Innlevering 3 – Gruppe 9

Emil Berglund

Andreas B. Olaussen

Khalid H. Osman

Sebastian W. Thomsen

Ida K. Tollaksen

Høgskolen i Østfold

Ord: 3500

Emnekode: ITF20319

Innhold

Introduksjon – Tankegang og arbeidsfordeling

I denne sprinten har vi gått videre i prosessen om å utvikle vårt produkt. Fra å videre-definere produktet og designe prototype i Sprint 1, har vi nå kommet godt i gang på både Front-End og Back-End delene av produktet vårt.

Fra å jobbe mye med det samme i forrige Sprint, har vi nå splittet oss mer, og fordelt individuelle arbeidsoppgaver. Vi ble tidlig i prosessen enig om at løsningen vår burde ta løsningen som en applikasjon. Hovedsakelig fordi dette er en effektiv måte å sende varslinger på, og i tillegg en veldig enkel og intuitiv måte å se varslinger på. Denne løsningen tillater også flere utvidelser senere i løpet, hvis man ønsker det.

Ettersom Back-End delen av produktet vektlegges mer i oppgaven vi har fått, ble dette hovedprioriteten. I denne sprinten har derfor 4 av 5 medlemmer jobbet med Back-End. Som Back-End språk valgte vi Java og SQL, hvor 3 medlemmer jobbet med Java, og 1 med SQL. Dette er fordi at alle gruppemedlemmene har hatt Programmering 2 og Databasesystemer, og fordi Ole-Edvard bruker Java i forelesning.

Front-End syntes vi derimot var prikken over i-en, og var en nødvendighet for å vise hvordan vi tenker oss en fullverdig løsning. Et gruppemedlem fikk derfor arbeidsoppgaven om å skrive Front-End kode, med hovedfokus på å få til navigasjon, og fullverdig ux. Funksjonalitet som innlogging og adaptiv kode, var lav prioritet. Som språk i Front-End valget vi Vue, med komponentsystemet Quasar. Hva Vue og Quasar er, bli bedre forklart senere i besvarelsen.

Ettersom vi også har fått tilbakemeldinger på tidligere innleveringer, følte vi det også var nødvendig å se til disse. Ett gruppemedlem fikk dermed i oppgave å se over og forbedre tidligere besvarelser, slik at disse holder standarden vi ønsker til endelig innlevering.

Front-End – Vue & Quasar

Som Front-End har vi valgt å bruke Vue og Quasar som er to helt nye kodespråk og verktøy som ingen av oss hadde vært borte i. Emil ble tipset om å bruke disse, og fikk dermed ansvaret for Front-End. Han har på egenhånd med litt opplæring fra en kompis lært seg både Vue og Quasar, for å bruke i denne oppgaven, men også for å ha kunnskapen til senere i arbeidslivet. For å ha noe å vise til det dere leser, ligger det på GitHub en video som viser og forklarer Front-End struktur og resultat. URL ligger under «Modeller og lenker».

Kort introduksjon til Vue og Quasar:

Vue er et allsidig rammeverk for å bygge brukergrensesnitt og enkeltsideapplikasjoner (*Vue.js*, 2014). Vue bygger mye på HTML, CSS og JavaScript hvor mye syntaxen er lik. 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 Vue-prosjektet gitt oss muligheten til å importere modulen «capacitor» som gir oss muligheten til å sende koden til Xcode hvor den oversettes til Swift, som er Apples kodespråk (*Capacitor*, 2024). Videre kan man laste ned en prototype på iOS enheter for videre bruk og testing.

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,* 2020). Quasar kan brukes til flere ting, men hovedsakelig for utvikling av responsive web- og mobilapper. I vårt tilfelle, har Quasar gjort det veldig effektivt å opprette elementer som knapper med funksjonalitet, men også for stilsetting med ferdigdefinerte klasser som «q-mt-md», som gir «margin-top» med «medium» avstand.

Fremgangsmåte og arbeidsteknikk:

Ettersom Vue og Quasar var nytt for meg (Emil) ble det en bratt læringskurve og mye arbeid for å i det hele tatt forstå kodespråket og rammeverket. Først og fremst måtte jeg initialere et Vue prosjekt og injisere Quasar rammeverket før jeg kunne begynne med kodingen.

Etter å ha opprettet et Vue prosjekt, med fungerende rammeverk kunne jeg begynne å kode. Fremgangsmåten min derifra var veldig enkel, men tok likevel lang tid grunnet mange

fremmedelementer. Mitt mål under utviklingen var å kode prosjektet så likt prototypen som mulig. Jeg begynte derfor med velkomstsiden og jobbet med videre, før jeg til slutt endte på dashbord siden, og undersidene som fulgte der. Mitt fokus lå i starten kun på design, i hvert fall før jeg ble komfortabel med Vue og Quasar. Videre i løpet lærte jeg meg mer, og gikk noen ganger tilbake og endret struktur.

En ting jeg la fokus på under utviklingen var hyppige commits og pushes til GitHub. Jeg syntes dette gjør det lettere i etterkant å se tilbake til, og ikke minst for andre å forstå. Jeg utførte derfor commits så ofte som mulig, som også ga meg muligheten til å gå tilbake til tidligere versjoner. Alle originale commits vil ligge under «ux-design» på GitHub, før jeg ved ferdigstillinger av sider, har merget med develop. Main har vi blitt enige om å kun bruke til Sprint innleveringer, dere vil derfor finne alt der også.

Tankegangen bak Vue og Quasar

Valget bak Vue og Quasar kommer mye av vennen min som jobber som utvikler, og bruker disse på dagligbasis. Som venn og utvikler anbefalte han disse to, både fordi det er veldig relevant til arbeidslivet, veldig lærerikt og ikke minst fordi de er gode midler for Front-End utvikling.

I tillegg til en sterk anbefaling, fant jeg fort ut av at Vue i seg selv ikke er veldig ulikt fra HTML og CSS noe som var veldig positivt. Som tidligere nevnt, gir Vue mange muligheter for utvidelser og andre komponenter. Fordelene rundt bruken av Vue ble derfor mer opplysende og er hvorfor jeg ikke byttet til noe annet.

Vue og Quasar var som tidligere påpekt helt nytt for meg, og jeg var i starten veldig bekymret for om valget jeg tok var noe jeg kom til å angre på. Dette var derimot ikke tilfelle. Vue og Quasar er veldig enkle i bruk, hvor Vue kan benyttes som ren HTML og CSS hvis man ønsker det. Quasar gir det detaljerte oppskrifter og forklaringer på alle komponentene sine, som gjør bruk og læring super enkelt.

Selv om jeg kanskje har gjort det tungvint for meg selv, ved å lære meg to helt nye ting, har det vært veldig lærerikt, og jeg ser bare fordeler ved å ha brukt disse i utviklingen av dette prosjektet.

Implementasjon av Back-End i Front-End:

Tanken videre, for det endelige produktet er å implementere Java-Back-End og SQL-Back-End sammen med Front-End. Hvordan dette gjøres, vet vi foreløpig ikke, og det har derfor blitt noen midlertidige løsninger for å illustrere hvordan løsningen skulle sett ut.

Slik Front-End fungerer ved innlevering, er det satt opp en Firebase som i samarbeid med google lar en bruker registrere seg, og logge inn. Ved å logge inn med google, får vi som utviklere tilgang til; fornavn, etternavn og e-post ved første øyekast. Dette kan videre brukes til brukerkontroll og brukerhåndtering. Slik det er nå, kan man derfor ikke registrere seg med tradisjonelle skjemaer. Her er det da tiltenkt at man skal kunne bruke disse skjemaene og sende det til SQL-databasen som tar vare på dette.

Hovedpunktet i appen er å varsle brukeren når den kommer hjem om det lønner seg å lade eller ikke. Slik det er nå, snakker ikke front-end koden sammen med Java koden, som henter denne informasjonen ved bruk av API. Tanken her, er å visualisere og hente nødvendig informasjon som Java får fra API. Dette kan hentes ved at Java skriver til en lesbar JSON fil, som deretter Front-End kan bruke og videreformidle brukeren.

Videre må vi ha en måte å varsle brukeren på, om betingelsene rundt posisjon og strømpriser er sanne. Det vi ønsker her, er at Java skal håndtere varslingen, og at Front-End skal implementere det. Java skal derfor sjekke strømprisen og brukerens posisjon og avgjøre hva brukeren bør gjøre, før resultatet kan leses av Front-End. Slik det er nå, er dette ikke koblet til Front-End.

Back-End – Java

Strømpriser

I dette prosjektet bruker vi blant annet Java som backend-kode. Appen vår skal ha et varslingssystem hvor man får en notification når strømprisen er lav og når det er gunstig å lade bilen. I forbindelse med dette har vi laget en javaklasse som heter GetElectricityPrices som henter strømprisdata fra et åpent API fra: https://www.hvakosterstrommen.no/strompris-api.

Man kan hente strømpriser fra ulike soner i Norge, samt i koden bestemme hvilken dag og hvilke timer man ønsker å hente strømpriser fra. Klassen kobler seg til API ved bruk av HTTP-forespørsel, behandler strømprisene og returnerer prisene i en liste med 24 elementer, altså strømprisen for den valgte dagen for hver time av døgnet som er spesifisert i gitt tidssone. I klassen blir det brukt en "Logger" for å logge informasjon og feilmeldinger. API-responsen blir lest linje for linje og samlet i en StringBuilder og blir tolket som JSON med JSONArray. All koden ligger i en try/except blokk.

I tillegg til selve koden for å hente strømpriser har jeg laget en egen klasse som heter ElectricityPriceCalculator som skal ta inn verdiene fra listen og utfører tre forskjellige funksjoner i tre forskjellige metoder. Klassen kan regne ut gjennomsnittsprisen, laveste strømpris og høyeste strømpris.

Videre har jeg laget flere klasser. En som heter ElectricityPriceData som henter all informasjon fra listen som pris, tidsstart, tidsslutt og prissone. Planen er at denne klassen skal bli brukt sammen med metodene nevnt ovenfor for å kalkulere og returnere gjennomsnittspris, høyest- og lavestpris til appen. Den kan også i fremtiden sende all data om strømpris for et døgn.

Jeg har også startet med å refaktorere koden til GetElectricityPrices-klassen ved å lage en egen klasse for å parse JSON, samt en for HTTP-Forespørsler. Grunnen til dette er for å dele opp koden i mindre deler og flere metoder for å kunne teste koden med Mockito på en bedre måte, samt at det blir lettere med feilsøking og å kunne bruke klassene til muligens andre programmer i fremtiden.

Bibliotekene som blir brukt i koden min er: java.io.BufferedReader, java.io.InputStreamReader, java.net.HttpURLConnection, java.net.URL, java.util.ArrayList, java.util.List, org.json.JSONArray, org.json.JSONObject, org.slf4j.Logger og

org.slf4j.LoggerFactory.

BufferedReader blir brukt til å lese tekst linje fra linje fra inputStreamen som jeg får brukt på grunn av InputStreamReader. HttpURLConnection blir brukt for å lage en HTTP-tilkobling til API-et som jeg henter strømprisene fra. Java.net.URL blir brukt for bygge URL og opprette en forbindelse med serveren. ArrayList og List blir brukt for å lagre strømpriser i en liste. JSONArray og JSONObject brukes for å behandle JSON-data som kommer via API-et og lagrer det som objekter i en liste. Logger og LoggerFactory brukes for å lagge antall objekter som blir hentet via API-et, samt logger hvilken URL som har blitt brukt og om det dukker opp noen feilmeldinger.

Avhengigheter som jeg har brukt og plassert i en .pomfil er: org.jason som jeg trenger for å bruke JSONArray og JSONObject i forbindelse med å hente og bruke JSON-data fra API-et. Jeg bruker også org.slf4j og ch.qos.logback for å logge informasjon under kjøring av programmet.

Under utvikling av koden pusher jeg og commiter endringene ofte til GitHub-repoet for å dokumentere arbeidet som har blitt gjort. Koden min skal bli løst koblet kode slik at de fleste komponentene skal være uavhengige av hverandre, enkle å vedlikeholde, samt lettere å utvide.

GPS

Vi skal lage en GPS-funksjonalitet i vår applikasjon som registrerer brukerens posisjon og sjekker når brukeren ankommer hjemmet. Etter at systemet har registrerer at brukeren har ankommet sin definerte hjemme-sone, skal applikasjonen sende informasjon om strømprisinformasjon. Formålet med denne funksjonaliteten er å gi brukeren konktekstbasert informasjon direkte relatert til lading, uten at brukeren manuelt trenger å kontrollere det selv.

I utviklingen av denne funksjonaliteten er planen å legge til en metode som baserer seg på å opprette en sone virtuell sone rundt personens hjem. Derfor må vi definere et geofence - område, ved hjelp av et sett med GPS – koordinatorer, der systemet registrer brukerens bevegelse inn i sonen. Vi tenker derfor å bruke geofence-API-et sammen med Google Location API-et, som skal gi oss nøyaktig posisjonssporing.

Konto-håndtering

Vi har også laget klasser og metoder som gjør at brukere kan opprette konto og logge inn. Når man registrerer seg gjør man dette med e-post og passord, disse verdiene blir da lagret på et tekstdokument som man senere henter brukerinformasjon fra når brukeren forsøker å logge inn. Passordet blir kryptert ved hjelp av BCrypt i tillegg til at det får et tilfeldig generert salt slik at passordene ikke ligger leselige i filen. Planen videre er å gjøre om slik at brukerinformasjonen heller blir lagret på en database, men så langt har vi ikke kommet enda.

Når brukeren forsøker å logge inn, sjekkes det om passordet brukeren skriver inn matcher det krypterte passordet som er lagret med den e-posten som brukeren ønsker å logge inn med.

Det er laget tester som sjekker om koden fungerer på den måten som er forventet, dersom man skulle forsøke å opprette en konto med en e-post som allerede finnes i filen, logge inn med en email som ikke finnes og logge inn med et passord som ikke er likt det lagrede passordet. Den tester også om det fungerer dersom registrering og innlogging er suksessfull.

Planen videre i prosjektet er å implementere arkitektur slik at det vil være lettere i fremtiden å utvide og teste koden, i tillegg til å få koblet koden opp mot en database for bedre lagring av data.

Varslinger

I første del av utviklingen av notifikasjoner begynte jeg med å undersøke hvordan man implementerer push-notifikasjoner i Java. Jeg fant ut at det først og fremst er viktig å spørre brukeren om tillatelse til å sende push-notifikasjoner. Siden dette er en varslingsapp, er dette et svært viktig kriterium. Informasjonen om brukeren tillater eller ikke må også lagres, slik at de ikke blir spurt hver gang de åpner appen.

Videre satte jeg opp et Firebase-prosjekt og skrev kode for å initialisere Firebase. Jeg opprettet også klassene Notifikasjon og PushNotifikasjonSender for å kunne opprette og håndtere objekter av disse klassene. Deretter utviklet jeg tester for Notifikasjon og PushNotifikasjonSender for å sikre at disse funksjonene fungerer som de skal. Testene er relativt enkle og trenger videreutvikling, men gir for øyeblikket korrekte resultater. Det er også skrevet noe enkel kode i main, men denne delen er ikke helt oppe å gå ennå.

Hovedfokuset i denne sprinten har vært den grunnleggende koden og oppretelsene og utføringen av testene.

I tillegg gjorde jeg små justeringer i koden underveis som oppdatering av filstruktur og tilføying av detaljerte kommenterer foran funksjoner. Dette ble gjort for å gjøre vedlikehold og lesing av koden enklere og mer intuitiv.

Database - SQL

Vi har også tenkt å etablere en database med flere sentrale tabeller, for å sikre en presis håndtering av brukere og strømdata i prosjektet. Tabellene skal inneholde essensiell informasjon om strømpriser og brukere.

Tabellen “Brukere” skal inneholde relevant informasjon om hver bruker. Alle rader skall ha unike ID-er for gjenkjenning av unike brukere. Den skal også inneholde informasjon, som fornavn, etternavn, e-post og brukerens posisjon. Posisjons felt er svært viktig i og med at brukerens plassering skal sammenlignes med en definert geofence – sone rundt brukerens hjem, slik at appen kan sende varslinger når brukeren er innenfor den sonen.

“Strømpriser” tabellen lagrer både historisk og nåværende informasjon om strømpriser. Strømpriser tabellen skal brukerne en oversikt over prisutvikling. Formålet med denne tabellen er å gi brukere en oversikt over prisutvikling. Viktig data som lagres i denne tabellen er data som “år”, “måned” og “dag”. Det skal gjøre det mulig å hente informasjon om bestemte dager og tider på døgnet. Tabellen har også et felt for pris soner (“price_zone”), som viser til prisen i norske kroner (NO1 – NO5). Vi har også feltet “nok_per_kwh”, som skal spesifisere prisen for strøm i korner per kWh. Til slutt har vi Start- og sluttidspunkt feltet, hvor hver prissone lagres som tidsstempler. Dette gjør vi for å sikre korrekt pris ved hvert tidspunkt

Implementeringen av Databasen vil spille en stor rolle i å håndtere informasjon på en sikker og nøyaktig måte. Ved hjelp av en velstrukturert database kan vi tilby personaliserte varslinger til de forskjellige brukerne, som gir brukerne innsikt i både nåværende og historiske strømpriser.

Retrospekt fra forrige sprint

Gruppearbeid

Sammenliknet med forrige sprint, har arbeidsoppgavene vært mer spredt og tilliten til hvert gruppemedlem har vært viktigere enn noen gang. I denne sprinten har hvert gruppemedlem måtte jobbe med hver sin oppgave, hvor oppgavene sammen skulle utgjøre en større løsning.

I denne sprinten har vi ikke vært like flinke på å møtes, som i den forrige. Mye av dette kommer av klare mål og gode arbeidsfordelinger. Vi har muligens følt at dette ikke var like nødvendig, ettersom vi tidlig fordelte oppgaver og satt krav til hva hvert medlem skulle oppnå med sin del av oppgaven.

Vi har hatt oppsummeringsmøter, altså møter hvor vi har fortalt hva vi har gjort, hva vi har oppnådd, og ikke minst hva planen er videre. Disse møtene har gitt alle innblikk i hva alle har gjort til enhver tid, og ikke minst gitt rom for kommentarer, forslag eller forbedringer fra andre i gruppen

Ettersom hvert gruppemedlem fikk hver sin oppgave, har vi tilegnet oss veldig mye ny kunnskap og forskjellig kunnskap. Noe negativt ved å ta denne tilnærmingen, er at alle i gruppa får ikke vært innom alt. Det betyr egentlig at noen medlemmer lærer mer enn andre, og læringen blir veldig entydig. Vi kunne sikkert løst arbeidsfordelingen på mange måter, men vi følte i øyeblikket at dette var den mest effektive måten gjør det på. Som gruppe derimot, har vi lært veldig mye nytt. Mye av dette kommer av at vi ble tvungne til å lære oss nye ting, for å oppnå det vi ønsket.

Endringer av tidligere dokumenter

Etter første utkast av tidligere dokumenter mottok vi en rekke ting vi kunne forbedre på for å gjøre dokumentet mer komplett og strukturert. Vi måtte blant annet legge til kapittelnummering, slik at sensor enkelt kan navigere og forstå innholdet. Vi manglet også to brukerhistorier som vi måtte legge til basert på scenarioet. Til slutt måtte vi legge til litt mer spesifikke situasjoner i scenarioet.

Ut ifra tilbakemeldingene vi fikk på scenarioet så vi at vi manglet en del detaljer, som kunne gi et mer realistisk bilde i hvordan appen vår ville fungere i praksis. Etter endringer beskriver

scenarioet nå bedre hvordan en typisk bruker ville brukt produktet. Endringene vil la til fokuserte på å lage et bilde av «Hvordan brukeren bruker systemet når han kommer hjem etter en lang dag på jobb» og «Hvordan systemet hjelper brukeren med å ta smarte ladebeslutninger uten å måtte bruke tid på å sjekke priser selv». Disse endringene gir leseren en bedre forståelse av produktet.

Den andre tilbakemeldingen vi fikk gikk ut på å legge til kapittelnummering. Dette skulle bidra til å gjøre prosjektet mer oversiktlig. Derfor la vi til en innholdsfortegnelse. Dette gjør det rett og slett lettere for leser å finne fram til de ulike delene av dokumentet.

Den siste endringen vi gjorde var å legge til brukerhistorie. Dette var noe vi ikke hadde i det første utkastet. Vi valgte derfor å legge til to spesifikke brukerhistorier for å illustrere hvordan og hvorfor bruker ville brukt produktet. Begge disse bruker historiene er relevante til det vi allerede har og handler om de brukerne vi brukte i både Personasene og scenarioene.

Oppsummering av fremtidig arbeid:

Front-End

Fremover er målet å bli ferdig med alle sidene som er mulig å navigere seg til. Dette gjelder funksjonalitet og design. Målet er også å klare å koble sammen Front-End og Back-End.

Kontohåndtering

Planen videre i prosjektet er å implementere arkitektur slik at det vil være lettere i fremtiden å utvide og teste koden, i tillegg til å få koblet koden opp mot en database for bedre lagring av data.

Strømdata

Planen videre med Java-koden vil være å bli ferdig med refaktoreringen, skrive ordentlig kommentarer om klassene i henhold til kravene vi har lært om i forelesning, samt lage Mockito-tester for å teste all funksjonalitet til koden. Det skal også bli mulig å sende data om strømpriser til en database. I neste sprint blir fokuset mitt å forberede arkitekturen og gjøre koden min testbar, samt få til en bedre mappestruktur.

Varslinger

Planen for neste sprint blir å videreutvikle testene for å sende en varsling til å omhandle flere senarioer og mer spesifikt rettet mot de varslingene vi tenker at appen skal sende.

Modeller, lenker og struktur – GitHub

Lenke til GitHub:

https://github.com/EmilB04/S.E.O.T_Gruppe-9/tree/main

Filsti for assets

- Video av Front-End: «S.E.O.T_Gruppe9/Sprint_oversikt/Videoer»
- Lenker til Figma og Trello: «S.E.O.T_Gruppe9/Sprint_oversikt/Lenker»
- Kopi av Figma: «S.E.O.T_Gruppe9/Sprint_oversikt/Bilder»

Branches på GitHub

For enkelhetens skyld ligger de endelige resultatene for denne Sprinten i «Main»-branchen på GitHub. Vil dere se individuelle commits, og fremgang, kan dere gå inn på hver enkelt branch og se. Branchene vil bli endret etter innlevering ettersom det er mange som ikke brukes, og vi har planer om å slå sammen mye av Java-testingen til samme branch, men heller differensiere med mappestruktur.

- Front-End vil ligge under «ux-design»
- Brukerhåndtering vil ligge under «Java-Testing»
- Strømpriser vil ligge under «current-electricity-price»
- Varslinger vil ligge under «notifications»

Litteratur

Vue.js. (2014). Vuejs.org. https://vuejs.org/

Quasar Framework - Build high-performance VueJS user interfaces in record time. (2020). Quasar Framework. https://quasar.dev/

Capacitor by Ionic - Cross-platform apps with web technology. (2024). Capacitor. https://capacitorjs.com/

Strømpris API - Åpent og gratis. (2024). Hvakosterstrommen.no. https://www.hvakosterstrommen.no/strompris-api