# **Avhengigheter**

## Eksterne avhengigheter

I vår applikasjon, har vi også valgt å lage en «fake» for databasen, klassen kalt ListRepository. I realiteten hadde denne klassen egentlig hatt en eller annen form for kommunikasjon med en database gjennom en server, hvor fra den kan hente, sende og endre data til og fra. Dette eksterne systemet vil da ha ansvar for å oppbevare de forskjellige dataene om brukere, parkeringsplasser, reservasjoner og hvor mye hver bruker må betale. Serveren vil også ha del ansvar i å sørge for at brukere ikke kan be om data som de ikke bør se.

Når det gjelder betalinger trengs det et eksternt betalingssystem som kommuniserer med applikasjonen, når bruker må betale for leing av parkeringsplass. Det vil altså si dette eksterne system har ansvar for overføring av penger mellom mange forskjellige kontoer. Eksempel på et slikt system kunne være Vipps eller Klarna.

## Interne avhengigheter

Kivy vår framework for å lage det grafiske utseende. Kivy har i tillegg noen avhengigheter som også må lastes ned før Kivy kan brukes. Dette er kivy-dep.angle, kivy-deps.glew, kivy-deps.sdl2. Disse 3 må være nedlastet i miljøet for at Kivy skal fungerer korrekt. I tillegg brukes det også 2(3 inklusive pytest-mock om den fortsatt skal brukes) 1 framework for testing kalt pytest. Tester er avhengig av dette framework at testene skal kunne kjøre korrekt. Noen tester har vi også vært nødt til å test om funksjoner blir aktivert i det hele tatt, til dette har vi brukt biblioteket mock. Noen tester blir da også avhengig av dette bibliotek, for å kunne kjøre. (Bør også si noe om Python versjoner her!!!). #mer om freezegun.

Alle disse blir installert automatisk i miljøet ditt med setup.py filen. Les mer om oppsett av miljø og setup.py i *kom i gang med systemet* seksjonen.

avgrenser hva systemet skal ha ansvar for?

# **Beskrivelse av systemet**

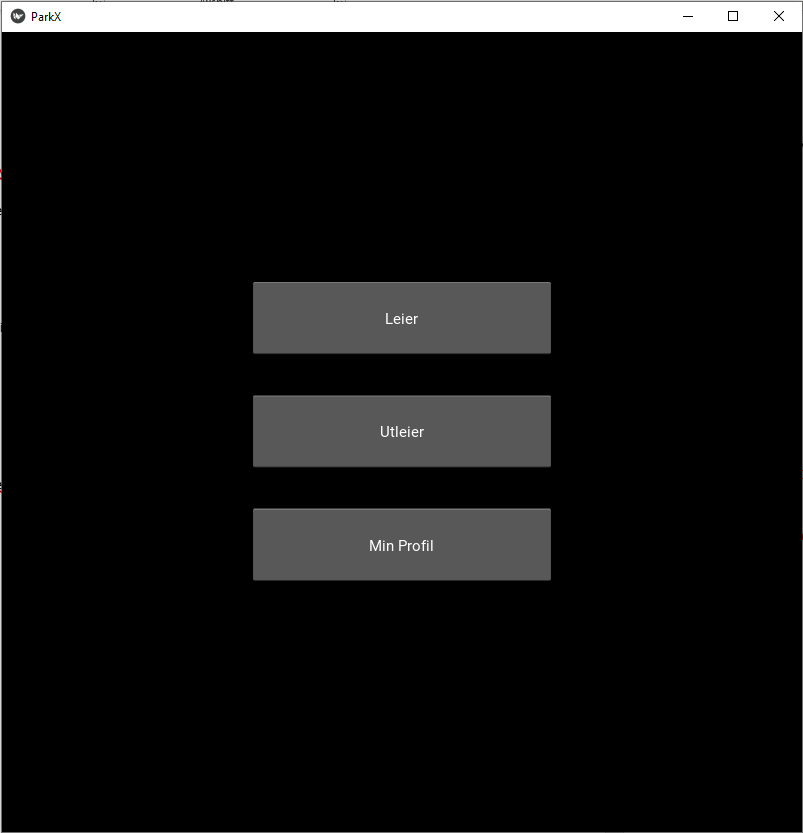
ParkX består av flere individuelle deler kalt klasser som kommunisere med hverandre. Hver del har ansvaret for en spesiell del av det systemet gjør. Sammen sørger alle klassene for å lage et større system som tilfredsstiller flere krav.

## Prototypens komponenter

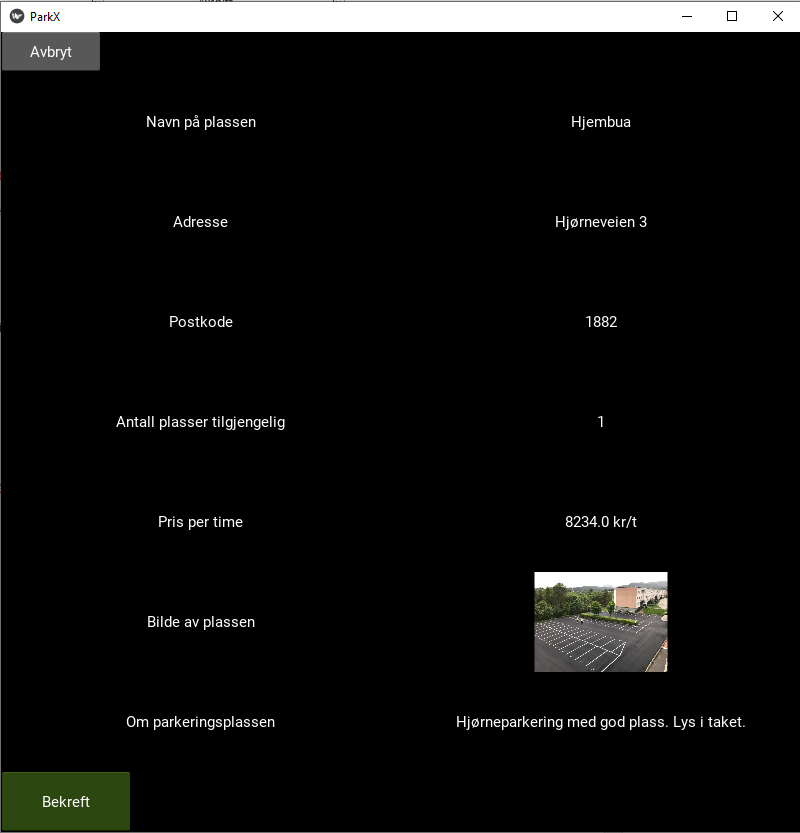
Parkingplace er klassen som holder styr på lagring av den data som hører til hvert enkle parkeringsplass. Det vil altså si Parkingplace eksempelvis: holder styr på data som adresse, pris\_per\_time, antall og om parkeringsplassen er i bruk. Hver parkeringsplass har også en id som brukes, hyppig i bruk av andre klasser for å få takk i den riktige parkeringsplass. I tillegg holder også Parkingplace funksjonalitet til å manipulere denne data. Eksempelvis, kan Parkingplace oppdaterer sin egen status til si om parkering er blitt startet av en annen bruker, eller sagt på en annen måte om parkeringsplassen er i bruk. I tillegg til å endre statusen til motsatt vei, altså at parkeringsplassen ikke lengere er i bruk. Hovedjobben til Parkingplace er holde den individuelle data for hver parkeringsplass, som enkelt kan trekkes ut og brukes av andre klasser.

ListRepository er en klasse som tar seg av jobben med å behandle alle parkeringsplasser. Klassen har en liste som skal holde alle parkeringsplasser som skal vises for brukeren. ListRepository holder i tillegg en del funksjoner for å operere på denne listen som holder alle parkeringsplasser. Her kommer bl.a. sletting, endring og laging av nye parkeringsplasser. Alle disse tar bruk av parkeringsplassens id for å kunne, eksempelvis: slette den riktige parkeringsplass. Når parkeringsplasser lages er det viktig og noterer seg at informasjonen som kommer inn for å kunne lage en parkeringsplass kommer i form av en dictionary. Dictionary er en samling av data. I tillegg finnes det også en del funksjonalitet for å trekke ut en eller hele listen med parkeringsplasser. For å trekke ut en spesifikk brukes også id til parkeringsplassen. ListRepository er en fake database, siden vi ikke kan koble en riktig database til prototypen. ListRepository hadde derfor i en mer realistisk implementasjon ikke holdt funksjonalitet for å endre direkte på parkeringsplasser, men hellere holdt funksjonalitet for å kontakte en ekstern database, også fortelle hvordan den skal gi eller endre data. Hovedjobben til ListRepository er og oppfører seg som en fake database, som kan hente og opererer på parkeringsplasser.

Gui er vår frontend klasse. Klassen extender BoxLayout som er en klasse fra avhengigheten Kivy. Denne hjelper med å enkelt kunne lage et grafisk user interface for brukeren. Gui klassen består hovedsakelig av funksjoner som legger inn elementer som knapper, bilder og tekst i vinduet. Når bruker klikker på en knapp, blir andre funksjoner i Gui aktivert, disse er kalt handlers. Det kan skje flere forskjellige ting når en bruker klikker på en knapp. Noen knapper som eksempelvis i startsiden av programmet vil, få vinduet til å bytte side(figur1). Dette gjøres ved å fjerne elementer som er på siden og bytte dem ut med nye. Når nye sider må bygges opp basert på parkeringsplassen eller mange parkeringsplasser, vil Gui aktiverer funksjoner i klassen kalt ParkingController (mer om denne under). Disse funksjoner vil ofte aktiverer andre funksjoner i ListRepository som gjør at ParkingController får en parkeringsplass eller en liste med parkeringsplasser. Deretter vil ParkingController returnerer disse til Gui, som da kan bygge opp siden utfra denne data. Andre knapper som «Bekreft» knappen som ses på siden hvor man bekrefter at man vil leie parkeringen(figur2). Vil i tillegg til å bytte side også aktiverer funksjoner i ParkingController som forteller ListRepository hvordan den eksempelvis skal endre en parkeringsplass (mer om hvordan funksjonene henger sammen i hvordan systemet henger sammen). Gui klassens hovedoppgave er og presenterer informasjonen om parkeringsplasser til brukeren, men skal også fungerer som startpunktet for brukerens input.



Figur 1:bilde av start siden til ParkX



Figur 2: bilde av siden etter klikket på lei parkering

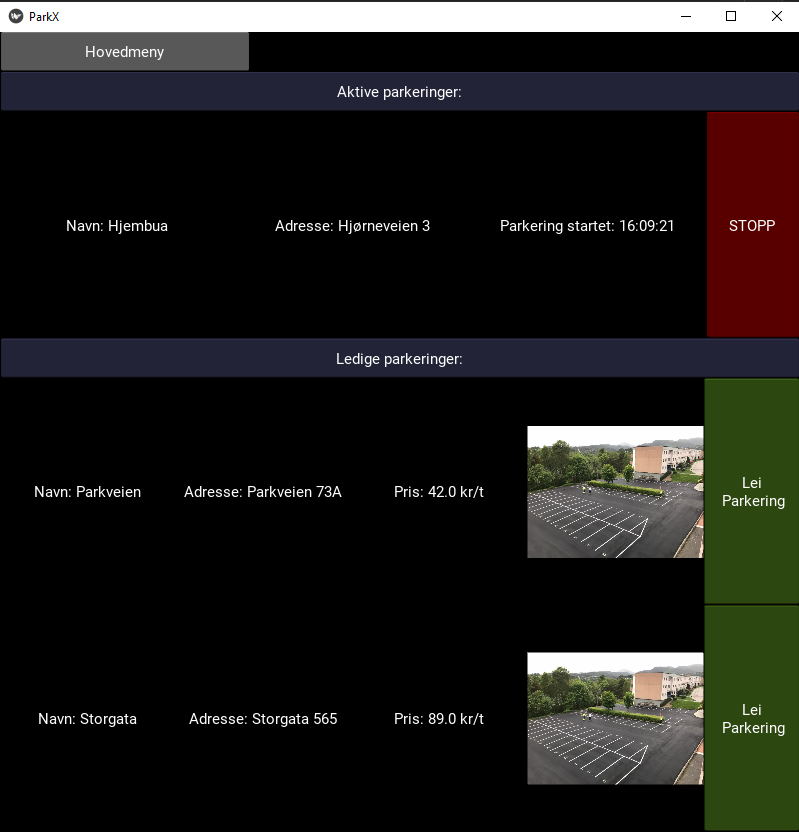
ParkingController klassen er et mellom lag mellom ListRepository og Gui klassen. Den har til oppgave sørge for data flyter enkelt mellom Gui og ListRepository. ParkingController spiller derfor en viktig rolle når Gui ber om data som den vil presenterer for brukeren, eller hvis bruker lager en nye parkeringsplass som må legges til ListRepository. ParkingController har derfor både funksjoner for å kunne kalle ListRepository for å kunne hente parkeringsplasser, samt og leverer informasjon for å kunne lage parkeringsplasser. Disse funksjoner oftest aktivert av Gui kaller dem, som en reaksjon til at bruker gjør noe. ParkingController har også som oppgave å sørge for at informasjon som kommer inn når en parkeringsplass detaljer må endres at de kommer inn på riktig form. Når nye parkeringsplasser skal lages, sørger også kontrolleren for å tildele en ny id til den nye parkeringsplassen. ParkingController hovedoppgave er og fungerer som et mellomlag som kan se gjennom dataen som sendes til ListRepository, samt at den klarer å spør hente parkeringsplasser fra ListRepository og sende dem til Gui.

Sist, men ikke minst har vi også en klasse som styrer betaling. Klassen er skal mest fungerer som en stub, altså en klasse som representere noe funksjonalitet som egentlig en tredjepart program skal styre. Klassen styrer funksjonalitet for automatisk betaling, men har også funksjonalitet for å lage nye betalinger samt å betale dem. Payment klassen har også ansvar for å sørge for om godkjente betalingsmidler er aktivert eller ikke. Egentlig bør et tredjepart program holde styr på dette, så vi har satt to knapper hvor man kan slå av og på godkjente betalingsmidler for å se hvordan programmet oppfører seg med og uten betalingsmidler.

## Hvordan komponentene henger sammen

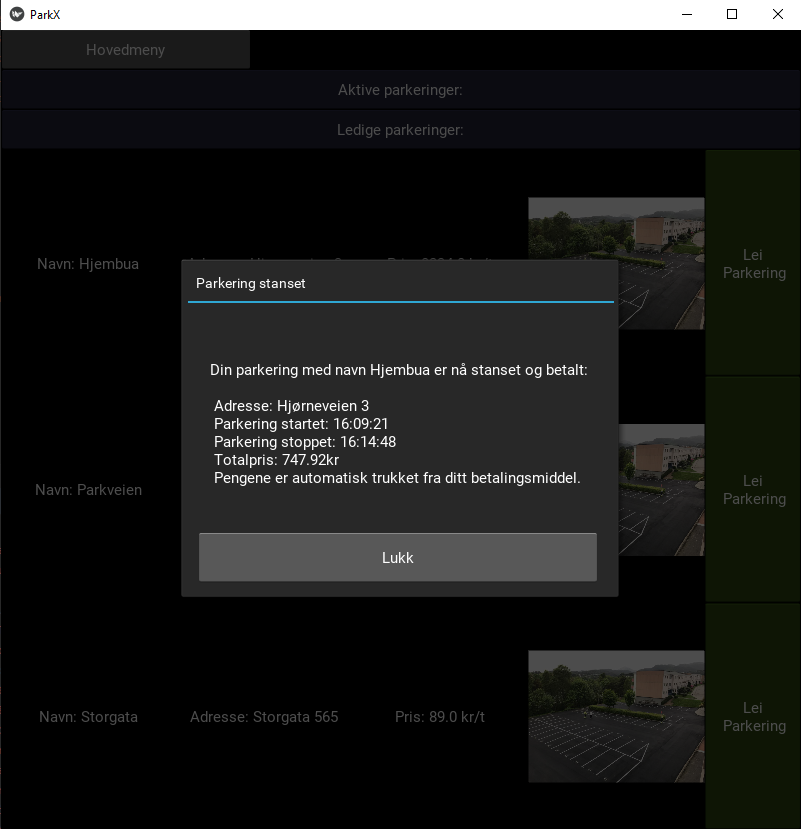
### GUI / hvordan sidene går til hver side

Bruker vil alltid starte på start siden kalt hovedmeny. Her vil bruker få tre velg leier, utleier og min profil(se figur 1). Hvis bruker klikker leier, blir bruker sendt til leier listen. Her vises en liste med parkeringsplasser(se figur 3). Dette er parkeringsplasser bruker kan leie og allerede er i gang med å leie.

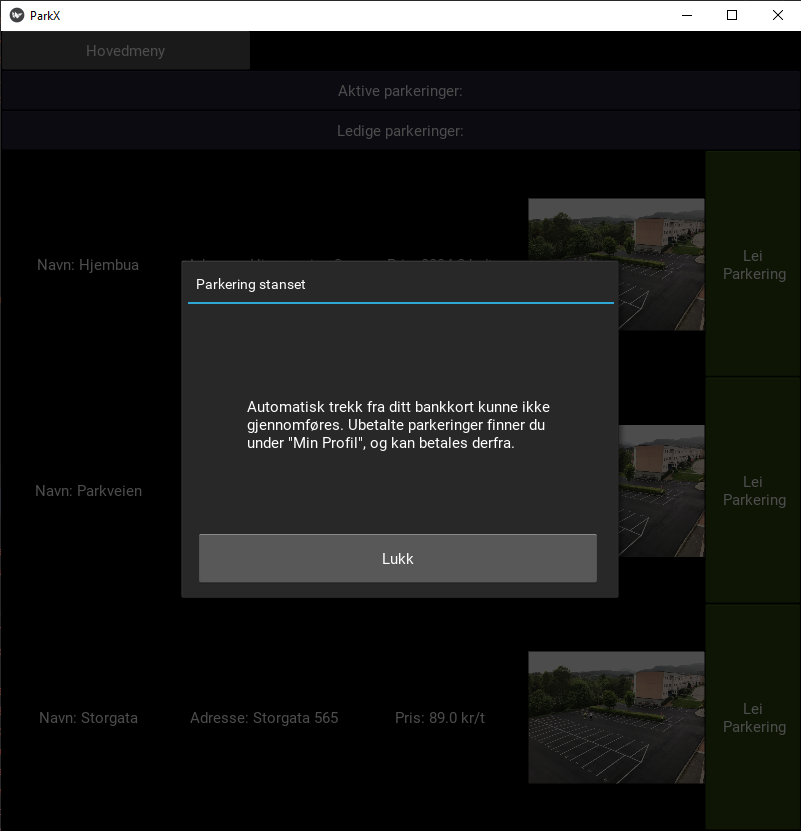


Figur 3: leiers liste med parkeringsplasser

Her fra har bruker 3 valg, hovedmeny, stopp eller lei parkering. Trykker bruker hovedmeny, går bruker tilbake til hovedmeny. Trykker bruker stopp parkeringsplass vil gi en popup. Denne er forskjellig på en betingelse om godkjente betalingsmidler er gitt, les mer om dette under payment. Hvis godkjente betalingsmidler er gitt vil en pop vise informasjon om den nå betalte parkering (se figur 4). Hvis godkjente betalingsmidler ikke er gitt vil en pop-up vises som forteller godkjente betalingsmidler ikke er aktivert (se figur 5).



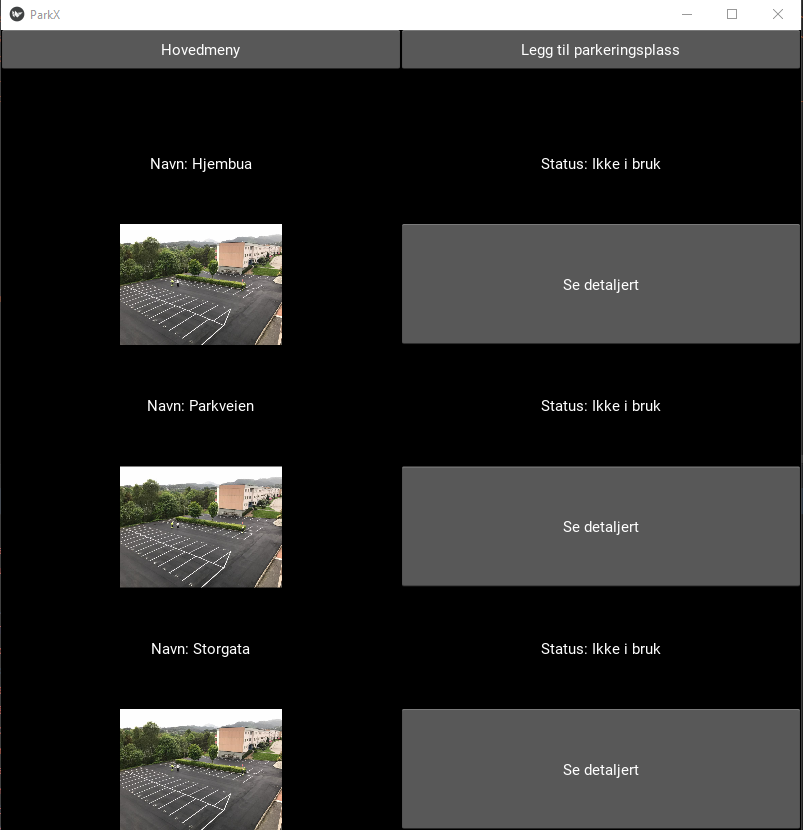
Figur 4: pop-up av informasjon om avsluttet parkering



Figur 5: pop-up info om ikke gitte betalingsmidler

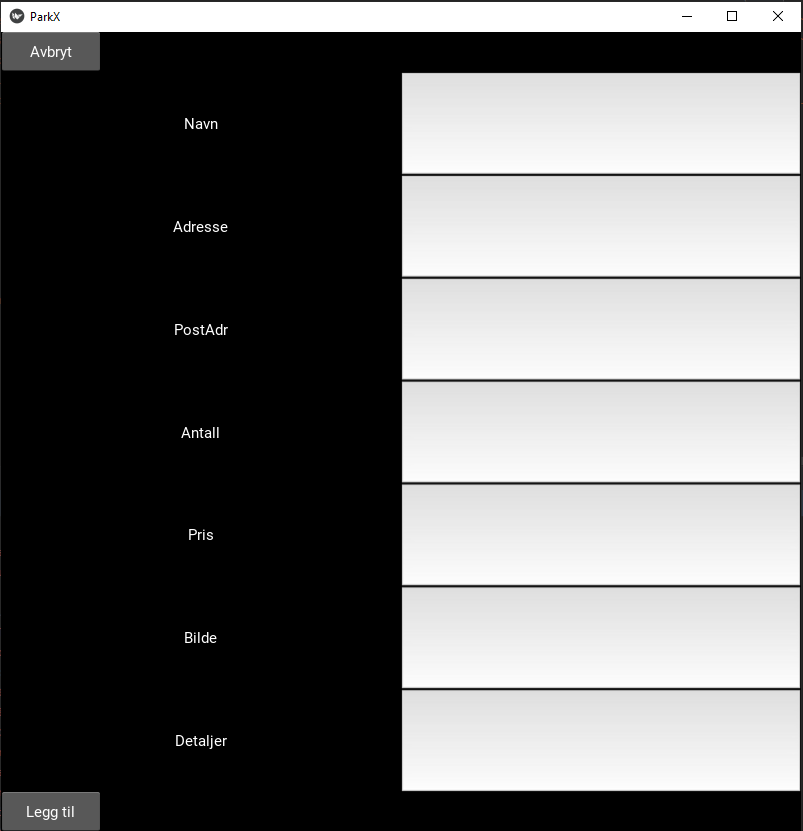
Hvis bruker klikker lei parkering, vil en de vises parkering i detalj (se figur 2). Her får bruker valget å trykke avbryt som sender bruker tilbake til listen med parkeringsplasser. Men bruker får også valget å trykke bekreft, hvor heretter brukes sendes tilbake til listen, med parkeringsplasser. Den parkeringsplassen som ble trykket bekreft på skal ligges til listen over parkeringsplasser leies.

I hovedmenyen får bruker også valget å trykke utleier. Her vil bruker også sendt til en liste med parkeringsplasser (se figur 6). Dette er listen over parkeringsplasser som brukeren selv har lagt ut. Her har brukeren også valget å tilbake til hovedmenyen ved å klikke hovedmeny. Bruker kan tillegg også legge til en parkeringsplass ved å klikke på den knapp som indikerer det. Heretter vil bruker bli sendt til et skjema, hvor en må fylle inn informasjon om den nye parkeringsplass (se figur 7). Her kan bruker avbryte som sender bruker tilbake til listen igjen. Bruker kan også klikke legg til. Hvis all informasjon er skrevet inn og korrekt vil bruker bli sendt tilbake til listen, hvis ikke vil en pop-up feilmelding be deg prøve på nytt (se figur 8).

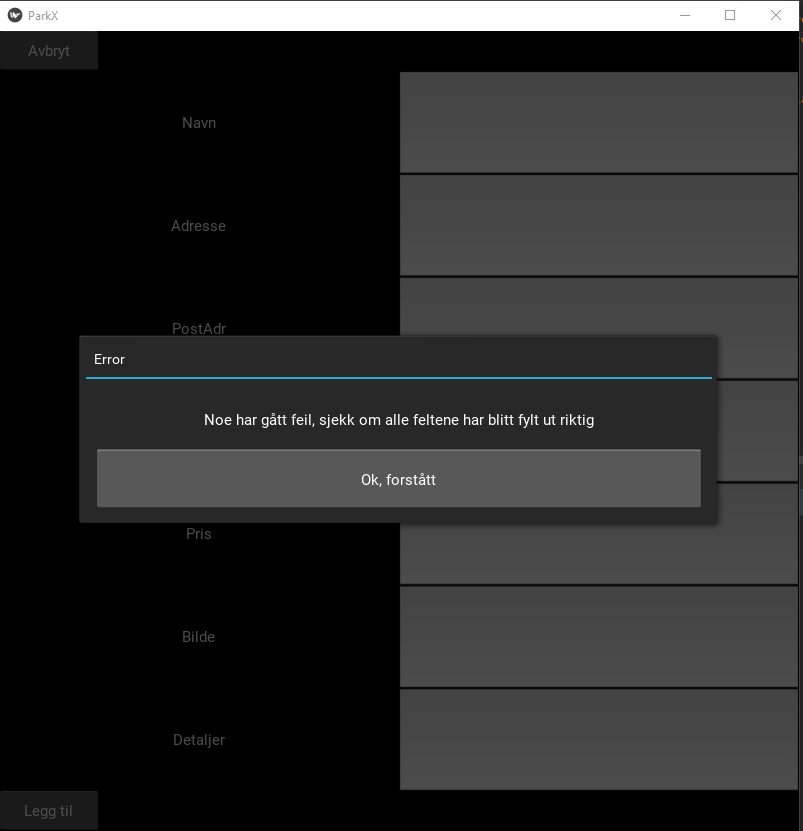


Figur 6: utleiers liste med parkeringsplasser

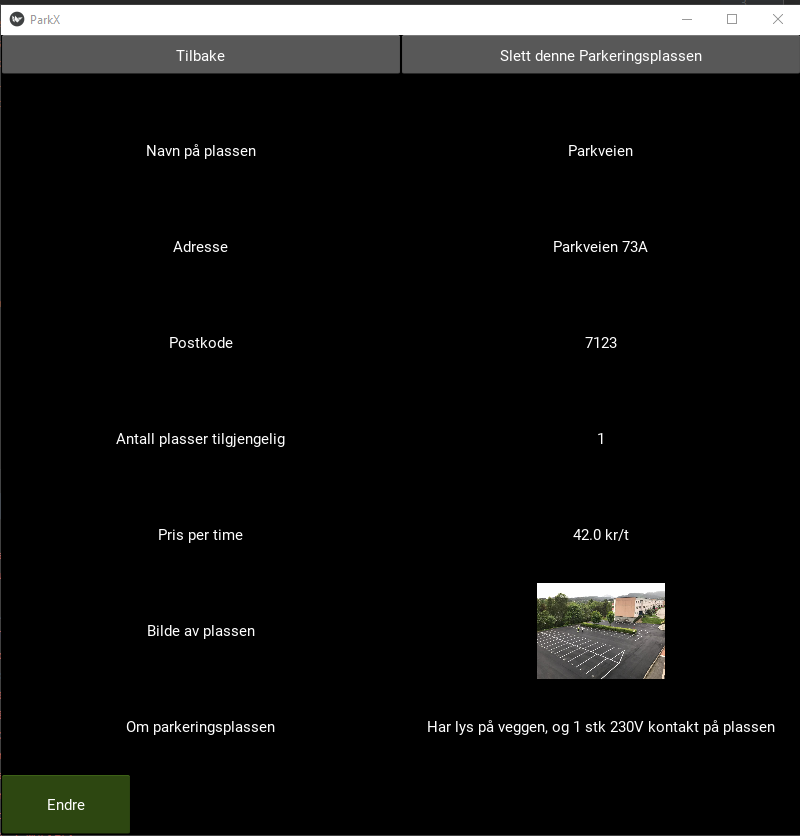
Bruker får også muligheten til å se i detalj. Trykker bruker her, blir en sendt en detaljert visning (se figur 9). Her har bruker også valget å gå tilbake til listen ved å klikke «tilbake». Bruker kan også trykke slett parkeringsplassen, da vil bruker også bli sendt tilbake til parkeringsplass listen. Bruker for også muligheten for å endre parkeringsplassen ved å klikke på «endre». Da vil bruker også bli sendt til et allerede utfylt skjema med den informasjonen som parkeringsplassen hadde fra før (se figur 10). Denne fungerer på presis samme måte som legg til ny parkeringsplass skjemaet og vil også gi en feilmelding pop-up om ikke den angitte informasjon i skjemaet er korrekt.



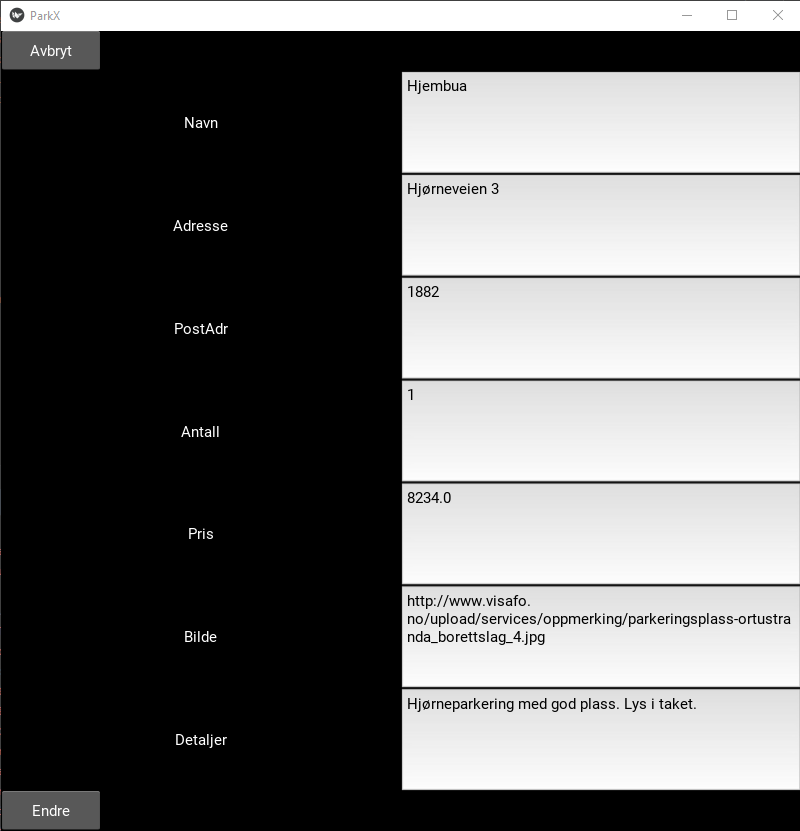
Figur 7: skjema for ny parkeringsplass



Figur 8: feilmelding pop-up om angitt info er riktig



Figur 9: detaljert visning for utleieren



Figur 10: Endre parkeringsplass skjema for utleier

Fra hovedmenyen for også muligheten for å se min profil, ved å klikke på knappen som angir det. Min profil siden er ment for å vise parkeringer som har vært leiet, samt prisen det kostet (se figur 11). Her har bruker igjen muligheten for å klikke på hovedmeny som sender bruker tilbake til hovedmenyen. Bruker har i tillegg to knapper for aktiverer godkjentbetalingsmidler og deaktiverer godkjentbetalingsmidler, disse styrer også hvilken pop-up som vises etter en parkeringsplass. Betalingsløsningen består av tanken om automatisk betaling, men siden vi tenkte at den automatisk trekk løsningen skulle være styrt av en tredjeparts ordningen er det bare blitt lagt inn knapp som aktiverer de to stadier. Er godkjent betalingsmidler deaktivert vil parkeringer som ikke blir betalt da komme inn i listen som vises i midten av vinduet. Bruker får muligheten for betale disse utestående parkeringer, ved å klikke «betal utestående». Heretter kan to pop-vises. Den ene sier du må aktiverer godkjente betalingsmidler om denne ikke er aktivert. Den andre vil si alt er blitt betalt.

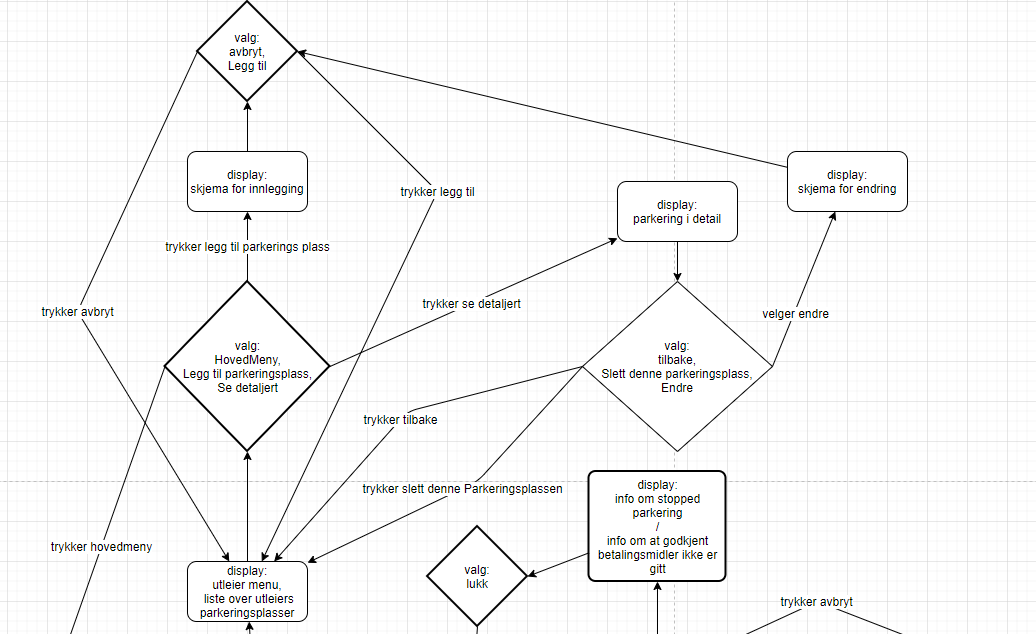
En større visning av hvordan GUI fungerer utfra bruker handlinger er vist på figur 12, i tillegg til 13, 14 og 15 som viser i detalj.



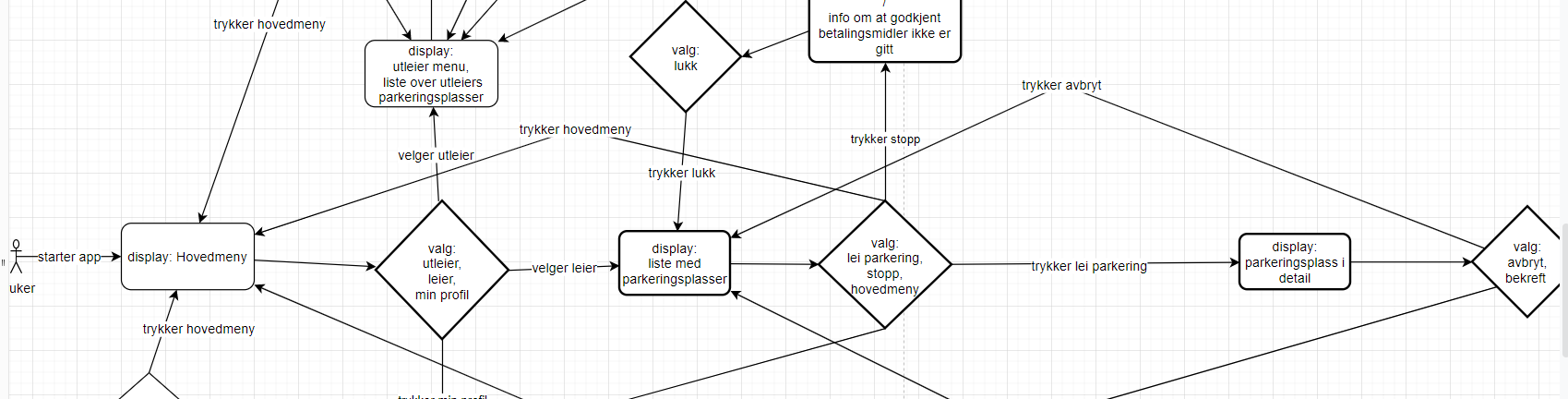
Figur 11: min profil siden

## 

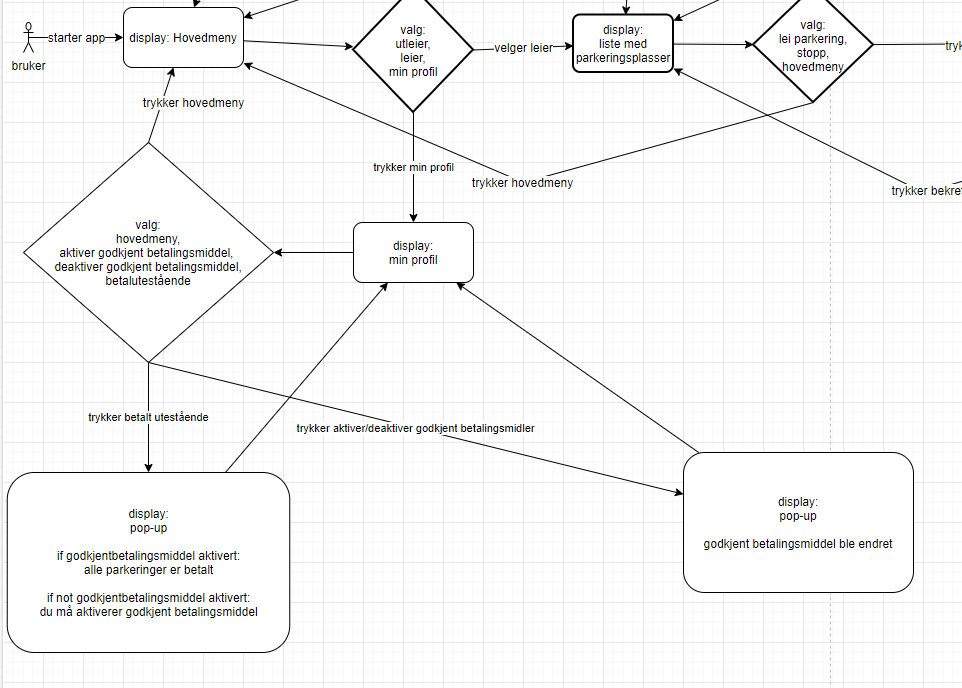
Figur 12: tilstand diagram for hvordan sidene endrer seg, se mer detalj under



Figur 13: øverst del av diagrammet



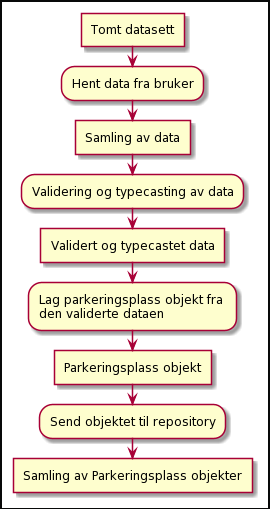
Figur 14: midterst del av diagrammet (zoom inn hvis du ser dette på pc, men det er uklart hva som står)



Figur 15: nederste del av diagrammet

### Innlegging av nye parkeringsplasser

Innlegging av nye parkeringsplasser har inngangspunkt i innlegging skjemaet i utleier delen av GUI. Når informasjon til den nye parkeringsplassen er fylt inn blir informasjonen samlet i en dictionary og deretter sendt til ParkingController. Det første ParkingController vil gjøre er å tildele informasjonen en id. Denne vil også være en del av informasjonen som parkeringsplassen lages utfra ParkingController kommer deretter til å gjøre en validering av den input som ble kjørt. Denne validering går bl.a. ut på å teste om informasjon er på riktig format. Eks: antall parkeringsplasser kan bare være et tall. Men valideringen vil også typecaste informasjonen. Dvs. overfører informasjonen den til den type data vi trenger den i. Dette må gjøres siden all informasjon som kommer fra tekstboksene kommer som en streng. Hvis valideringen feiler av eksempelvis, ikke alle felter er blitt fylt ut eller gitt i feil format, vil en feilmelding bli sendt tilbake til Gui. En pop vil da vises som sier informasjon ble skrevet inn feil. Hvis valideringen ikke feiler, blir informasjonen sendt videre til ListRepository. Her vil ListRepository lage en ny parkeringsplass ut fra informasjonen som ble mottatt. Heretter vil parkeringsplassen bli lagret i ListRepository sin liste, hvor alle parkeringsplasser blir lagret.



Figur : diagram som viser informasjonsflytt ved innlegging av ny parkeringsplass

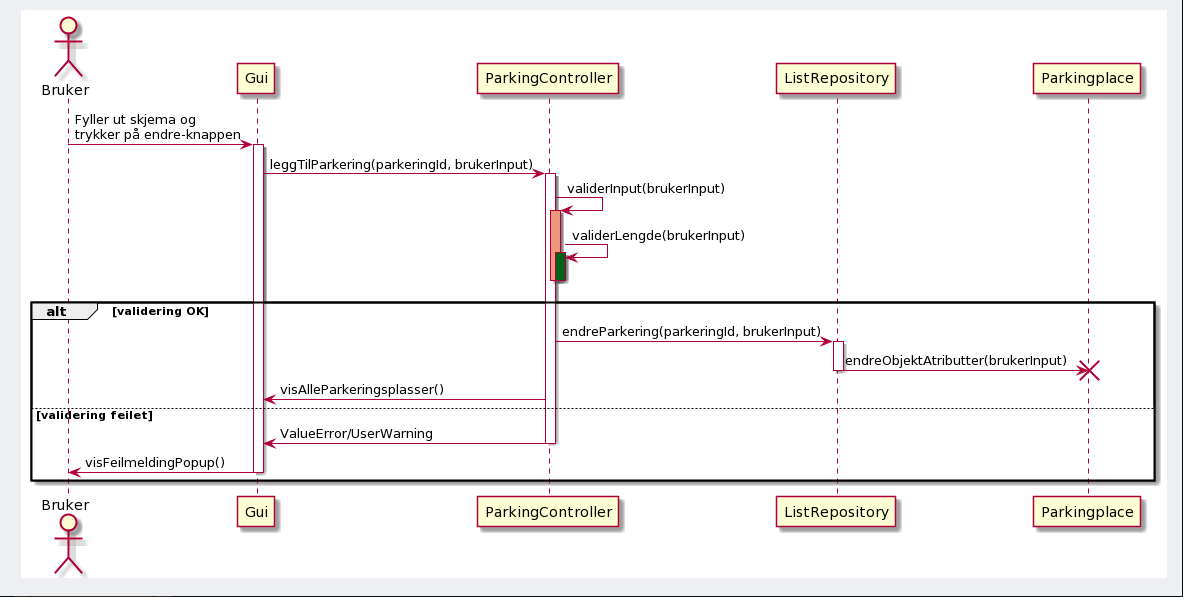
### 

Figure 17: diagram visning av innleggign av ny parkeringsplass

### Endre/slette parkeringsplasser

Endring av parkeringsplasser skjer også i et skjema som ved innlegging av en ny parkeringsplass, men finnes et annet sted. Dette skjema finnes når man trykker endre, i den detaljerte visning av parkeringsplassen for utleier. Skjemaet skiller seg også ut for innlegging av ny parkeringsplass på den måten at all informasjonen som parkeringsplassen hadde fra før står i skjemaets tekstbokser. Når det klikkes endre parkeringsplass er stegene veldig like stegene for innlegging av parkeringsplass. Informasjonen blir sendt til ParkingController for validering og typecasting, deretter blir informasjonen sendt videre til ListRepository som finner frem den riktige parkeringsplassen. For å finne den riktige parkeringsplass brukes id som parkeringsplassen fikk tildelt, da den først ble laget. Det går ikke an å endre på id’en ved endring av parkeringsplass.

Sletting av parkeringsplass, kan skje hvis bruker klikker på slett parkeringsplass i detaljertvisning av parkeringsplassen for utleier. Parkeringsplassens id vil da bli sendt til ParkingController, som da igjen forteller ListRepository at den skal slette parkeringsplassen med den angitte id.



Figur 18: diagram som viser prosessen i endring av en parkeringsplass (stegene er veldig like det i innlegging av parkeringsplass)

### Visning av parkeringsplasser

Visning av parkeringsplasser skjer i GUI. Over ble det gått gjennom hvordan GUI endrer seg når brukeren beveger seg mellom forskjellige sider. Her vil det bli gått gjennom hvordan GUI ren faktisk får informasjonen som skal vise frem.

Det er to forskjellige måter parkeringsplasser vises frem på, i en liste eller en detaljert parkeringsplass om gangen. Begge stegene er ganske like. Hvis bruker som leier eksempelvis trykker på lei parkering, skal parkeringen vises i detalj før bruker klikker bekreft. Får å vise parkeringen i detalj må GUI få den individuelle parkeringsplassen detaljer. Først og fremst når lei parkeringsknappen trykkes vil en funksjon bli aktivert som kommer til å fjerne alt som var på siden og deretter bygge den opp på nytt. Fra den forrige siden vil den ha fått en id, denne id er gjemt i selve knappen slik at når knappen trykkes vil den vite hvilken parkeringsplass den skal vise med id’en. GUI sender deretter id’en til en spesiell funksjon i ParkingController som som forteller at GUI vil ha informasjonen som tilhører parkeringsplassen med den id. ParkingController vil heretter be ListRepository om å gi den parkeringsplassen med den angitte id, i form av en Parkingplace. Denne returnerer ListRepository til ParkingController som sender den videre til GUI. GUI vil ta imot denne og bygge opp GUI utfra informasjonen som parkeringsplassen har. Dvs. den vil vise navn, adresse og bilde i tillegg til andre felter som hører til parkeringsplassen (se figur 2). Dette skjer ved visning av en parkeringsplass, siden er og metoden er den samme for utleier siden bortsett fra at de to knapper er bygget opp med forskjellige knapper å klikke på (se figur 9).

Ved liste-visning når brukeren skal finne parkeringsplassen som brukeren vil leie, blir flere forskjellige parkeringsplasser vist frem med de viktigste detaljer, som navn, adresse og pris (se figur 3). Når denne siden bygges opp, vil GUI si til ParkingController at den trenger listen med alle parkeringsplasser. ParkingController vil heretter be ListRepository om å få listen med alle parkeringsplasser. Denne listen vil deretter bli returnert til GUI som da vil bygge opp en liten beskrivelse for hver parkeringsplass som finnes i listen. I tillegg vil hver liten beskrivelse også ha en knapp, denne knappen får tildelt id’en til parkeringsplassen den står sammen med. På den måten får funksjonen som blir aktivert ved trykk vite hvilken parkeringsplass den har med å gjøre. Denne logikk gjelder også for endring og sletting av parkeringsplass. Når knapper trykkes og det skal inngå en spesiell handling på en parkeringsplass, følger id’en til parkeringsplassen med på knappen.

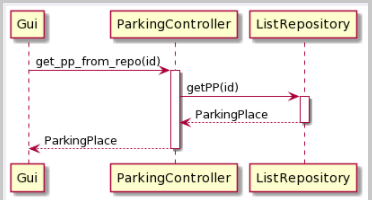


Figure 19: diagram som viser prosessen å hente parkeringsplass informasjon

### Leie og stoppe leie av parkeringsplass

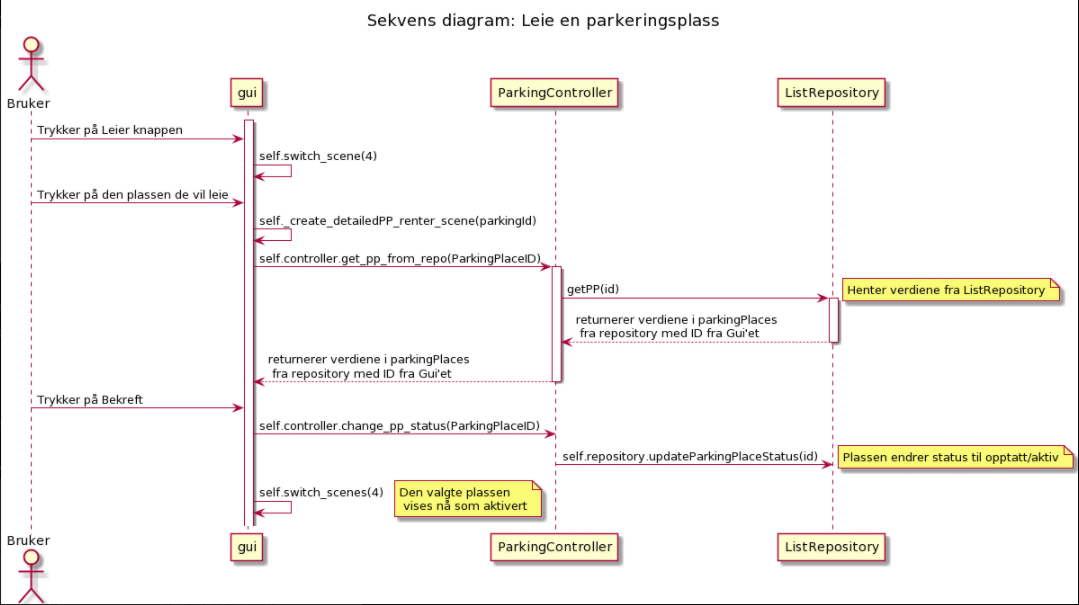
Som beskrevet over ved visning av parkeringsplasser kan vi leie parkeringsplasser. Leie parkeringsplasser skjer i leiers oversikt over alle parkeringsplasser. Leier klikker på lei parkering ved den parkeringsplassen som leier vil leie, hvor etter det byttes til en ny side som viser parkeringsplassen i mer detalj. Oppbygging av selve siden er beskrevet i visning av parkeringsplasser, men allikevel tatt med i diagrammet under for å vise den fulle prosessen. Selve leie prosessen starter når leier klikker på bekreft i den detaljerte visning. Når bruker klikker bekreft vil id’en til den valgte parkeringsplassen bli sendt til ParkingController. ParkingController vil heretter aktivere en funksjon ListRepository med id’en som vil endre statusen til den parkeringsplassen. Parkeringsplassen har også et felt som har oppgave å holde starttidspunktet for leie av parkeringsplassen. Denne vil også bli satt til det nåværende tidspunktet, når endring av status funksjonen blir aktivert. Etter parkeringsplassen har endret status til at den nå blir leiet, vil GUI igjen bytte tilbake til oversikt siden. Men parkeringsplassen som nå blir leiet ligger i den den øverste listen under kalt aktive parkeringsplasser. Denne holder nå en knapp som sier stopp. 

Figure 20: diagram som viser leie prosessen i systemet

Når stopp knappen trykkes vil GUI sende en id’en til ParkingController, som da igjen ber ListRepository endre på statusen til parkeringsplassen med den angitte parkeringsplassen. Denne gang endres statusen til ikke å ikke lengere være i bruk. Heretter vil oversikt siden oppdaterer seg, og en av to forskjellige pop-up vil vises. Godkjente betalingsmidler er ikke aktive eller informasjon om den nå betalte parkeringen. Forklaring til betaling finner du under «lage og betale payments». Når disse pop-up lages vil det igjen bli kalt funksjoner i ParkingController og ListRepository for å lage stopptidspunktet og hente parkeringsplassen som parkeringen ble stoppet på. Hver parkeringsplass har en funksjon for å regne ut hvor mye det kostet å leie parkeringsplassen, denne blir aktivert for å kunne regne prisen det kostet å leie parkeringen, men vil også resette parkering starttidspunktet.

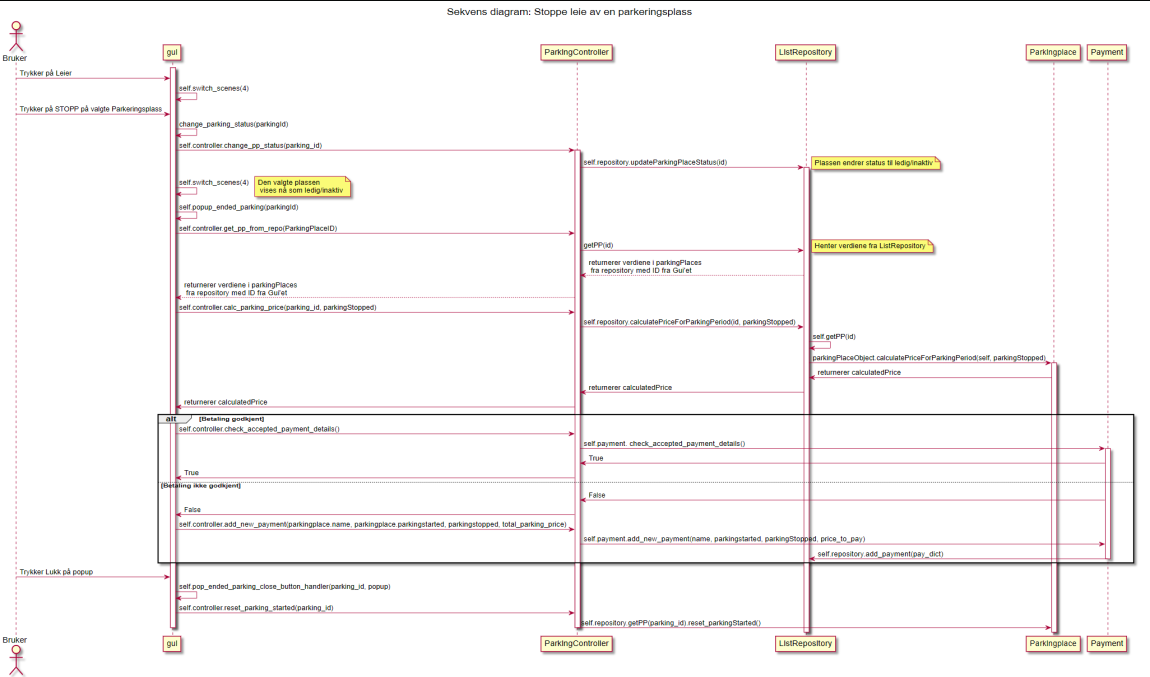


Figure 21: diagram som viser stopp parkering prosessen

### Payment

Payments kan oppfører seg på to forskjellige måter i ParkX, bestemt om godkjente betalingsmidler er aktivert eller ikke. Hvis aktivert vil automatisk trekk skje ved trykk på stopp knappen (betalingsprosessen er beskrevet i neste avsnitt). Da vil betalingen ikke bli lagret i selve programmet og en pop-up vil vises med hva den kostet og i hvilket tidsrom du stod på plassen (se figur 4). All denne informasjon regnes ut som beskrevet over ved å hente parkeringsplassens starttidspunkt og pris per time og regne ut ved også å bruke sluttidspunktet.

Hvis godkjente betalingsmidler ikke er aktivert, vil en annen type pop-up vises. Denne vil fortelle automatisk trekk ikke kunne bli gjort fordi godkjente betalingsmidler ikke er aktivert (se figur 5). Nå vil selve betalingen bli lagret. Informasjonen som vises vanligvis om godkjente betalingsmidler er gitt, vil nå i stedet bli sendt til ParkingController. Heretter sender den videre denne informasjonen til Payment, her vil informasjonen bli samlet i et dictionary og bli sendt til ListRepository som lagrer det i en egen liste hvor alle payments blir lagret.

For å slå av og på godkjente betalingsmidler må brukeren inn på min profil side. Øverst på denne side ses to knapper aktiver og deaktiver godkjente betalingsmidler. Ved klikk på disse endres programmet til å oppføre seg som godkjente betalingsmidler er gitt eller hvis de ikke var gitt. I tillegg vil det i midten av siden bli vist en liste med de betalinger som ikke har blitt betalt som resultat av at godkjente parkeringsmidler ikke var gitt. Slår bruker på godkjente parkeringsmidler vil de neste parkeringer bli automatisk betalt. Men for å betale forrige ikke betalte parkeringsplasser må bruker klikke på knappen «betal utestående». I tillegg må godkjente betalingsmidler også være på for å kunne betale. Hvis ikke godkjente betalingsmidler er på får man bare en pop-up, som sier dette må bli aktivert før det kan betales. Når betalingen endelig skjer ved klikk på «betal utestående» og godkjente betalingsmidler er aktivert, vil de først bli sendt beskjed til ParkingController. ParkingController vil aktivere en funksjon i Payment som sier alle parkeringsplasser skal bli betalt. Denne metoden sjekker først om betalingsmidlene er aktivert. Hvis aktivert vil metoden sende melding til ListRepository om å tømme listen med betalinger.

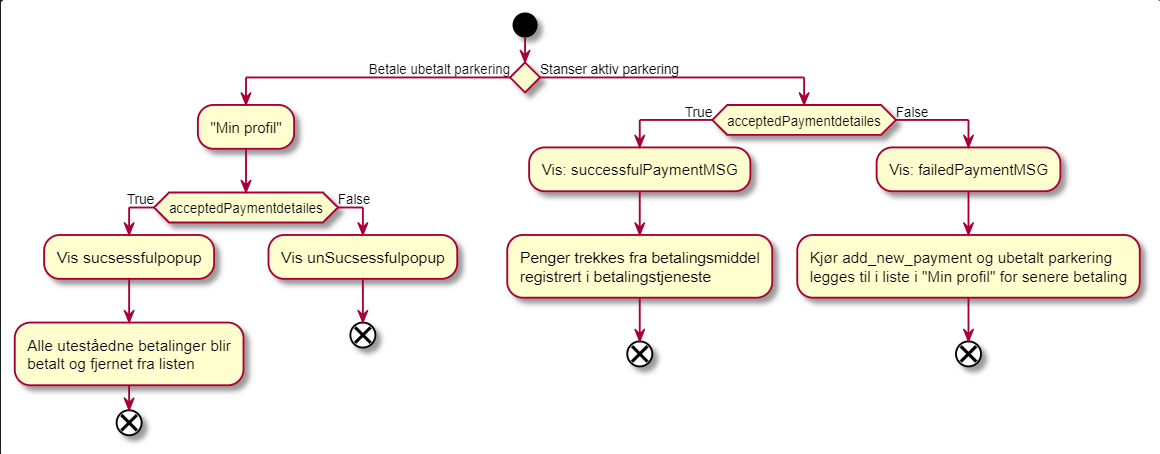


Figure 22: diagram som viser betalingsprosessen

## Svakheter i prototypen

Siden dette bare er en prototype og ikke er noe ferdig produkt, er det forventet det er noen svakheter i prototypen. Noen av svakhetene kommer også direkte av at det er en prototype og mangelen på oppkobling til eksterne servere og databaser.

I prototypen valgte vi at å lage en bruker-klasse ikke var viktig akkurat for prototypen. Det var fortsatt mulig å vise frem hvordan programmet kommer til å se ut når det er ferdig. Men dette er en svakhet til prototypen. Ingen bruker klasse betyr altså vi ikke vil kunne teste hvordan systemet ser ut når vi har flere brukere. Det ses også i systemet direkte ved at parkeringsplasser som legges inn er de parkeringsplasser som man kan leie. Dette valget er med mening siden at bruker klassen ikke får verdi, før vi kan koble det opp servere som gjør vi kan gi forskjellige brukere forskjellige parkeringsplasser. Denne svakhet forsvinner når vi kan koble oss til en server og blir tvunget til å dele opp parkeringsplasser på brukere.

Prototypen har også noen svakheter angående sletting og endring av parkeringsplasser som er i bruk. Leier man en parkeringsplass, går det også an å slette og endre parkeringsplassen. Dette problemet hører også til i større grad at vi ikke har noen bruker klasse. Uten bruker klassen blir det altså også vanskeligere å si noe om hvilke rettigheter bruker har, i tillegg er det er spørsmål om lov og hvilke endringer utleier kan gjøre, mens parkeringsplassen blir utleid.

Prototypen har også noen mindre svakheter, som evnen til å vise bilder går kun gjennom en URL. Vi bestemte dette ikke var noen MVP å gå i dybden på hvordan å åpne file explorer eller brukerens bilder på mobil. Dette hadde selvfølgelig vært gjennomført på en bedre måte hvis ikke dette bare var en prototype. Et annet mindre problem er at tiden som blir satt for å regne ut når parkering starter og stopper ikke går på dato. Dvs. prototypen vil altså vise feil om bruker leier mer enn en dag med prototypen. Dette blir ikke noe problem man nødvendigvis legger merke til med mindre prototypen brukes over lengere tid.