**Prosjektinnlevering – Gruppe 9**

Emil Berglund

Andreas B. Olaussen

Khalid H. Osman

Sebastian W. Thomsen

Ida K. Tollaksen

Høgskolen i Østfold

Innholdsfortegnelse

[Introduksjon 4](#_Toc183045913)

[Problemstilling 5](#_Toc183045914)

[Løsningsbeskrivelse 6](#_Toc183045915)

[Kjernefunksjonalitet: 6](#_Toc183045916)

[Åpen og fleksibel platform 6](#_Toc183045917)

[Data og tilgang 6](#_Toc183045918)

[Ekspanderbarhet og antakelser 6](#_Toc183045919)

[Begrensninger 7](#_Toc183045920)

[Funksjonelle krav 9](#_Toc183045921)

[Identifisering av nødvendige funksjoner 9](#_Toc183045922)

[Funksjoner relatert til brukeren 10](#_Toc183045923)

[Gruppekrav basert på funksjonelle områder 12](#_Toc183045924)

[Forklaring av krav som dekkes i prototypen 12](#_Toc183045925)

[Ikke-funksjonelle krav: 14](#_Toc183045926)

[Brukere av systemet 15](#_Toc183045927)

[Systemarkitektur 19](#_Toc183045928)

[GitHub: 19](#_Toc183045929)

[Mappestruktur 20](#_Toc183045930)

[Kodestruktur 20](#_Toc183045931)

[Prototypedokumentasjon 22](#_Toc183045932)

[Testing 26](#_Toc183045933)

[Utviklingsprosess 29](#_Toc183045934)

[Sprints 29](#_Toc183045935)

[Erfaringer (diskusjon) 29](#_Toc183045936)

[Kodeprinsipper 29](#_Toc183045937)

[Avhengigheter 29](#_Toc183045938)

[Valg underveis (diskusjon) 29](#_Toc183045939)

[Instruksjoner for bruk: 30](#_Toc183045940)

[Front-end 30](#_Toc183045941)

[Introduksjon til Vue og Qusar: 30](#_Toc183045942)

[Filstruktur 30](#_Toc183045943)

[Kjøring og testing av front-end 31](#_Toc183045944)

[Back-end 32](#_Toc183045945)

[Electricity (nivå 3) 32](#_Toc183045946)

[GPS (nivå 3) 32](#_Toc183045947)

[Notifications 32](#_Toc183045948)

[Users 34](#_Toc183045949)

[Konklusjon og diskusjon rundt videre arbeid 36](#_Toc183045950)

[Konklusjon 36](#_Toc183045951)

[Videre arbeid 36](#_Toc183045952)

[Modeller og vedlegg 38](#_Toc183045953)

[Relevante lenker 38](#_Toc183045954)

# 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 ønsket 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 beskriver 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 strømaktører 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 platform

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 overkomplisere løsningen under utviklingen. Eksempler på fremtidige utvidelser kan derimot være som følger:

* Smart varsling: 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.

## Begrensninger

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

1. Begrenset tilgang til data.
   1. 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 API-er, som ikke vil være tilpasset brukerens valg av strømselskap.
   2. 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
   1. 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
   1. 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
   1. 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
   1. 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
   1. 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 skrur av funksjonaliteten.
7. Juridiske og regulatoriske barrierer
   1. 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 | Middles | 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 dashbord | 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.

1. **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. For eksempel: dersom prisen på strøm 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.

1. **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. **For eksempel:** kan 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.

1. **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 gjørs 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**: 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.

1. Dashbord 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:** Dashbord 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 dashbord og kan se detaljer om batteristatus og forbrukshistorikk. Videre beregnes ladeanbefalinger ut ifra strømprisen og bilens gjenværende batteri nivå.

**Eksempel fra applikasjonens prototype:**

* Dashbord: “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 | Firebase 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 u prototypen, men noen av funksjonene krever fremdeles videreutvikling. FK8 som handler om dashbord 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.

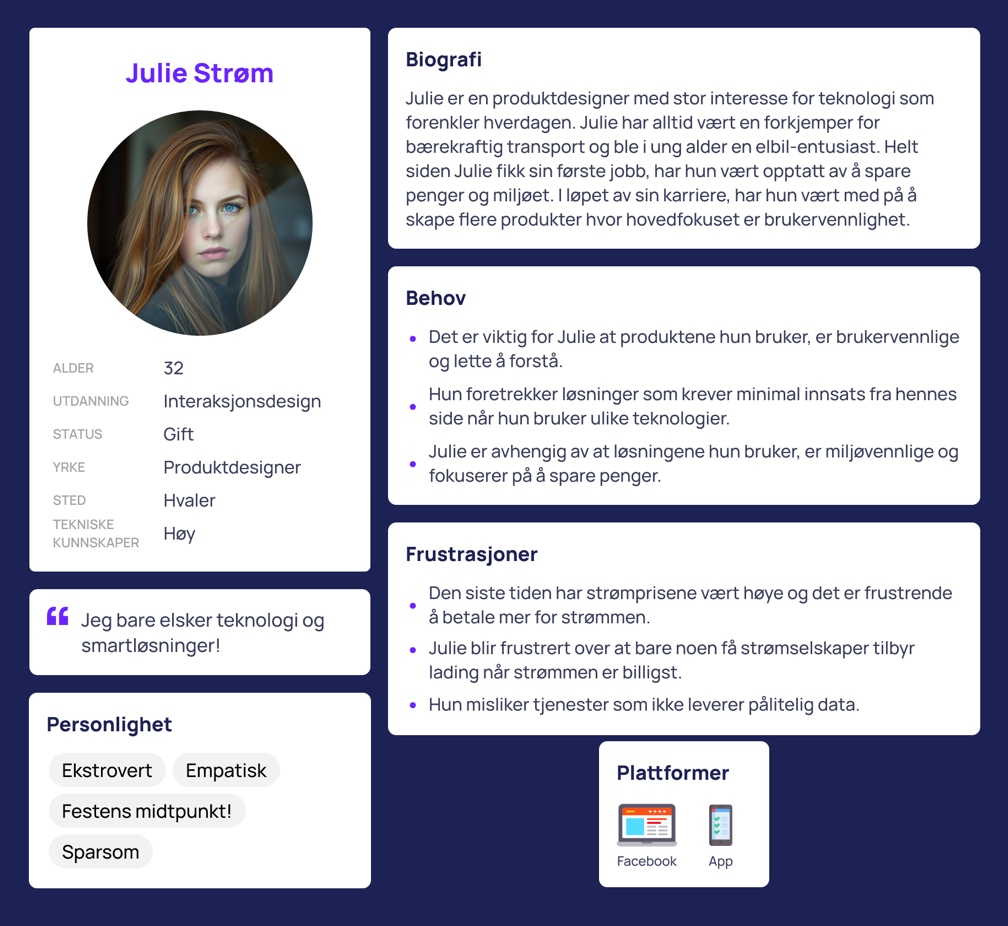
, sikkerhet, tilgjengelighet, og andre tekniske krav.

* Krav som ikke estimeres i nytte/kompleksitet.

# 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**

**Brukerhistorie 1: Lading når strømmen er billigst**

Som en miljøinterressert 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 konfigurerer appen til å starte og stoppe ladingen automatisk i dette tidsrommet, slik at hun kan fokusere på andre ting resten av dagen.

**Brukerhistorie 2: Oversiktlig informasjon**

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

**Scenario 2: Registrere bilen**

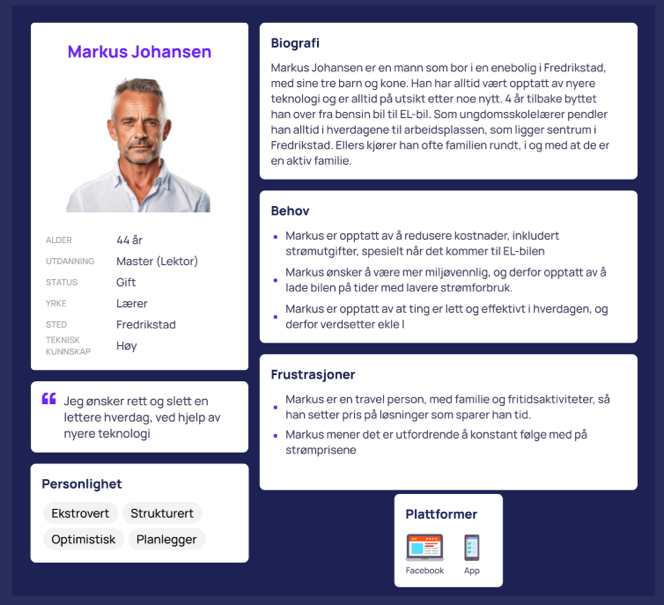
Julie har akkurat kjøpt seg ny elbil og klør i fingrene etter å registre den i VarsEL-appen sin. Hun åpner appen, navigerer seg til dashbordet, trykker på “Se alle kjøretøy” og legger til den nye bilen. På Dashbordet har hun nå en oversikt over bilen sin, batteristatus, høyest og lavest strømpris, samt 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. Siden bilen allerede er koblet til hjemmeladeren, spør appen om hun ønsker å starte ladingen. Med et raskt trykk på “Ja” i appen aktiverer hun ladingen og kan fortsette med middagsforberedelsene uten å bekymre seg.



**Markus Johansen**

**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 strømgrafen**

Markus har lunsjpause på jobben og tar frem VarsEL-appen. Han navigerer seg via dashbordet til strømprisgrafen og analyserer gårsdagens, 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 og om han ønsker å lade bilen. Markus trykker på “Ja” og fortsetter med det han skal gjøre resten av dagen.

# 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 forså i etterkant.
3. Pull requests: Opprett gjerne pull requests og ta i mot 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. I back-end bør man differensiere på ulike klasser, og avgrense tester fra klasser, slik at disse ikke blandes.

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.

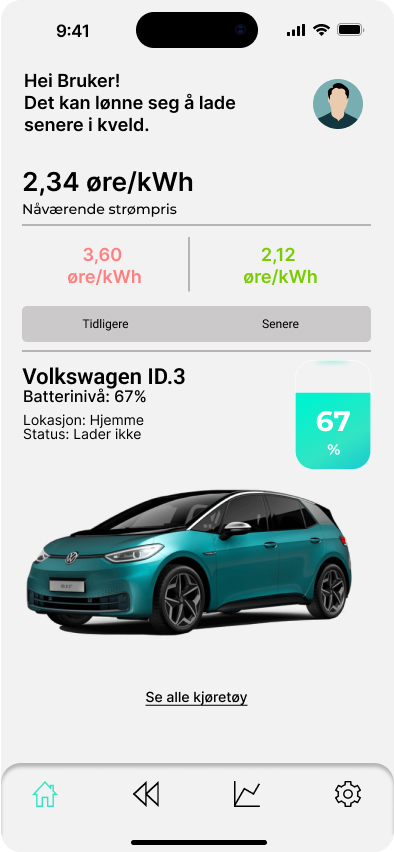
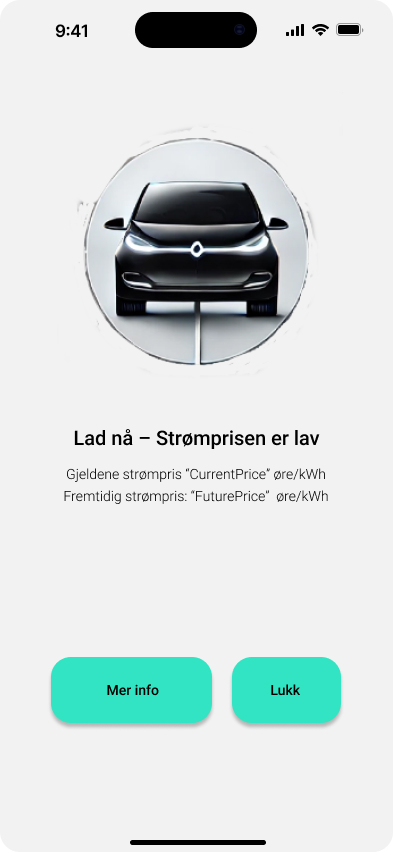
## Kodestruktur

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 personas, brukerhistorier og scenarior, 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 Jonatan!” 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 en gps-funksjonensom 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.

For det interaktive, visuelle brukergrensesnittet har vi brukt Vue.js

**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 potensiale 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, hashing 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 effiktivitet 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:
   * “HttpGetClientTests”:
     + 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.

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

## Erfaringer (diskusjon)

## Kodeprinsipper

* BackEnd

## Avhengigheter

## Valg underveis (diskusjon)

* Hvordan teamet har jobbet (smidig metodikk og sprintplaner).
* Bruk av Kanban eller lignende for oppgavesporing.
* Oversikt over arbeidsfordeling og timebruk.

# Instruksjoner for bruk:

## Front-end

### Introduksjon til Vue og Qusar:

Vue er et rammeverk ment for å bygge brukergrensitt 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-enhter.

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 vises hvordan et typisk Vue prosjekt er satt opp. Figuren ekskluderer alle konfigurasjonsfiler, da disse genereres automatisk og ligger utenfor en mappestruktur:

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse

### 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 en 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

### Electricity (nivå 3)

### GPS (nivå 3)

### Notifications

Funksjonalitet for å sende push-notifikasjoner til brukere via Firebase Cloud Messaging (FCM), basert på strømpriser og batterinivå på registrerte elektriske kjøretøy. Det består av fire hovedklasser som samarbeider om å generere og sende push varslingene til brukeren.

Firebase Cloud Messaging (FCM):

Firebase Cloud Messaging (FCM) er en plattform for sending av push-notifikasjoner til mobil- og nettapplikasjoner. FCM er en del av Firebase, Googles utviklingsplattform, 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 erskalerbart og håndterer automatisk sending av meldinger til enheter, uavhengig av antall. Dette gjør det svært gunstig for applikasjoner som må sende varsler til mange brukere samtidig. Enten applikasjonen har få eller mange brukere, vil FCM automatisk tilpasse seg og levere notifikasjonene effektivt.
* 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:

Den første klassen heter FirebaseInitializer og er ansvarlig for å initialisere Firebase-applikasjonen ved hjelp av en tjenestekonto som er spesifisert i miljøvariabelen med navn “GOOGLE\_APPLICATION\_CREDENTIALS”. Denne klassen sørger for at Firebase kun initialiseres én gang for å unngå redundant initialisering. Dersom det oppstår en feil under initialisering, for eksempel hvis miljøvariabelen ikke er satt riktig eller det er problemer med tilgang til tjenestekontoen, gis relevante feilmeldinger som kan hjelpe med feilsøking.

Den andre klassen Notifikasjon er en enkel datamodell som representerer en push-notifikasjon. Denne klassen inneholder tre hovedattributter: token, som er identifikatoren for en enhet som skal motta notifikasjonen; title, som er tittelen på notifikasjonen; og body, som er innholdet i notifikasjonen. Klassen inneholder også getter- og setter-metoder for disse attributtene, noe som gjør det enkelt å hente og endre verdiene etter behov. Dette gjør at objektet kan brukes fleksibelt i applikasjonen.

Den tredje klassen NotifikasjonService er ansvarlig for logikken som genererer notifikasjoner basert på spesifikke forhold som for eksempel lave eller høye strømpriser. Denne klassen inneholder metodene getStromprisNotifikasjon og getStromnivaNotifikasjon, som bestemmer om det skal sendes en notifikasjon basert på henholdsvis strømpris og batterinivå. Hvis de spesifiserte betingelsene er oppfylt (for eksempel at strømprisen er lav nok eller batterinivået er under et visst nivå), genereres en notifikasjon som deretter kan sendes til brukeren. Dette gjør at applikasjonen kan tilby relevante varsler til brukeren i sanntid. Den fjerde klassen PushNotifikasjonSender håndterer selve sendingen av notifikasjonene til Firebase.

Denne klassen bruker Firebase Messaging API til å bygge og sende meldinger. Metoden sendNotifikasjonTilFirebase brukes til å sende en notifikasjon dersom de nødvendige betingelsene er oppfylt. Klassen benytter enhetens token, tittelen, og innholdet fra Notifikasjon-objektet for å sende meldingen via FCM. Dette gjør det mulig å sende push-notifikasjoner direkte til brukeren, avhengig av de genererte betingelsene fra NotifikasjonService.

### 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 ladingstidspunkter 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.
* Levere en backend 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 front-end.
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, samtidig som man beholder 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.
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 fungere 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/ETfAYYa9NmVOvCu7i6Q7tkkBA8zO2acE7bhsSFaZRmYxpw>
* <https://1drv.ms/x/c/3b274f0dbc59c8fd/ETfAYYa9NmVOvCu7i6Q7tkkBnrOtC0Ic96sK_Uz8y4KXqQ> (Skrivebeskyttet - Lever denne)

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>

Lenke til Front-End:

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