vil man kunne se hva hvert medlem har gjort til enhver sprint. Vi har brukt git aktivt, og det er mulig å se på commit-historikk for mer detaljert informasjon. Trello vil også gi en bedre forståelse av hva som ble gjort periodevis til hver sprint.

## **Utfordringer underveis**

En av de største utfordringene i prosjektet var integreringen av eksterne tjenester som Firebase og GPS-sporing. Dette gjaldt spesielt håndteringen av API-nøkler og konfigurasjonen av disse. Feil i API-kallene førte til mye debugging, men disse utfordringene ble løst etter grundig testing og feilsøking.

# Instruksjoner for bruk

## Front-end

### Introduksjon til Vue og Quasar

Vue er et rammeverk ment for å bygge brukergrensesnitt og enkeltsideapplikasjon. Vue bygger på kodespråkene HTML, CSS og JavaScript. Ved å opprette et vue-prosjekt tvinges man til en viss filstruktur og visse regler ved filnavn og filreferering, men til gjengjeld får man muligheten til å koble opp mange eksterne tjenester. Eksempelvis har man muligheten til å koble til en tjeneste som heter Capacitor som gjør det mulig å sende prosjektet direkte til et program som heter Xcode, som gjør det mulig å teste en applikasjon på Apple-enheter.

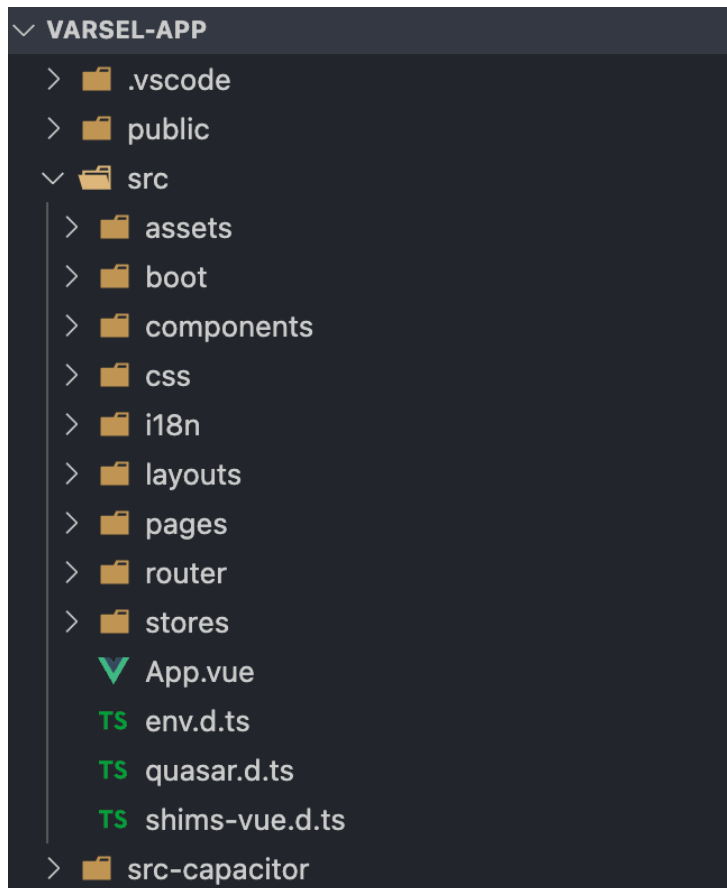
Quasar er bygget for Vue, og er i seg selv ikke et selvstendig verktøy eller kodespråk. Quasar er et rammeverk som forenkler utvikling ved å gi brukeren tilgang til ferdigdefinerte-klasser. Quasar kan brukes til flere ting, men hovedsakelig for utvikling av responsive web- og mobilapper. Quasar gjør derfor utviklingen av et nettsted enklere og mer effektivt.

### Filstruktur

«De interessante» filene i et Vue-prosjekt ligger i mappen *src* (source). Her vil man blant annet kunne se *pages*, *components* og *assets*.

- Pages refererer til selve sidene man kan navigere seg til på nettstedet.
- Components refererer til objekter som gjerne gjenbrukes, eksempelvis knapper.
- Assets referer til hjelpeobjekter eller eiendeler. Eksempelvis legges bilder og ikoner her. Filer som legges i assets skal ikke være nødvendige for at nettsiden skal kjøre, men oppleves ofte som en rød tråd, som gjør at brukeropplevelsen blir bedre.

Under er det lagt ved en figur som viser hvordan et typisk Vue prosjekt er satt opp. Figuren ekskluderer alle konfigurasjonsfiler, da disse genereres automatisk og ligger utenfor en mappestruktur:



## Kjøring og testing av front-end

For å bruke eller teste Front-End-kode som er laget med Vue og Quasar må man enten legge prosjektet på en server, som klarer å bygge nettsidene. Eventuelt må man installere nødvendige pakker og konfigurasjonsfiler lokalt. Prosessen er enklere på Mac ettersom det er bygget opp på en annen måte enn Windows. Vil man kunne kjøre Front-End på Windows, vil det være nødvendig med blant annet WSL (Windows Subsystem for Linux).

En full guide for installasjon av nødvendige pakker og filer ligger som en del av README filen i Develop-branchen på GitHub.

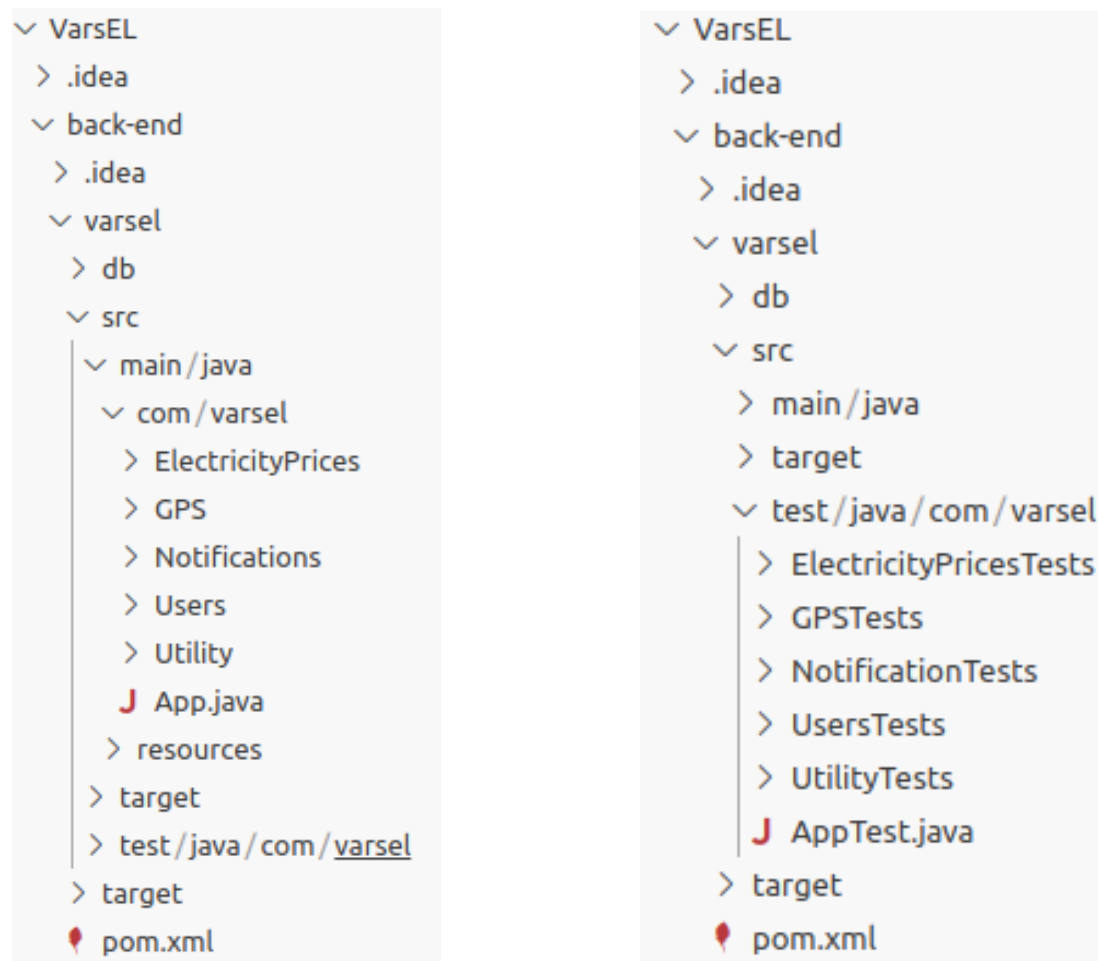
For testing og utvikling er det for enkelhetens skyld opprettet en Cloudflare server, som kjører nettstedet 24/7. Dette gjør at dere slipper å installere pakker og konfigurasjonsfiler for å se resultatet av koden. Lenken til denne finner dere under, men også i bunnen av dokumentet, sammen med andre relevante lenker.

- <https://varsel-app.pages.dev/auth/welcome>

## Back-end

Som back-end kode har vi brukt programmeringsspråket “Java”. Dette er et av verdens mest populære og brukte programmeringsspråk som kan kjøre på nærmest alle enheter. Grunnen til at vi valgte dette som vårt back-end-programmeringsspråk er fordi det kan brukes overalt, det er ikke så vanskelig å forstå, samt at det er enkelt å videreutvikle.

## Filstruktur



Over er det to bilder som representerer det viktigste i back-end filstrukturen. Bildet til venstre viser overordnet hvordan man navigerer seg frem til hovedkoden som man kan bruke til å bruke de forskjellige funksjonene som vi har laget. I bildet til venstre ser man hvordan man navigerer seg til funksjonstestene. “Pil ned” betyr at jeg allerede har trykket på mappen, slik at dere ser hva dere skal trykke på for å navigere dere til koden eller testene. I mappen “varsel” som du ser på com/varsel, ligger det 5 ulike mapper som hver inneholder kode knyttet til hva de gjør/handler om. I “App.java” har vi laget eksempler på hvordan man får

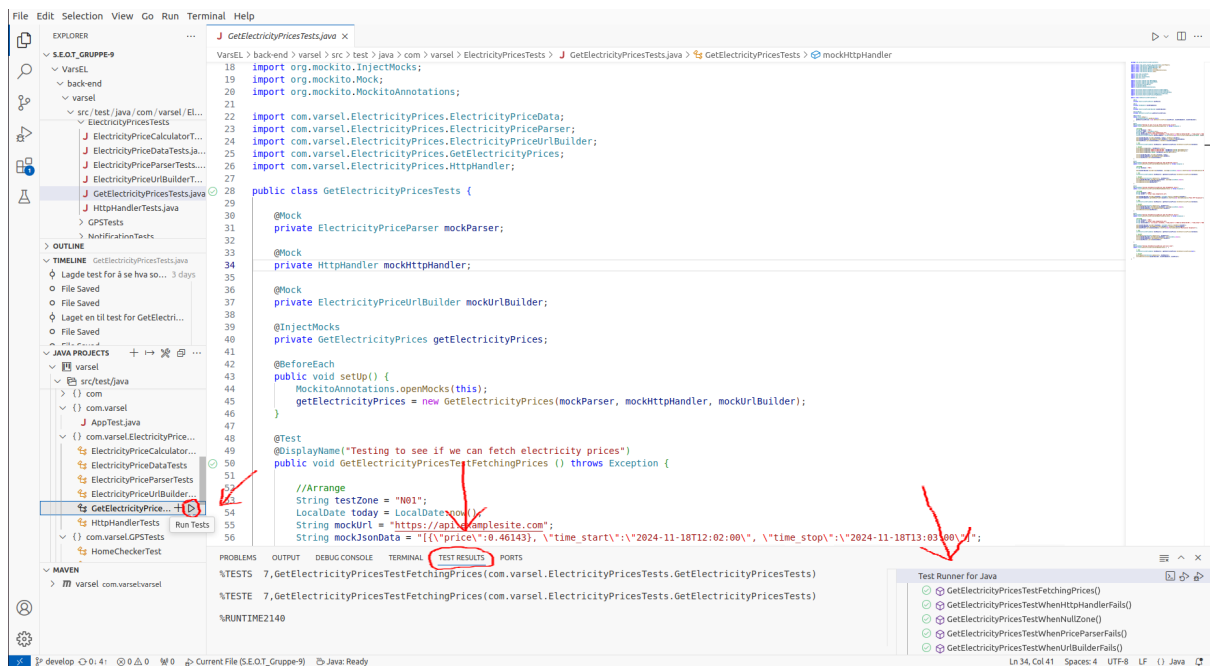
kjørt de viktigste programmene i mappene. I alle programmene, med unntak av App.java så er kommentarene skrevet på engelsk, slik at det er tilgjengelig for de aller fleste.

## Kjøring og testing av back-end

Under utviklingen, kjøring og testing av javakoden har vi hovedsakelig brukt programmet “Visual Studio Code”. For å bygge eller bruke koden som ligger i denne mappestrukturen så er den aller viktigste filen “**pom.xml**”. Hvis dere ser på bildet over så er den markert med en rød ballong. I denne filen ligger det kode som prosjektet trenger for å kunne kjøre programmene, uten denne filen vil funksjonene og testene ikke fungere. Mappestrukturen er laget med et verktøy som heter “Maven”.

For alle programmene i de fem mappene, har det blitt laget tester som undersøker at de viktigste delene av funksjonene fungerer som de skal. På mappestrukturbildet til høyre kan man se hvordan man navigerer seg frem til testene. Hver test har samme navn som hvilket program de tester. For å bruke testene for å se om programmene kjører som de skal, så trykker du deg inn på det programmet du ønsker å teste.

Merk at testprogrammene alltid slutter på “Tests”. Testene skiller seg fra hverandre med “@Test”, og under dette står det “@DisplayName”, hvor man ser hva de forskjellige testene tester. For å prøve å kjøre testene trykker du eksempelvis på et av programnavnene “GetElectricityPriceTests”, så ser du nederst på siden hvor det er et oppgavevindu. Trykk på “Test Results”, videre ser du på oppgavelinjen helt til venstre i Visual Studio Code at det står de forskjellige navnene på filene med tester i, trykk på ønsket program du vil teste og trykk på “Run tests”. På høyre siden av vinduet vil du se “Test runner for Java”, hvor man ser at testene blir utført. Hvis alt lyser grønt så fungerer testene som de skal. Hvis noen av de lyser rødt vil det si at en av testene ikke har fungert. Da kan man trykke på navnet som er rødt for å finne ut hvor feilen ligger. Under er et bilde som viser hva man skal trykke på.



## ElectricityPrices

I mappen “ElectricityPrices” ligger all funksjonalitet knyttet til strømpriser. Den inneholder kode som henter, prosesserer og analyserer strømprisdata. Den bygger også en URL som brukes til å hente strømpriser fra et API. I selve koden i hvert av programmene ligger det kommentarer som forklarer hva de forskjellige programmene gjør, samt hvordan man kjører dem. I App.Java kan dere teste de ulike funksjonene med et tekstbasert grensesnitt.

- ElectricityPriceCalculator analyserer strømprisdata og inneholder funksjoner som kan brukes for å beregne gjennomsnittsstrømprisen for et døgn, beregne laveste strømpris, samt høyeste strømpris.
- ElectricityPriceData kan brukes for å hente spesifikke strømpriser fra strømprisdata som har blitt hentet.
- ElectricityPriceParser tolker data fra API’et (Nettsiden vi henter strømpriser fra), og konverterer det til en liste som kan leses av systemet i riktig format.
- ElectricityPriceUrlBuilder bygger en URL som brukes for å hente strømpriser fra API-et.
- GetElectricityPrices er programmet som bruker programmene nevnt ovenfor sammen for å faktisk hente strømprisene.

Når GetElectricityPrices kjører så kommer det opp meldinger i terminalvinduet, nederst i Visual Studio Code som forteller om strømprisene har blitt hentet og hvilken nettside de er

hentet fra. Dersom det oppstår noe feil med henting av strømpriser, vil det dukke opp en melding som enkelt kan fortelle dere i hvilken av programmene det har skjedd en feil.

## Utility

Denne mappen inneholder verktøy for HTTP-forespørsler som blir brukt i prosjektet vårt for å hente strømpriser fra API-et.

- HttpClient er et grensesnitt som definerer hvordan HTTP-forespørlene skal bli behandlet av et annet program.
- HttpClient bruker grensesnittet til HttpClient for å hente data fra API-et.

## GPS

GPS funksjonaliteten i systemet er designet for at bruker kan håndtere posisjonsrelaterte oppgaver. Dette er alt fra at bruker kan sette ny hjemposisjon til å sjekke om bruker er i nærheten av sin egendefinerte “hjem” sone. Denne funksjonaliteten er sentral for å personalisere opplevelsen og ikke minst sikre nøyaktige beregninger basert på brukerens posisjon.

Systemet har tre hovedkomponenter for GPS-funksjonalitet:

1. “HomeChecker” - Dette er en klasse som sjekker om bruker er innenfor en forhåndsdefinert radius rundt et “hjem”. Ved hjelp av “Haversinne” formelen beregner klassen avstanden mellom brukerens nåværende posisjon og brukerens hjemmeposisjon. Dersom avstanden er mindre enn radiusen som er satt til (100 meter som standard), returnerer den at brukeren er hjemme. Den er også dynamisk og bruker kan oppdatere sin hjemme posisjon
2. "LocationService" - Er en klasse som både lagrer og håndterer brukerens nåværende posisjonsdata. Dette gjør den ved hjelp av å lagre både breddegrad (latitude) og lengdegrad (longitude) som representerer brukerens posisjon. Brukerens posisjon kan oppdateres gjennom metoder og nye koordinater kan hentes for videre bruk i systemet.
3. “MainActivity” - Denne klassen koordinerer interaksjonen mellom “HomeChecker” klassen og “LocationService” klassen. Det fungerer slik at den henter brukerens nåværende posisjon fra LocationService klassen og sjekker i HomeChcker klassen om bruker er i nærheten av hjemmet. Dette kan blant annet brukes i menygrensesnittet i App.java klassen hvor man kan se om bruker er innenfor hjem-sone.

4. “GPSMenu” - Er en enkel løsning hvor bruker lett kan navigere gjennom funksjonene ved et tekstbasert menygrensesnitt. Dette lar brukere velge handlinger som oppdatering av hjemposisjon eller sjekk nåværende posisjon mot hjemposisjon.

Gjennom menyen kan bruker:

1. Definere en ny hjemposisjon ved å oppgi ny bredde- og lengdegrad.
2. Sjekke om nåværende posisjon er innenfor definerte hjem-sonen (standard satt til: breddegrad (latitude) = 59.911 og lengdegrad (longitude) = 10.757).
3. Gå tilbake til hovedmeny.

GPS funksjonaliteten kan testes i App.java klassen ved å velge GPS (3) alternativet i hovedmenyen. Denne strukturen i systemet gjør det enkelt for brukere av systemet å utføre nødvendige GPS-relaterte oppgaver, samtidig som det gir en fleksibel løsning.

## Notifications

I dette systemet brukes Firebase Cloud Messaging (FCM) for å sende push-notifikasjoner til brukere, basert på strømpriser og batterinivået på registrerte elektriske kjøretøy. For å oppnå denne funksjonaliteten samarbeider fire hovedklasser for å generere og sende push-varsler til brukerne.

Firebase Cloud Messaging (FCM) er en plattform for sending av push-notifikasjoner til mobil- og nettapplikasjoner. FCM er en del av Googles utviklingsplattform, Firebase, og det er et pålitelig og skalerbart system for å sende meldinger til brukere på tvers av forskjellige enheter og plattformer. I dette systemet brukes FCM for å sende push-varsler, og noen av grunnene til at vi valgte FCM er:

- FCM er skalerbart og håndterer automatisk sending av meldinger til enheter, uavhengig av antall. Enten applikasjonen har få eller mange brukere, vil FCM automatisk tilpasse seg og levere notifikasjonene effektivt. Dette gjør det lettere å vedlikeholde funksjonaliteten over tid, ettersom økningen i antall brukere håndteres automatisk, uten behov for manuell administrasjon.
- FCM støtter de fleste kjente plattformer, inkludert Android og iOS. Dette gjør det mulig å sende notifikasjoner til brukere på tvers av forskjellige enheter, uavhengig av om de bruker en Android eller iOS-telefon.
- FCM er gratis å bruke, noe som gjør det til en kostnadseffektiv løsning for å sende push-notifikasjoner.



- FCM er lett å integrere med eksisterende applikasjoner, spesielt når Firebase allerede er i bruk. Siden Firebase også er integrert for innlogging i dette systemet, blir det enkelt å legge til for push-notifikasjoner.
- FCM sørger for pålitelig levering av meldinger, selv når enhetene er offline når meldingen først sendes. Hvis enheten er frakoblet, vil FCM automatisk forsøke å levere meldingen når enheten blir tilgjengelig igjen.
- En av FCMs kraftige funksjoner er muligheten til å sende målrettede meldinger. FCM lar utviklere segmentere brukere basert på atferd, interesser eller demografi, og sende personlige meldinger. Dette kan være nyttig for å sende spesifikke varsler til ulike grupper, for eksempel varsler om lave strømpriser til brukere i bestemte geografiske områder. Selv om dette er en svært nyttig funksjon, har det ikke vært en prioritet i utviklingen av vår MVP (Minimum Viable Product)

De fire Klassene som organiserer sending av push-notifikasjoner:

Den første klassen, `FirebaseInitializer`, har ansvaret for å initialisere Firebase-applikasjonen ved å bruke en tjenestekonto som spesifiseres i miljøvariabelen

`"GOOGLE_APPLICATION_CREDENTIALS"`. Tjenestekontoen inneholder sensitiv informasjon, som for eksempel autentiseringsnøkler og tilgangsrettigheter, og derfor lagres den som en miljøvariabel i stedet for som en API-nøkkel eller et dokument direkte i koden. Dette bidrar til å beskytte informasjonen og redusere sikkerhetsrisikoen. Denne klassen sørger for at Firebase kun initialiseres én gang, noe som forhindrer unødvendig redundans. Hvis det oppstår feil under initialisering, for eksempel at miljøvariabelen ikke er riktig satt eller det er problemer med tilgang til tjenestekontoen, vil relevante feilmeldinger bli logget for å hjelpe med feilsøking.

Ettersom tjenestekontoen settes til en miljøvariabel vil det være nødvendig å kjøre en kommando i terminalen hvor man setter denne miljøvariabelen til `service-account-file.json` filen som holder på informasjonen som trengs. Dette kan gjøres med denne kommandoen i terminalen:

```
set GOOGLE_APPLICATION_CREDENTIALS=C:\path\to\your\service-account-file.json
```

`service-account-file.json` ligger under resurser i back-end mappen.

Den andre klassen, Notifikasjon, er en enkel datamodell som representerer en push-notifikasjon. Den inneholder tre hovedattributter: token, som identifiserer enheten som skal motta notifikasjonen; title, som er tittelen på notifikasjonen; og body, som inneholder selve notifikasjons meldingen.

Den tredje klassen, NotifikasjonService, er ansvarlig for å generere notifikasjoner basert på spesifikke betingelser, som for eksempel lave eller høye strømpriser. Denne klassen inneholder metodene getStromprisNotifikasjon og getStromnivaNotifikasjon, som avgjør om en notifikasjon skal sendes avhengig av strømprisen eller batterinivået. Hvis de definerte betingelsene er oppfylt – for eksempel hvis strømprisen er under et visst nivå eller batterinivået er lavt – vil en passende notifikasjon genereres og være klar for utsendelse. Dette gjør det mulig for applikasjonen å levere relevante og tidsriktige varsler til brukeren.

Den fjerde klassen, PushNotifikasjonSender, håndterer selve prosessen med å sende notifikasjonene til Firebase. Denne klassen bruker Firebase Messaging API for å bygge og sende meldinger. Metoden sendNotifikasjonTilFirebase benyttes for å sende notifikasjoner når nødvendige betingelser er oppfylt. Klassen bruker enhetens token, tittelen og innholdet fra Notifikasjon-objektet til å sende meldingen via FCM. Dette muliggjør effektiv levering av push-notifikasjoner til brukeren, basert på de relevante betingelsene generert av NotifikasjonService.

Opprinnelig ble NotifikasjonService og PushNotifikasjonSender opprettet som en enkelt klasse med mye delt funksjonalitet, noe som kunne gi inntrykk av redundant kode. Det er derimot gunstig å fordele koden i to separate klasser, ettersom deres funksjonaliteter er forskjellige, og vi har som mål å utvikle et system som er både ryddig og lett å vedlikeholde. Dette følger grunnleggende kodeprinsipper som sier at en klasse bør ha ett klart og avgrenset ansvar for å unngå kompleksitet og rot. Dette gjør det også lettere å utvikle mer målrettede og detaljerte tester, samtidig som det blir enklere å opprettholde funksjonaliteten over tid, ettersom endringer kan gjøres i én klasse uten å påvirke den andre.

Koden for å teste notifikasjons funksjonaliteten er også implementert i App.java, men kjøring av notifikasjon testen vil ikke gi forventede resultater. Dette skyldes at funksjonaliteten for å hente gyldige FCM-tokens ennå ikke er implementert. Å implementere dette er veldig omfattende og tidkrevende, ettersom det involverer tett samarbeid mellom front-end og back-end. Siden hovedfokuset i utviklingen har vært å levere en MVP (Minimum Viable Product), har dette blitt nedprioritert i denne fasen av prosjektet.

## Users

Det er her funksjonene for registrering, innlogging og lagring av brukerinformasjon skal ligge.

Ved registrering av en ny bruker skal systemet be om en e-postadresse og et passord. Passordet skal deretter krypteres før det lagres. Dette sikrer at passordet ikke kan leses i klartekst, dersom noen skulle få tilgang til lagringsfilen.

Når brukeren ønsker å logge inn, skal systemet sammenligne e-postadressen og passordet som er oppgitt, med de krypterte dataene som ble lagret under registreringen. På denne måten kan systemet bekrefte om passordet matcher det som er lagret uten å eksponere passordet i klartekst.

Brukerinformasjonen skal lagres i en tekstfil. Hver bruker skal representeres med en linje som inneholder e-postadressen og den krypterte versjonen av passordet. Dette formatet gjør det enkelt å hente ut eller oppdatere brukerdata ved behov.

Systemet bør bygges opp av tre hoveddeler.

- User, som skal representere en bruker med e-postadresse og passord. Den skal inneholde nødvendige metoder for å håndtere disse dataene, som å hente e-post eller kryptert passord. Dette er kjernen i brukerdataene som skal lagres og brukes i systemet.
- UserRepository, som skal håndtere lagring og uthenting av brukere. Her skal funksjonaliteten for å skrive til og lese fra tekstfilen implementeres. Den må sørge for at data lagres i riktig format og gi mulighet for å hente ut brukere basert på e-postadresser.
- UserService, som skal inneholde logikken for registrering og innlogging av brukere. Det er her sjekkene for gyldige data, kryptering av passord og kommunikasjon med UserRepository skal implementeres. Målet er å sikre at brukere kan opprettes og autentiseres på en trygg og effektiv måte.

For testing og utvikling er det laget et enkelt tekstbasert brukergrensesnitt. Dette skal gjøre det lettere å teste funksjonene og danne seg et bilde av hvordan systemet skal fungere i et sluttprodukt. Dette brukergrensesnittet kan brukes til å simulere registrering, innlogging og håndtering av brukerinformasjon.

# Konklusjon og diskusjon rundt videre arbeid

## Konklusjon

Gjennom prosjektet har vi lagt til rette for en løsning som kombinerer flere nøkkelfunksjonaliteter for å gjøre hverdagen enklere for elbilbrukere. Hovedmålet har vært å lage en plattform som varsler brukeren om optimale ladetidspunkter basert på strømpris og geografisk plassering. Selv om løsningen ikke er fullverdig, er det lagt til rette for å fullføre løsningen. Dette har vi delvis oppnådd ved å:

- Implementere en responsiv app bygget med Vue og Quasar for fleksibilitet og brukervennlighet.
- Opprette et robust system for lokalisering ved hjelp av GPS-teknologi som kan fastslå om brukeren er hjemme.
- Utvikle en funksjonalitet for å sjekke strømprisene i sanntid og sammenligne dem med spådommer om fremtidige priser for å gi anbefalinger om lading.
- Integrere varsler som informerer brukeren om når det er mest økonomisk gunstig å lade elbilen.
- Leverer en back-end med tilkobling til eksterne API-er for strømpriser og geolokasjon, noe som gir en dynamisk og datadrevet løsning.
- Opprette en fleksibel og skalerbar arkitektur med en strukturert kodebase som gjør vedlikehold og videreutvikling effektivt.

Løsningen gir brukeren mulighet til å logge inn, og motta presise varsler basert på faktiske behov. Appen er designet for å være brukervennlig, pålitelig og enkel å forstå, noe som sikrer høy nytteverdi for målgruppen.

## Videre arbeid

Selv om vi har oppnådd mye, er ikke løsningen fullverdig, og det finnes flere muligheter for å forbedre og utvide den fullverdige løsningen. Under er det en liste med ting som bør gjøres for å fullføre løsningen, og mulige utvidelser for videre utvikling.

1. Koble sammen Front-End og Back-End.
  - For at løsningen skal bli fullverdig, er det avgjørende at man kobler sammen begge delene av prosjektet. Dette vil resultere i en fullverdig brukeropplevelse og dynamikk i det brukeren faktisk vil se og oppleve.
2. Fullføre Front-End
  - Foreløpig er Front-end delvis ferdig. Dette er fordi den ikke var like avgjørende i dette prosjektet, og ble hovedsakelig laget for å gi brukeren en følelse på hvordan en fullverdig løsning ville sett ut.
  - Her kreves det derfor at alle undersider implementeres, at man kan logge inn med valgfritt e-post domene, og ikke bare Google. I tillegg skal dataene om brukere lagres i en database.
3. Utvidelse av datakilder
  - I løsningen er det lagt opp til ett åpent API for å hente ut strømpriser, og annen relevant info. Her bør det vurderes å inngå samarbeid med strømleverandører for direkte tilgang til sanntidsdata. Husk fortsatt å beholde åpenheten til løsningen.
  - Hent eventuelt data fra flere API-er og finn ut av hvem som har en høyest suksessrate når det kommer til å forutse riktig, eller likende.
4. Utvidelse av funksjonalitet
  - Gi brukeren mulighet til å få en bedre oversikt over lademønstre og strømforbruk over tid.
  - Videreutvikle notifikasjonsfunksjonaliteten for å håndtere gyldige FCM-tokens og implementere nødvendig feilhåndtering.
5. Plattformuavhengighet
  - Sikre enda bedre støtte for mobile nettlesere og hybridapper for IOS og Android. Vurder eventuelt å lansere en fullverdig applikasjon.
6. Brukeropplevelse og tilbakemeldinger
  - For å sikre at løsningen fungerer optimalt og at brukeropplevelsen er optimal, bør det være mulig å rapportere inn feil, mangler og forbedringer.
  - Gjennomfør eventuelle brukerundersøkelser for å forstå hvordan appen oppleves og hvor forbedringer kan gjøres.

# Modeller og vedlegg

Modeller som er relevant for nåværende dokument, og videre forståelse og dokumentasjon av prototype og løsning finnes under «*Dokumentasjon/Modeller*» i ZIP-mappen eller på GitHub.

## Relevante lenker

Lenke til timeliste (ligger også vedlagt):

- [https://1drv.ms/x/c/3b274f0dbc59c8fd/ETfAYYa9NmVOvCu7i6Q7tkkBnrOtC0Ic96sK\\_Uz8y4KXqQ](https://1drv.ms/x/c/3b274f0dbc59c8fd/ETfAYYa9NmVOvCu7i6Q7tkkBnrOtC0Ic96sK_Uz8y4KXqQ)

Lenke til Trello:

- <https://trello.com/b/g7QFBDkP/itf20319-1-24h-software-engineer>
- <https://trello.com/invite/b/66d85ee6de102e85bc54c773/ATTIcdf6677519c556865d398f381dfd7bd800DEA557/itf20319-1-24h-software-engineer>

Lenke til GitHub:

- [https://github.com/EmilB04/S.E.O.T\\_Gruppe-9](https://github.com/EmilB04/S.E.O.T_Gruppe-9)

Lenke til Front-End:

- <https://varsel-app.pages.dev/auth/welcome>