Høgskolen i Østfold

Software Engineering & Testing 2019

Dokumentasjon av billettsystem

Gruppe 4

Axel Grefslie
axelg@hiof.no

Camilla Kolstad
camilko@hiof.no

Daniel Kallak
danielk@hiof.no

Geir Kirksæther
geirki@hiof.no

Innholdsfortegnelse

Figurliste

Introduksjon til systemet

Prosjektets omfang

Oppgaven presentert i Software Engineering og Testing 2019 går ut på å utvikle en prototype av et kjernesystem. Dette systemet skal benyttes av kinoer, som skal bistå lokale arrangører med å kunne opprette arrangementer, selge billetter og håndtere inngangskontroll. Et eksempel på et slikt arrangement kan være en lokal skole som vil selge billetter til en tur. I tillegg skal systemet kunne brukes som et vanlig billettsystem for kinoene. Et eksempel på et slikt arrangement kan være en kinoforestilling til en film.

Parallelt med utviklingen av kjernesystemet skal det utvikles tester som viser at kravene vi har utviklet for systemet er tilfredsstilt, samt avdekke feil som kan oppstå ved bruk av systemet. Det skal også skrives dokumentasjon som beskriver prototypen og tar for seg alle elementer som må beskrives på et nivå slik at en utestående person forstår og kan overta utviklingen.

Ordforklaringer

Backend	Delen av et system som ikke er tilgjengelig for en bruker og den delen av ligger nærmest databasen.
Frontend	Delen av systemet som ligger nærmest brukeren. Koden som formerer det man visuelt ser.
Mocking	Brukt i forbindelse med unit testing. En teknikk som gjør det mulig å teste kode uten å være avhengig av avhengigheter.
Maven	Apache Maven er et byggesystem. Det definerer hvordan .java filer blir kompilert

	til .class filer og pakket til .jar filer.
Gradle	Et byggesystem basert på Apache Maven.
Intellij	Et utviklingsmiljø hvor man kan utvikle programvare.
jUnit	Et verktøy for å teste programvarekomponenter skrevet i programmeringsspråket Java.
Java	Objektorientert programmeringsspråk
Klasse	En klasse er en blåkopi hvor individuelle objekter blir opprettet fra.
Metode	En metode er en gruppe med statements som skal utføre en operasjon.
Variabel	Blir brukt til å lagre informasjon som kan refereres til eller manipuleres i et program.
Objekt	Objekter blir opprettet fra hvordan en klasse har definert den.
API	Application Programming Interface. Et programmeringsgrensesnitt.

Hensikten med systemet

Underholdningsbildet har blitt større i både store og små arrangementer. Kinoene ønsker derfor å kunne bistå alle lokale arrangementer med salg. Hensikten er at kinoer skal kunne benytte systemet til å selge vanlig kinobilletter, kunne opprette andre arrangementer for eksterne arrangører og bistå med salg av billetter til disse.

Hva går løsningen ut på

Løsningen i dette prosjektet går først og fremst helt overordnet ut på å ha et bestillingssystem for landets kinoer slik at de kan administrere salg av egne billetter i tillegg til å administrere arrangementer for andre lokale arrangører typisk innenfor kultur, utdanning og uteliv. Tidligere har mange av disse administrert sine egne arrangement med det administrasjonsapparatet dette innebærer. Kinoene, som allerede har hatt et slikt system for bestilling og reservering av plasser til kinobilletter, de kan derfor bistå andre aktører slik at de kan konsentrere seg om sitt arrangement og la kinoene ordne med bestillingene.

Systemet er utviklet slik at det er en kino administrator som styrer alle bestillingene. Denne administratoren vil kunne foreta alle de konkrete handlingene koblet til et arrangement. Når en arrangør vil opprette et arrangement vil systemet via en kino administrator opprette et arrangement med nødvendig informasjon og sette en eventuell billettpris for arrangementet. Kontroll av billett er noe en arrangør utfører ved inngang eller aktivering av arrangementet for den aktuelle kunden som har kjøpt eller skaffet seg en billett. Denne kontrollen kan være f.eks. en håndscanner utenfor en konsert. Basalt sett er denne sjekk av gyldighet på billett kun det som en arrangør gjør i systemet. Alt annet foretas av kino administrator. Når en arrangør også utfører sjekk av gyldig billett er også dette logisk da det er de som eier arrangementet. Denne sjekken av gyldighet er også noe en kino administrator gjør, som nevnt innledningsvis er det kino administrator som skal bistå de lokale arrangørene med dette. På et arrangement som f.eks. en sopptur med den lokale barnehagen kan barnehagen få tilsendt en liste med billetter som er gyldige og eventuelt ugyldige og der direkte foreta en kontroll.

Det er kinoen som er administrator av alle arrangementer. Når arrangementet er opprettet fra kino administrator, settes riktig arrangør koblet til arrangementet. Deretter settes kontaktinformasjon og en eventuell pris. Når en arrangør er koblet til et arrangement settes det også en provisjon til kinoen som opprettet arrangementet. Hensikten bak dette er

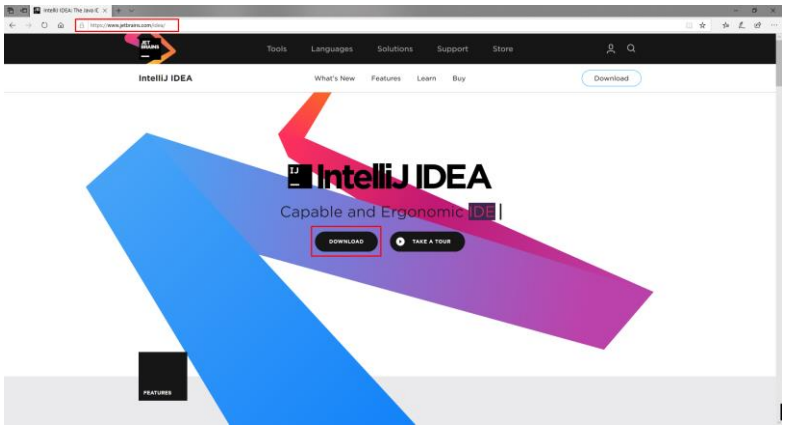
tilnærmet selvforklarende men det ville vært liten hensikt i at en kino administrerer slike andre arrangementer hvis det ikke ble utviklet en form for provisjon til kinoen. Når det opprettes en filmvisning under en kino som har kinosalen så settes provisjonen til kinoen til 100 %. Denne raten på provisjon bør i etterkant kunne endres i systemet. Prototypen tar ikke hensyn til dette, men vi vil komme tilbake til denne potensielle svakheten is systemet mot slutten av rapporten.

Systemet med prototypen i sentrum skal kunne fange opp alle handlinger som har med det å kjøpe, selge, administrere og avbestille billetter. Det er kino-administrator som gjør de fleste handlingene og det er de som samhandler med systemet. Det vil vi forklare i følgende scenarier for bruk av systemet senere i dokumentasjonen. Når løsningen er utviklet komplett vil det forenkle kjøp og salg av billetter for underholdningsbildet i norske byer. Det er dette systemet skal gjøre. Beskrivelse av disse handlingene vil bli dokumentert senere i dokumentasjonen.

Instruksjoner for oppstart av systemet

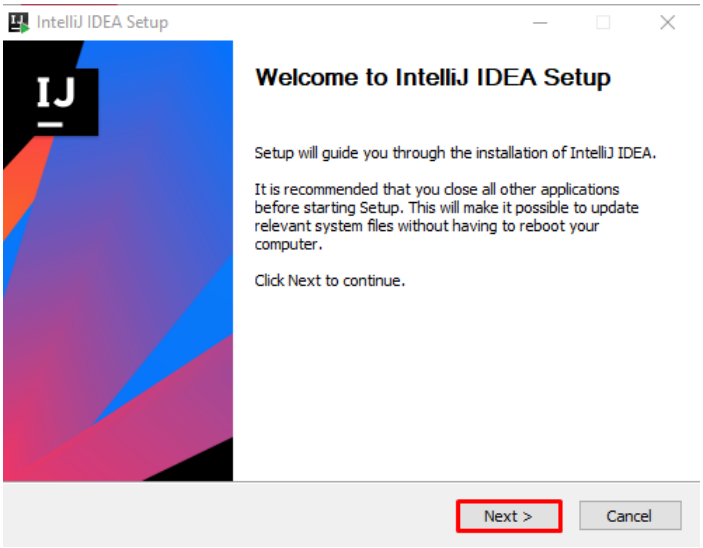
Her kommer instruksjoner over hvilke steg man må følge for at prototypen skal starte opp og kjøre på en ny maskin.

1. Hvis IntelliJ ikke er installert på datamaskinen som skal kjøre systemet, må det lastes ned fra https://www.jetbrains.com/idea/. Hvis IntelliJ finnes fra før, kan du gå videre til steg 3.



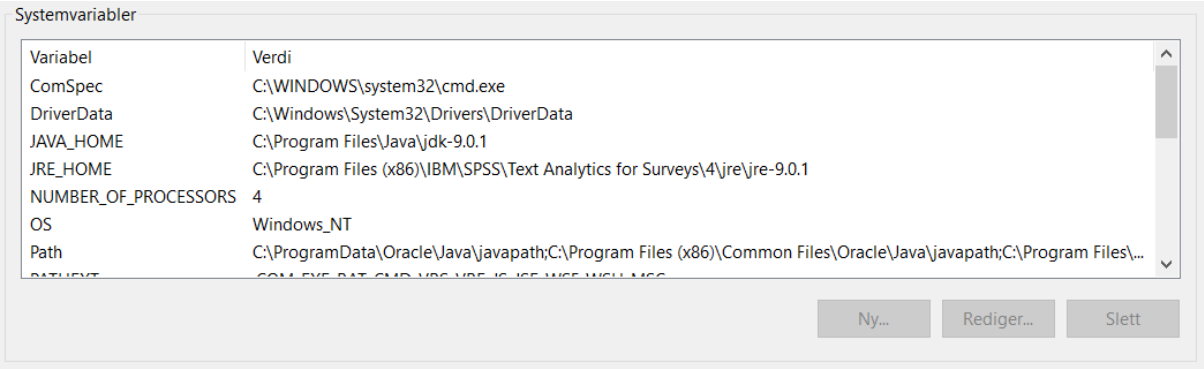
Figur 1.1: Et skjermbilde av nedlastingssiden for IntelliJ.

2. Følg deretter instruksene installasjonen av programmet gir for nedlastning.



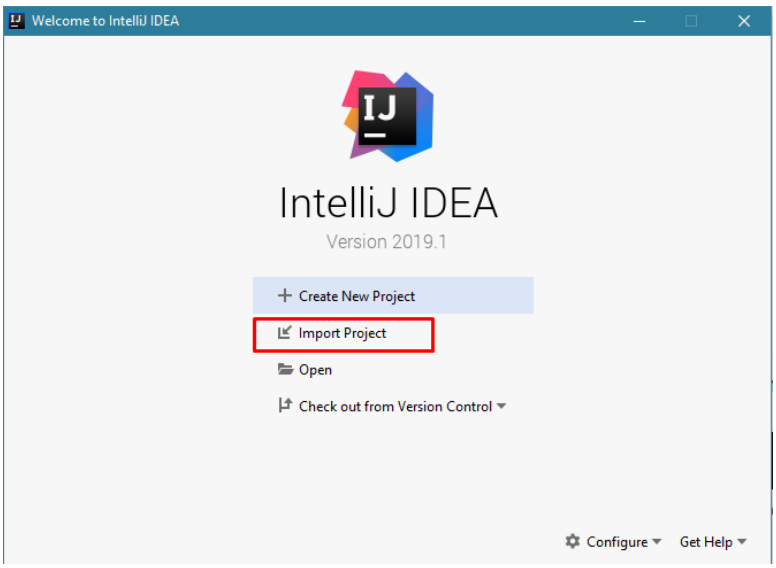
Figur 1.2: Et skjermbilde av installasjon av IntelliJ.

3. Om man ikke har konfigurert stien(path) til java i miljøvariablene, er det viktig at dette gjøres før man i det hele tatt får kjørt og kompilert prosjekter i IntelliJ IDEA. Om man da lokaliserer seg til Kontrollpanel → System og sikkerhet → System, finner man nede på siden at man kan endre innstillinger. Fra vinduet man får opp, trykker man videre på avansert og deretter på miljøvariabler. Figur 1.3 viser variablene, hvis JAVA_HOME ikke finnes i listen, så må den opprettes. Trykk på *Ny* og skriv inn det som står i figur 1.3.



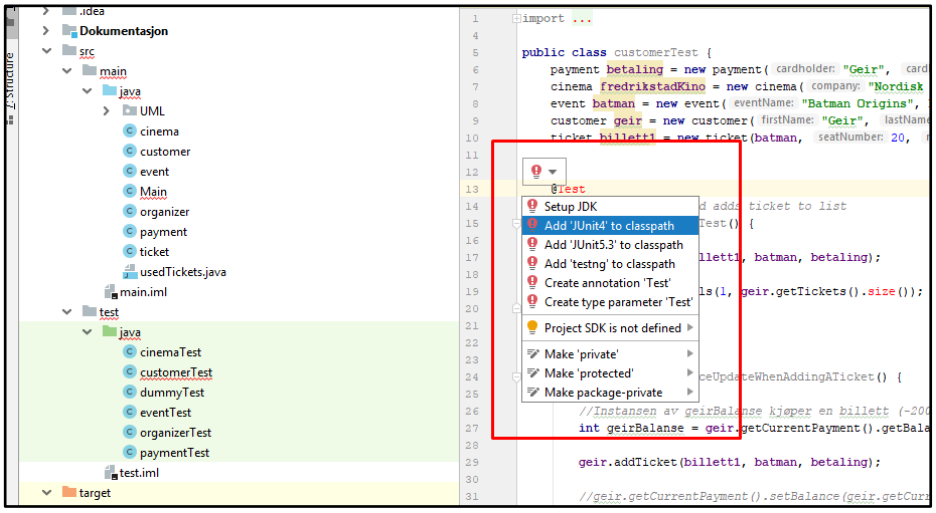
Figur 1.3: Et skjermbilde av hvordan man konfigurerer stien til miljøvariablene.

4. Når IntelliJ er ferdig installert, så kan prosjektet importeres fra datamaskinen. Klikk på Import Project og velg mappen systemet befinner seg i.



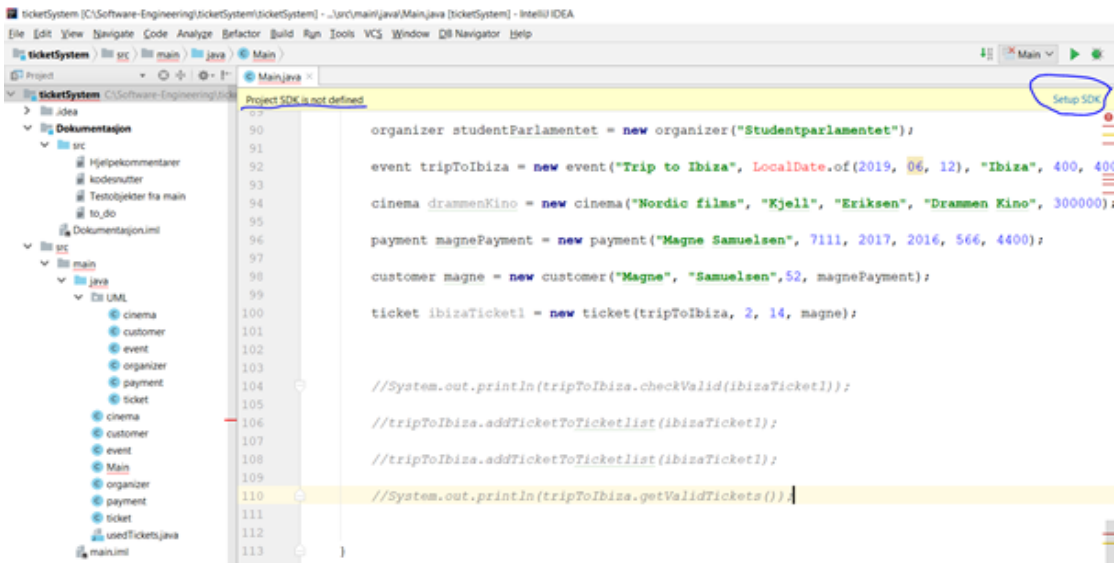
Figur 1.4: Et skjermbilde av IntelliJ og importering av prosjekt

5. Når prosjektet er importert inn i IntelliJ, må jUnit4 konfigureres for at testene skal kunne kjøre. Klikk på @Test i en testklasse som er rød og deretter klikk på lyspæren som vil dukke opp. Her vil det stå Add jUnit4 to classpath, klikk på den. Nå vil testene kunne kjøre (det kan også dukke opp en dialogboks som spør deg om du vil bruke JUnit4 fra IntellJ distribusjonen eller kopiere JUnit4 bibliotektsfilene til valgte katalog, her velger man det første).



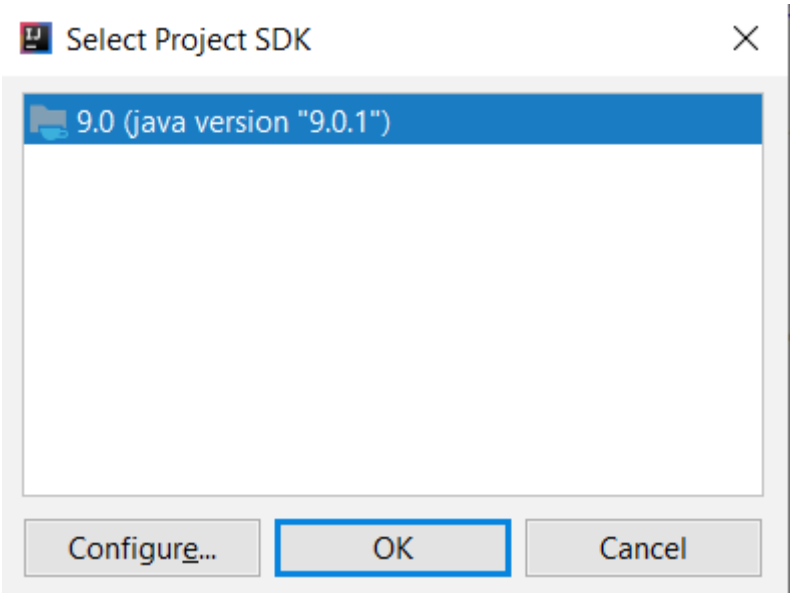
Figur 1.5: Et skjermbilde av hvordan man konfigurerer jUnit4.

6. Konfigurering av SDK. Hvis det dukker opp en feilmelding som sier *Project SDK is not defined* må følgende gjøres i IDEen:



Figur 1.6: Et skjermbilde av hvordan man konfigurer SDK.

Her ser du feilmeldingen. Trykk på *Setup SDK* og denne dialogboksen vil komme frem:



Figur 1.7: Et skjermbilde av hvordan man velger SDK.

Pass på at du har minst versjon 9.0 her. Prototypen er som nevnt i det ikke-funksjonelle kravet for korrekt Java-versjon som skal være på minst 9.0 plattformen. Trykk på OK på versjon 9.0. Hvis man ikke har noen versjoner installert, må man laste ned Java fra: URL til Oracle sin side her: https://www.oracle.com/technetwork/java/javase/downloads/index.html.

Instruksjoner for kjøring av .BAT filer for visualisering av use-cases

1. Finn frem til stien Software-Engineering\ticketSystem\ticketSystem\src\main\java.
2. Klikk på usecases.bat. Deretter kan du velge use-case og få ut visualisering av denne.

Overordnet beskrivelse og visualisering av prosjektet

Konstruktører til klassene vi har brukt

Cinema

Siden cinema-klassen arver fra organizer, har vi satt inn en super som lar oss opprette objekter av cinema-klassen med felter fra organizer. Forskjellen på et Cinema-objekt og et organizer-objekt er at Cinema-objekter får et eget cinema-navn, samt tilgang til forskjellige metoder i klassen.

```
public cinema(String cinemaName, String company,
              String referenceFirstname, String referenceLastname, int organizerBalance) {
    super(company, referenceFirstname, referenceLastname, organizerBalance);
    this.cinemaName = cinemaName;
}
```

Figur 2.1: Et skjermbilde av konstruktøren og super i cinema.

Customer

Dette er konstruktøren for klassen customer. Den krever for- og etternavn, alder og payment. Aldersrestriksjonen er til for å unngå at brukere som ikke er gamle nok for et event, ikke får tilgang til å kjøpe billett. Vi gir brukeren en gitt saldo så vi lettere kan ha noe å teste imot når vi kjøper og refunderer billetter med objektene. Payment og betaling er noe vi ikke kommer til å implementere noe videre på i prototypen.

```
public customer(String firstName, String lastName, int age, payment currentPayment) {
    this.currentPayment = currentPayment;
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
}
```

Figur 2.2: Et skjermbilde av konstruktøren i customer.

Event (Filmvisning)

Dette er konstruktøren for film-eventer. Vi har to konstruktører for event, slik at objektet blir lagt i riktig liste ettersom hvilke parameter vi bruker når vi oppretter objekter. Med dette kan vi presentere hvilke arrangementer som finnes basert på type arrangement.

```
//Constructor for all movies
public event(String eventName, LocalDate date, String location, int stage, int totalTickets,
             int ticketsRemaining, organizer organizer, int price, boolean isMovie) {
    this.eventName = eventName;
    this.date = date;
    this.location = location;
    this.stage = stage;
    this.price = price;
    this.totalTickets = totalTickets;
    this.ticketsRemaining = ticketsRemaining;
    this.organizer = organizer;
    this.isMovie = isMovie;
    event.getAllEvents().add(this);
    event.getFilmList().add(this);
}
```

Figur 2.3: Et skjermbilde av konstruktøren for film-eventer.

Event (Andre arrangementer)

```
//Constuctor for all other events
public event(String eventName, LocalDate date, String location, int totalTickets, int ticketsRemaining,
             organizer organizer, int price, boolean isMovie) {
    this.eventName = eventName;
    this.date = date;
    this.location = location;
    this.price = price;
    this.totalTickets = totalTickets;
    this.ticketsRemaining = ticketsRemaining;
    this.organizer = organizer;
    this.isMovie = isMovie;
    event.getAllEvents().add(this);
    event.getOtherEvents().add(this);
}
```

Figur 2.4: Et skjermbilde av konstruktøren for andre arrangementer.

Organizer

Konstruktøren for organizer.

```
public organizer(String company, String referenceFirstname, String referenceLastname, int organizerBalance) {
    this.company = company;
    this.referenceFirstname = referenceFirstname;
    this.referenceLastname = referenceLastname;
    this.organizerBalance = organizerBalance;
}
```

Figur 2.5: Skjermbilde av konstruktøren i klassen organizer.

Ticket

Konstruktør for "ticket". Her har vi to konstruktører. Forskjellen her er at den ene tillater opprettelse uten setenummer og rad. Konstruktøren med setenummer og rad er tilpasset opprettelse av cinema-objekter hvor seteplassering er nødvendig. Det er ikke gitt at alle arrangementer har bruk for setenummer og rad, så ved bruk av flere konstruktører lar vi arrangører velge hva som passer dem best.

```
public ticket(event etEvent, int seatNumber, int row, customer ticketHolder) {
    this.ticketOfEvent = etEvent;
    this.seatNumber = seatNumber;
    this.row = row;
    this.ticketHolder = ticketHolder;
    this.valid = false;
    this.ticketID = teller.incrementAndGet();
    ticketlist.add(this);
}

public ticket(event etEvent) {
    this.ticketOfEvent = etEvent;
    this.valid = false;
    this.ticketID = teller.incrementAndGet();
    ticketlist.add(this);
}
```

Figur 2.6: Skjermbilde av konstruktørene i klassen ticket.

Payment

Dette er bare en test klasse slik at vi har noe å teste imot knyttet til betaling etc. Denne klassen er ment for en annen utvikler å utvikle videre mot tentative betalingsløsninger som NETS, Paypal eller andre muligheter.

```
public payment(String cardholder, String cardNumber, int validFrom, int validTo, int CVC2, int balance) {
    this.cardholder = cardholder;
    this.cardNumber = cardNumber;
    this.validFrom = validFrom;
    this.validTo = validTo;
    this.CVC2 = CVC2;
    this.balance = balance;
}
```

Figur 2.7: Skjermbilde av konstruktøren for test klassen payment.

Metoder

Her vil vi komme med alle metoder som løser minstekravene vi har satt til prosjektoppgaven. Vi vil ikke skrive om alle metodene vi har brukt, men de som er mest viktige og som prosjektet avhenger mest av.

Cinema

sellTicket()

I denne klassen har vi kun brukt en metode, som er for å selge billett. I starten av metoden utfører vi en metode gitt i customer-klassen, som er å legge til billetten som blir kjøpt i en liste over kundens billetter. Vi har valgt å kjøre denne metoden her fordi dette alltid skjer når man kjøper en billett. Kunden blir så trukket for billetten. Deretter oppdateres eventet som billetten tilhører med én mindre billett tilgjengelig. Billetten blir så satt til å være gyldig.
Vi kjører en test og ser om billetten som skal bli solgt er en kinofilm eller ikke. Om den er det, oppdaterer vi cinema sin balanse med prisen som billetten koster. Om billetten ikke er en kinofilm, vil det si at eventet tilhører en organizer og prisen for hver billett skal fordeles mellom organizer og cinema. Vi oppdaterer derfor og gir cinema 20 % og organizeren 80 % av provisjonen. Organizer får 80 % fordi det er deres event, mens cinema får 20 % for jobben å selge billetten for organizeren.

```
public void sellTicket (customer customer, event event, organizer organizer, ticket ticket, payment payment) {

    customer.addTicket(ticket, event, payment);
    customer.getCurrentPayment().setBalance(customer.getCurrentPayment().getBalance() - event.getPrice());

    event.setTicketsRemaining(event.getTicketsRemaining()-1);

    ticket.setValid(true);

    if (event.isMovieEvent() == true) {
        this.setOrganizerBalance(this.getOrganizerBalance() + event.getPrice());

    }

    else if (event.isMovieEvent() == false) {
        organizer.setOrganizerBalance(organizer.getOrganizerBalance() + (event.getPrice() * 80 / 100));
        this.setOrganizerBalance(this.getOrganizerBalance() + (event.getPrice() * 20 / 100));
    }
}
```

Figur 2.8: Skjermbilde av metoden sellTicket().

Customer

addTicket()

Her har vi metoden for å gi kunden eierskap over en billett. For å kjøre denne metoden krever vi tre parametere (ticket, event, payment). Metoden består av en sjekk, som sjekker om payment objektets balanse er større en arrangementets pris. Hvis situasjonen er slik legges billetten til i kundens liste. Hvis ikke vil kunden få meldingen: "Du har ikke dekning på kortet!". Dette er selvfølgelig noe en tredjepart vanligvis ville tatt for seg, men vi har med funksjonaliteten for å oppdatere balansen til kunden, slik at vi kan kjøre tester mot dette.

```
public void addTicket(ticket ticket, event event, payment payment){

    if (payment.getBalance() > event.getPrice()) {
        tickets.add(ticket);
    }
    else {
        System.out.println("Du har ikke dekning på kortet!");
    }
}
```

Figur 2.9: Skjermbilde av metoden addTicket().

cancelTicket()

Denne metoden lar deg kansellere en billett. Her løper vi igjennom listen med billetter og finner riktig ID knyttet til billetten vi ønsker å kansellere. Når systemet finner riktig ID kjøres det en test på om billetten er gyldig eller ikke. Hvis billetten er gyldig, vil arrangementets tilgjengelige billetter bli oppdatert samt at pengene blir refundert til kunde. Hvis ikke billetten er gyldig.

```
public void cancelTicket (ticket ticket, event event) {
    for (int i = 0; i < tickets.size(); i++) {
        if (tickets.get(i).getTicketID() == ticket.getTicketID()) {

            if (tickets.get(i).isValid() == true) {
                event.setTicketsRemaining(event.getTicketsRemaining()+1);
                this.getCurrentPayment().setBalance(this.getCurrentPayment().getBalance() + event.getPrice());
            }
            else {
                System.out.println("Beklager, det har skjedd en feil!");

            }
            ticket.setValid(false);
            tickets.remove(i);
        }
    }
}
```

Figur 2.10: Skjermbilde av metoden cancelTicket().

Event

checkValid()

Denne metoden sjekker om en billett i et event er gyldig eller ikke. Den går først gjennom listen over billettene til eventet

```
public boolean checkValid (ticket ticket) {

    int index = ticketsForEvent.indexOf(ticket);

        while (ticketsForEvent.get(index).isValid()){
            System.out.println("Denne billetten er gyldig!");
            return true;
        }
    System.out.println("Billetten er IKKE gyldig!");
    return false;
}
```

Figur 2.11: Skjermbilde av metoden checkValid().

Organizer

addEvent()

Det eneste denne metoden gjør er å legge til arrangementer til en organizer, slik at de forskjellige organizerne kan få en liste over sine arrangementer.

```
public void addEvent(event event){
    events.add(event);
}
```

Figur 2.12: Skjermbilde av metoden addEvent().

Prosjektets ramme og begrensninger

Hovedrammen til prosjektet ligger i en programvareløsning som er en backend-løsning. Det er en backend-løsning når denne dokumentasjonen ble skrevet da gitt tidsramme ikke tillot for en grafisk utvikling av systemet mot en frontend system hvor en visuell tilnærming til systemet vil kunne vises hos kunden. I denne dokumentasjonen skal vi imidlertid komme med beskrivelse av hvordan dette på et tidspunkt kan implementeres på en fornuftig måte. Prototypen til dette systemet som vist i utviklingsdokumentasjon er uansett komplett i den forstand at det kan lages et grafisk brukergrensesnitt som enkelt kan brukes for å implementere dette for kunden, kino administratoren og arrangøren. Begrensninger til systemet ligger videre også på utvidelser og "nice-to-have" funksjonalitet knyttet opp til det å administrere et arrangement som utføres av kino administrator. Begrensningene påvirker ikke den viktigste personaen i systemet, kunden som faktisk er den som betaler for tjenestene til kino administrator og/eller leverandøren. Den påvirker heller ikke personaen, KinoAdmin. Utvikling av digital betalingsløsning er det kun benyttet en payment løsning med fiktive betalingsløsninger for å kunne dokumentere at en tentativ løsning faktisk vil fungere. Kravspesifikasjonen til dette er dokumentert senere i dokumentasjonen. Andre ting som heller ikke prototypen dekker gjelder lagringen av dataene i en mulig database samt en håndscanner API som skal foreta avlesning av gyldige eller ugyldige billetter ute på selve arrangementet.

Teknisk rammeverk og plattform

Programmeringsrammeverket er basert på Java versjon 9.0 og test-rammeverket er basert på jUnit4. Årsaken til valget av dette rammeverket er basert på prosjektgruppens forkunnskaper innenfor programmering og at det var det systemet vi hadde erfaring med fra tidligere emner innenfor programmering på høgskolen. Det skal gjøres oppmerksom på at det er benyttet Java versjon 9.0, selv om nyere versjon er kommet ut. Dette fordi at prototypen ble utviklet på maskiner hvor denne versjonen var installert. Ytterligere testing av systemet anbefales vi utføres i jUnit5 på grunn av at denne versjonen er mer robust og fleksibel enn jUnit4.

Systemets krav

Krav til kompetanse for utvikling

Personer som skal utvikle prosjektet videre trenger inngående kompetanse om Java og utviklet av kode for å utvikle systemet i retning av å få systemet presentert for kunden, kinoene i regionen. For å få på plass dette anbefales et grafisk brukergrensesnitt av typen JavaFX eller Java Swing for å få det presentert på en tilfredsstillende og forståelsesfull måte for kunden.
Videre er det behov for kompetanse innenfor testing og fortrinnsvis å videreutvikle testing i retning av å bruke jUnit5. Som nevnt benyttes jUnit4, men dette er grunnet vårt kunnskapsnivå og forutsetning når prosjektet ble unnfanget.

Krav og kravspesifikasjoner

Kravspesifikasjonene nedenfor er dokumentert i tabellform og er basert på Use-case diagrammet figur 6.1. De funksjonelle kravene gjenspeiler de viktigste funksjonene som systemet skal utføre. Metodenavn, beskrivelse av hva metodene gjør og hvor i selve koden testene er utført står oppført i tabellene nedenfor.

Dokumentasjon av krav vedrørende eksterne avhengigheter

I denne delen av dokumentasjonen dokumenteres eksterne avhengigheter som systemets prototype kun er utviklet en fiktiv betalingsløsning. Denne typen løsning er i denne dokumentasjonen beskrevet som en betalingsløsning hvor kortinformasjonen til kunden blir verifisert mot et eksternt system (f.eks. BankID). Prototypens stub er dokumentert i løpenummer #8.1 i denne prosjektdokumentasjonen. I kravet med samme løpenummer så kobles addTicket og checkBalance sammen slik at ved en videreutvikling av systemet kobles det godkjenning og eksterne muligheter til dette.

Det er naturlig hensiktsmessig å kunne ha mulighetene for å lagre bestillingene som gjennomføres. Det er ikke tatt høyde for en slik løsning her i denne prototypen, men vi

foreslår en databaseløsning for en sikker og skalerbar løsning på denne lagringen. Alternativt kan det benyttes en kommaseparert fil for lagring av dataene. Sistnevnte har også en sikkerhetsmessig utfordring, så database er helt klart å foretrekke.

Oversiktstabell over krav

Krav	**Løpenummer**
Selg billett kino	#01.1
Selg billett arrangement	#01.2
Avbestille billett kinoadmin	#02
Avbestille billett kunde	#03
Administrere arrangement	#04
Registrere arrangør	#05
Kontrollere billett kinoadmin	#06
Kontrollere billett arrangør	#06.1
Sjekk utvalg	#07
Kjøp billett	#08
Betalingsløsning - ekstern avhengighet	#08.1

Oversikt over hvert enkelt krav

Selg billett kino #01.1	
Aktører	Kino admin
Beskrivelse	Kino admin selger en billett for en kunde til en kinoforestilling.
Data	Ledig kinobillett og betaling fra kunde.
Stimulans	Kino admin må få betaling fra en kunde for å kunne gjennomføre et salg.
Respons	Kino admin selger en billett til en kinoforestilling og får 100 % av inntekten fra billetten.
Kommentarer	
Metodenavn	sellTicket

Selg billett arrangement #01.2	
Aktører	Kino admin

Beskrivelse	Kino admin selger en billett til et annet arrangement enn kinoforestilling.
Data	Ledig billett til arrangement og betaling fra kunde.
Stimulans	Kino admin må få betaling fra en kunde for å kunne gjennomføre et salg.
Respons	Kino admin selger en billett til et arrangement og får 20 % av inntektene fra billetten.
Kommentarer	Arrangør får 80 % av inntektene fra billetten.
Metodenavn	sellTicket

Avbestille billett kinoadmin #02	
Aktører	Kinoadmin
Beskrivelse	Kino admin avbestiller en billett for en kunde.
Data	En registrert billett som kan avbestilles.

Stimulans	Det må være kjøpt og registrert en billett for at kino admin skal kunne avbestille den.
Respons	Billetten blir lagt tilbake i listen over tilgjengelige billetter.
Kommentarer	
Metodenavn	cancelTicket

Avbestille billett kunde #03	
Aktører	Kunde
Beskrivelse	Kunden avbestiller en billett som er kjøpt.
Data	En registrert billett som kan avbestilles.
Stimulans	Kunde må ha kjøpt en billett som kan avbestilles.
Respons	Kunden får refundert beløpet som ble betalt for en billett.
Kommentarer	

Metodenavn	cancelTicket

Administrere arrangement #04	
Aktører	Kino admin
Beskrivelse	Kino admin kan opprette, endre og slette et arrangement.
Data	Navn på arrangement, dato, sted, sal, radnummer og setenummer.
Stimulans	Kino admin må opprette et arrangement ved å taste inn data. Kino admin kan endre dataen, og slette arrangementet.
Respons	Et arrangement blir opprettet i systemet, endret eller slettet fra systemet.
Kommentarer	Oppretter et objekt.
Metodenavn	

Registrere arrangør #05	
Aktører	Kino admin

Beskrivelse	Kino admin kan opprette en arrangør, som de skal selge billetter for.
Data	Navn på arrangøren og et navn på en kontaktperson.
Stimulans	Kino admin må registrere arrangør i systemet ved å skrive inn navn på arrangøren og navn på en kontaktperson.
Respons	Arrangøren blir opprettet og registrert i systemet. Kino admin kan nå selge billetter for arrangøren.
Kommentarer	Oppretter et objekt.
Metodenavn	

Kontrollere billett #06	
Aktører	Kino admin
Beskrivelse	Kino admin kontrollerer om en billett er gyldig / ubrukt. Sørger for at arrangøren får en liste over billetter som er gyldige/ugyldige slik at kontroll kan foretas.
Data	En billett som kan scannes.

Stimulans	Kino admin scanner billetten.
Respons	Kino admin får tilbakemelding om billetten er gyldig eller tidligere brukt.
Kommentarer	
Metodenavn	checkValid

Kontrollerer billett #06.1	
Aktører	Arrangør
Beskrivelse	Arrangøren av arrangementet kontrollerer om en billett er gyldig eller brukt.
Data	En billett som kan scannes.
Stimulans	Arrangøren scanner billetten.
Respons	Arrangøren får tilbakemelding om billetten er gyldig eller har tidligere blitt brukt.
Kommentarer	

Metodenavn	checkValid

Sjekker utvalg #07	
Aktører	Kunde, Kino Admin
Beskrivelse	Kunde sjekker utvalget av tilgjengelige billetter som er lagt ut av kino administratoren.
Data	Et opprettet arrangement med tilgjengelige billetter.
Stimulans	Det må eksistere et arrangement med en liste med tilgjengelige billetter.
Respons	Kunde får vist en liste over tilgjengelige billetter som kan kjøpes.
Kommentarer	
Metodenavn	event.getEventList() (event.java)

Kjøp billett #08	
Aktører	Kunde

Beskrivelse	Kunde betaler via digital betalingsløsning.
Data	Gyldig bankkort med tilstrekkelig dekning på konto.
Stimulans	Kunde må taste inn betalingsinformasjon i digital nettløsning.
Respons	Hvis kunde har dekning vil registrere en billett, og redusere saldo på konto til kunde.
Kommentarer	Stub
Metodenavn	addTicket

Kjøp billett #08.01 - Ekstern avhengighet betalingsløsning	
Aktører	Kunde, Bank
Beskrivelse	Kunde betaler via digital betalingsløsning. Banken godtar eller avviser transaksjonen.
Data	Registrert bankkort.

Stimulans	Saldo på kortets kortnummer er lik eller større enn kjøpsprisen på billetten.
Respons	Billetten tilgjengeliggjort for kunde. Oppdatering av ny saldo på bankkonto tilknyttet korttype.
Kommentarer	Stub
Metodenavn	addTicket (event.java) checkBalance (payment.java)

Ikke-funksjonelle krav

Under følger en del ikke-funksjonelle krav som prosjektgruppen vurderer som relevante i henhold til slik prototypen er utviklet slik den er nå. De ikke-funksjonelle kravene er det ikke utviklet noen tester i Java for.

Krav	**Viktighet**	**Konkret målbarhet**
Responstid på grafisk brukermiljø	Svært høy	0,1 sekund
Responstid på kontrollering av billett	Medium	2 sekunder
GDPR-lovgivning for lagring av kunde- og arrangørdata	Svært høy	Lovvedtak 54 (2017-2018)
Programmeringsspråk	Svært høy	Java
Korrekt java-versjon	Høy	Minst versjon 9.0
Testrammeverk	Svært høy	jUnit4
Integrert utviklingsmiljø (IDE)	Svært høy	IntelliJ IDEA 2018

Estimering av systemet

Estimering av funksjonelle krav nevnt i punkt Ikke-funksjonelle krav og estimatene for omfanget av utviklingen og forretningsnytten. Estimeringsmetoden som benyttes er T-shirt sizing metodikken. Denne metodikken følger en agil arbeidsmetode som gjør at vi kunne tilordne tidsrammer for de forskjellige stegene i utviklingen av prototypen. Prototypen ble utviklet med utgangspunkt i kjøp og salg av billett. Men utover ble den utvidet til å dekke forskjellige typer av kontroller. Estimatene for utviklingen av disse kontrollene dokumenteres i tabellen nedenfor. Ved å følge en agil utviklingsmetode og basere oss på iterativ utvikling unngår vi å bruke en masse tid til for-planlegging av prosjektet og heller utvikle det underveis. Først utvikles en liten del av prototypen, deretter dokumenteres dette i etterkant. Da er det naturlig å avdekke nye krav underveis som blir utviklet i prototypen, testet og dokumentert. Slik holder man på til man har utviklet en MVP (Minimum viable product) som prototypen skal utgjøre.

Estimering av krav

Krav	Utviklingstid	Viktighet
Selg billett kino #01.1	Medium	X-Large
Selg billett arrangement #01.2	Medium	X-Large
Avbestille billett kinoadmin #02	Large	Medium
Avbestille billett kunde #03	Small	Small
Administrere arrangement #04	Small	Large
Registrere arrangør #05	Small	Large
Kontrollere billett #06	Large	X-Large
Kontrollere billett #06.1	Large	X-Large
Sjekke utvalg #07	Small	Large
Kjøp billett #08	X-large	X-Large

Krav	Kommentar

Selg billett kino #01.1	Dette kravet har X-large som viktighet, fordi et av fundamentene i kjernesystemet er muligheten til å kunne selge en billett. For at en bedrift skal kunne operere og tjene penger, så må systemet ha en funksjon som gjør at man kan selge en eller flere billetter
Selg billett arrangement #01.2	Dette kravet har X-Large som viktighet, i likhet med "Selge billett kino". Dette er et av flere funksjonelle krav i systemet. Forskjellen på disse er at ved salg av billett til arrangement blir profitten delt mellom arrangør(80%) og kino(20%).
Avbestille billett kinoadmin #02	Dette kravet har Medium som viktighet. Dette er fordi vi ikke anser dette kravet som en av de viktigste funksjonene i systemet vårt. Derimot så vil vi forsøke å inkludere dette, slik at administratoren kan avbestille en billett for en kunde dersom et arrangement for eksempel blir avlyst.
Avbestille billett kunde #03	Dette kravet har Small som viktighet. Vi ønsker å implementere at en kunde skal kunne avbestille egen billett dersom det er gyldig grunnlag for det. Vi anser det som viktigere at administratoren skal kunne ha mulighet til dette, og derfor har det kravet høyere viktighet enn dette.
Administrere arrangement #04	Dette kravet har Large som viktighet. Det er viktig for kino administratoren som har opprettet arrangementet å kunne ha muligheten til å endre det i ettertid. Dette kan innebære endring av informasjon om

	arrangementet som dato, tid og tilgjengelige seter. Det gir også muligheten til å slette selve arrangementet.
Registrere arrangør #05	Dette kravet har Large som viktighet. Dette er et viktig krav, fordi en av hovedoppgavene til systemet er å selge billetter for andre arrangører. For å kunne utføre dette må kino administratoren kunne i første omgang registrere arrangøren som de skal selge billetter for, og deretter får muligheten til å opprette arrangementet til arrangøren.
Kontrollere billett kinoadmin #06	Dette kravet har X-Large som viktighet. Vi anser dette som et viktig krav, fordi det var et av kravene som var spesifikt stilt til prototypen. Kino admin skal kunne kontrollere gyldigheten av en billett.
Kontrollere billett arrangør #06.1	Dette kravet har X-Large som viktighet. Vi anser dette som et viktig krav, fordi det var et av kravene som spesifikt stilt til prototypen. Kino admin skal kunne bistå arrangør med å kontrollere billetter på arrangementer.
Sjekk utvalg #07	Dette kravet har Large som viktighet.. Vi anser dette kravet som viktig, fordi kunden som skal kjøpe en eller flere billetter må ha mulighet til å kunne velge fra et utvalg av tilgjengelige billetter. Dette innebærer ønsket forestilling, dato, klokkeslett og setenummer.

Kjøp billett #08	Dette kravet har X-Large som viktighet, på lik linje med krav #01. For at kinoen skal kunne selge billetter, må kunden ha mulighet til å kjøpe en eller flere billetter. For at systemet skal kunne generere en inntekt, så må kunden kunne via enten kinoen direkte eller via en nettløsning kunne kjøpe billetter.

Testdokumentasjon

Under følger en test-dokumentasjon av de viktigste USE-casene vi har beskrevet i kravspesifikasjonen. Testene er dokumentert med løpenummer slik at man kan finne de igjen i kravoversikten.

Løpenummer #01

Løpenummer	#01.1	#01.2
Javafil	cinema.java	
Testfil	cinemaTest.java	
Metodenavn	sellTicket	
Testnavn	setsTicketsRemaining isEventMovieEvent doesCinemaBalanceUpdateWhenSellingATicket	
Unike Tester	doesCinemaBalanceUpdateWhenIsMovieIsFalse	doesCinemaBalanceGetFullCutWhenIsMovieIsTrue

Løpenummer #02

Løpenummer	#02
Javafil	customer.java
Testfil	customerTest.java
Metodenavn	cancelTicket
Testnavn	TicketsRemainingGetsUpdated() TicketGetsRemovedFromListWhenCanceling()

Løpenummer #06

Løpenummer	#06		
Javafil	event.java		
Testfil	EventTest.java		
Metodenavn	getValidTickets	checkValid	addTicketToTicketlist
Testnavn	isTicketValid	isTicketValid	isTicketAddedToTicketlist

Løpenummer #08

Løpenummer	#08
Javafil	customer.java
Testfil	customerTest.java
Metodenavn	addTicket()
Testnavn	addTicketTest() doesBalanceUpdateWhenAddingATicket()

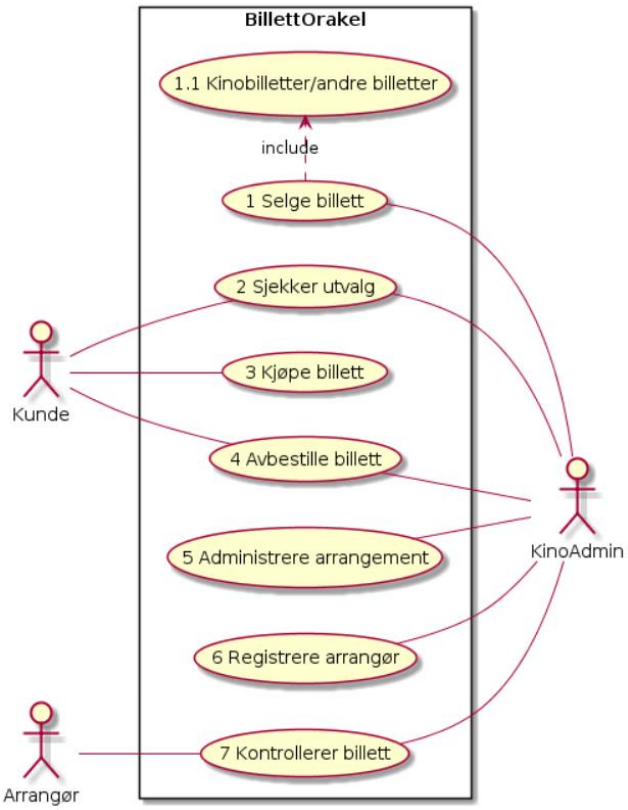
Løpenummer	#08.1
Javafil	payment.java
Testfil	paymentTest.java
Metodenavn	Det finnes ingen metoder i payment-klassen, men vi tester mot customer-klassen som bruker payment-klassen som et objekt *customerobjekt* getCurrentPayment().getBalance()
Testnavn	doesBalanceChange()

Diagrammer

Diagrammer over de viktigste funksjonene i systemet

Her i denne seksjonen dokumenteres de viktigste funksjonelle kravene (husk løpenummer): Selg billett, kjøp billett, avbestille billett og kontroller billett. Først illustreres aktivitetsdiagrammer for å få en overordnet forståelse for hva kravene gjør, deretter dokumenteres dette ytterligere i tilhørende sekvensdiagrammer. Til slutt for utviklernes del dokumenteres alle de tekniske delene av systemet i et UML-diagram slik at man kan koble det abstrakte sammen med det konkrete i koden.

Use case diagram



Figur 6.1: Use case diagram for prototypen.

Dette er use case diagrammet for prototypen til billettsystemet. Tallene på hvert use case referer til use casene i usecases.bat filen.

Personas / Use Case

Use case diagrammet vårt består av tre personas, som benytter systemet til å utføre ulike oppgaver. De tre aktørene vi har definert i modellen vår illustrert under er Kunde, Arrangør og Kinoadministrator. Disse har sine egne use cases som blir nærmere beskrevet om hva de innebærer nedenfor.

Arrangør

En arrangør har muligheten til å kontrollere gyldigheten til billetten ved inngangskontroll på arrangementet. Dette innebærer å kontrollere om en billett har tidligere blitt brukt.

KinoAdmin

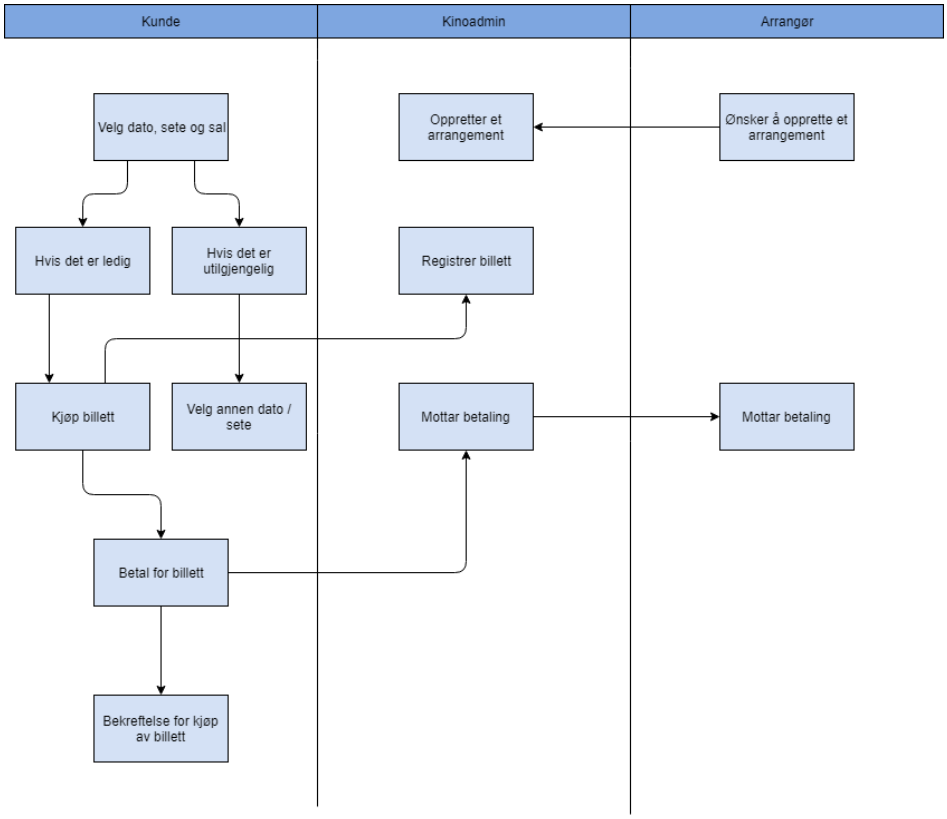
Kinoadministratoren har mulighet til å opprette en ny arrangør som de skal selge billetter for og kan også administrere selve arrangementet. Admin har også tilgang til å sjekke utvalget av billetter, samt selge en billett. Dette inkluderer også salg av andre typer billetter, for eksempel kinobillett eller en konsertbillett. Admin vil kunne i lik grad som arrangøren sjekke gyldigheten av billetten. Kinoadministratoren har også mulighet til å avbestille en billett for en kunde.

Kunde

En kunde vil ha mulighet til å kunne sjekke utvalget av tilgjengelige billetter hos de opprettede arrangementene og deretter kjøpe en eller flere billetter. Kunden har også mulighet til å avbestille billetten sin, som vil refundere pengene tilbake til kunden.

Aktivitetsdiagrammer

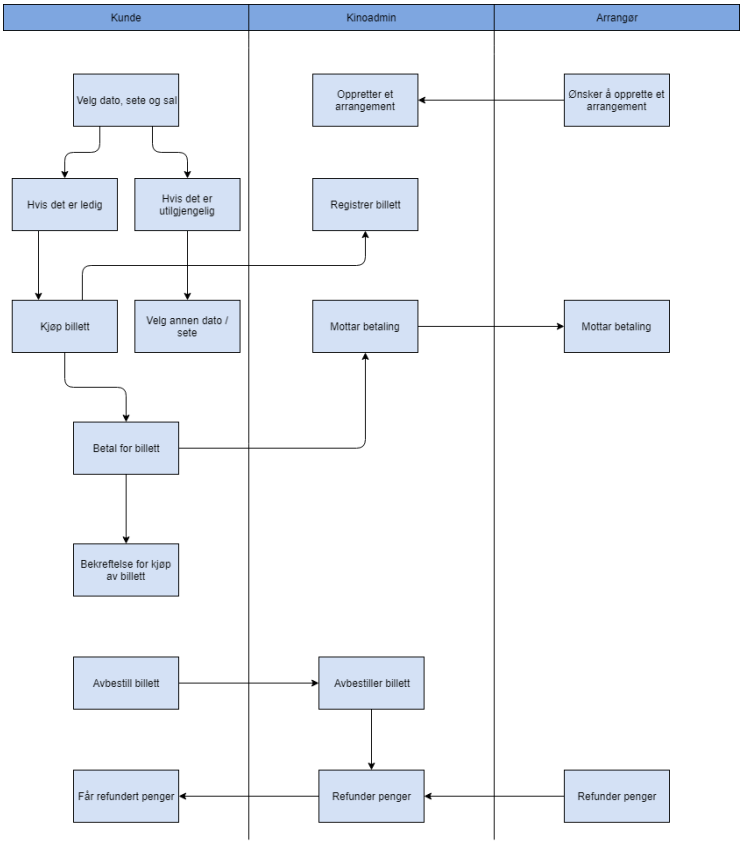
Kjøp / selg billett



Figur 6.2: Aktivitetsdiagram over use case kjøp og selg billett. Laget av: Camilla Kolstad

Dette aktivitetsdiagrammet tar for seg use case kjøp og selg billett fra figur 6.1. Diagrammet viser aktivitetene som utføres når en kunde kjøper en billett som går i tråd med salg av en billett. Den første aktiviteten som blir utført er oppretting av et arrangement som kino administrator gjør. Hvis det er en ekstern arrangør som ønsker å opprette et arrangement, så gjør kino administratoren dette. Kino administratoren oppretter også arrangementer for kinoen, for eksempel en forestilling. Når et arrangement er opprettet, så kan en kunde via et grensesnitt velge fra et utvalg av arrangementer - og deretter velge ønsket dato, sted, rad, sal og setenummer. Hvis det er tilgjengelig billett vil kunden kjøpe billetten og gå via en betalingsløsning, og til slutt få en bekreftelse for kjøp av billett. I tråd med disse aktivitetene, så vil en billett bli registrert hos kino administratoren i en liste med kjøpte billetter. Når kunden har betalt for billetten, vil kino administratoren motta en betaling. Dersom det det er en kinoforestilling vil kinoen få 100 % av inntektene. Hvis det er en annen type arrangement, vil den eksterne arrangøren få 80 % av inntektene, mens kinoen får 20 %.

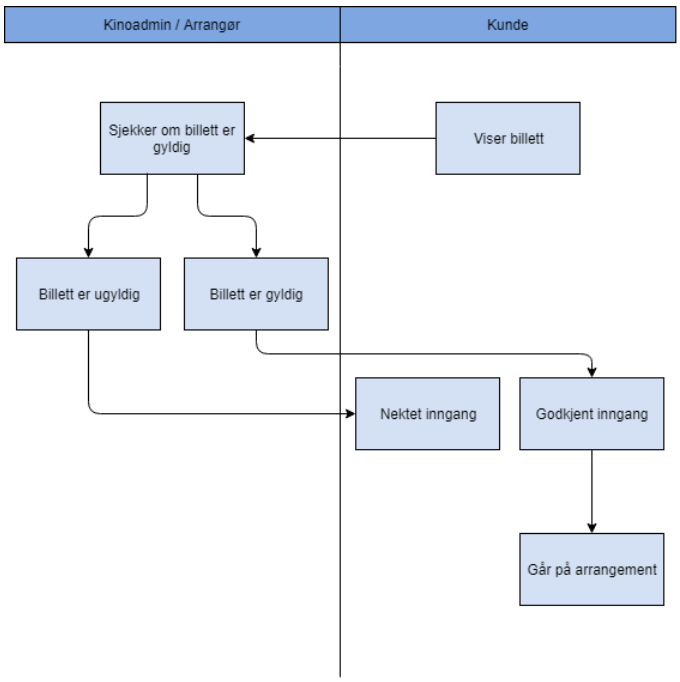
Avbestill billett



Figur 6.3: Aktivitetsdiagram over use case avbestill billett. Laget av: Geir Kirksæther

Dette aktivitetsdiagrammet tar for seg use case avbestill billett fra figur 6.3. Diagrammet tar også for seg aktivitetene som skjer på forhånd fra figur 6.2, ettersom det må ha vært et kjøp av en billett for å kunne avbestille den. Aktivitetene består av en kunde som vil avbestille en billett, eller kino administratoren som avbestiller billetten for kunden. Når kinoadministratoren avbestiller billetten via et grensesnitt, så vil de refundere pengene til kunden. Dersom det er en kinoforestilling, så vil kinoen refundere hele beløpet. Dersom det er et annet arrangement, så vil de refundere 20 %, og den eksterne arrangøren refundere 80 % av beløpet.

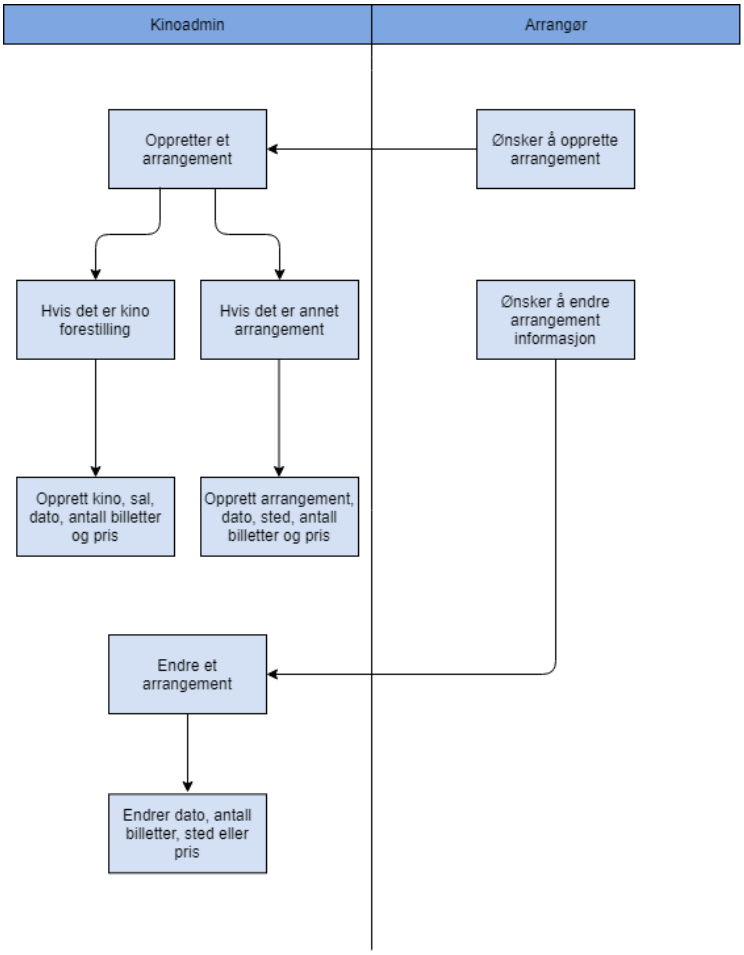
Kontroller billett



Figur 6.4: Aktivitetsdiagram over use case inngangskontroll. Laget av: Daniel Kallak

Dette aktivitetsdiagrammet tar for seg use case kontroller billett fra figur 6.1. Diagrammet viser aktivitetene som utføres når kino administratoren eller den eksterne arrangøren skal sjekke gyldigheten av en billett. Den som sjekker gyldigheten vil bruke en skanner som vil gi beskjed om billetten er brukt fra før, altså om den er ugyldig, eller om den er gyldig. Dersom billetten er gyldig vil kunden kunne gå på arrangementet. Hvis ikke, så vil kunden bli nektet inngang.

Administrer arrangement



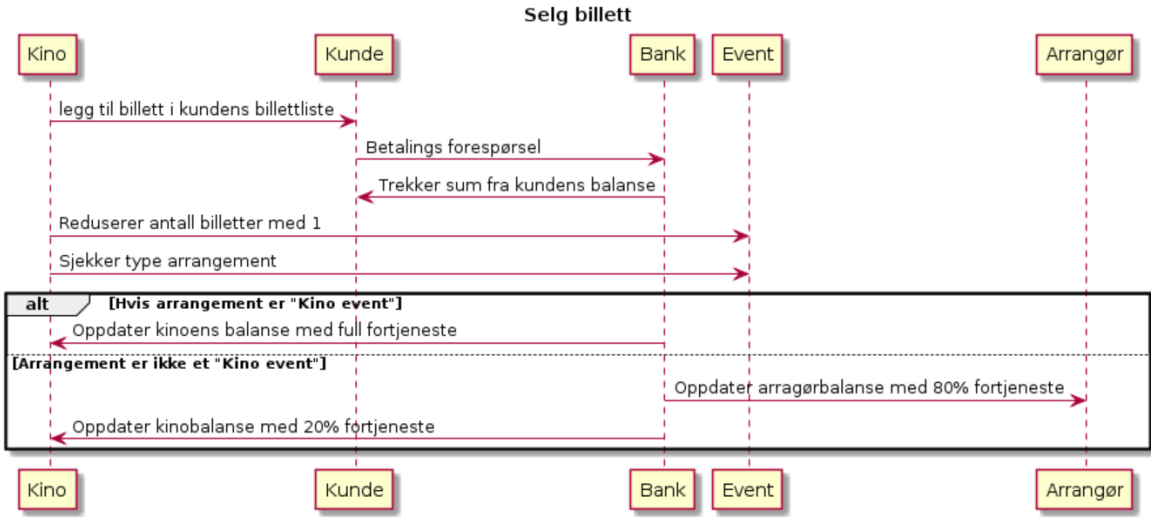
Figur 6.5: Aktivitetsdiagram over use case administrer arrangement. Laget av: Axel Grefslie

Dette aktivitetsdiagrammet tar for seg use case administrer arrangement. Diagrammet viser aktivitetene som blir utført av kino administratoren og arrangøren, når et arrangement skal opprettes eller endres. Kino administratoren kan opprette et arrangement, enten det er en kinoforestilling eller et arrangement fra en ekstern arrangør. Hvis det er en kinoforestilling, så oppretter kino administratoren et arrangement med kino, sal, dato, antall billetter og setenummer og pris. Hvis det er en annen type arrangement, så oppretter kino administratoren et arrangement, dato, sted, antall billetter og pris. Dersom kino administratoren, eller den eksterne arrangøren ønsker å endre arrangementet, kan det utføres av administratoren via et grensesnitt. Det kan da endres informasjon om dato, sted, antall billetter eller pris. Arrangementet kan også slettes.

Sekvensdiagrammer

I disse sekvensdiagrammer forklares mer inngående hva som skjer med systemet når handlingene utføres. Disse handlingene dokumenteres med løpenummeret spesifisert i de funksjonelle kravene til prototypen som tidligere er dokumentert i oversiktstabellen over krav.

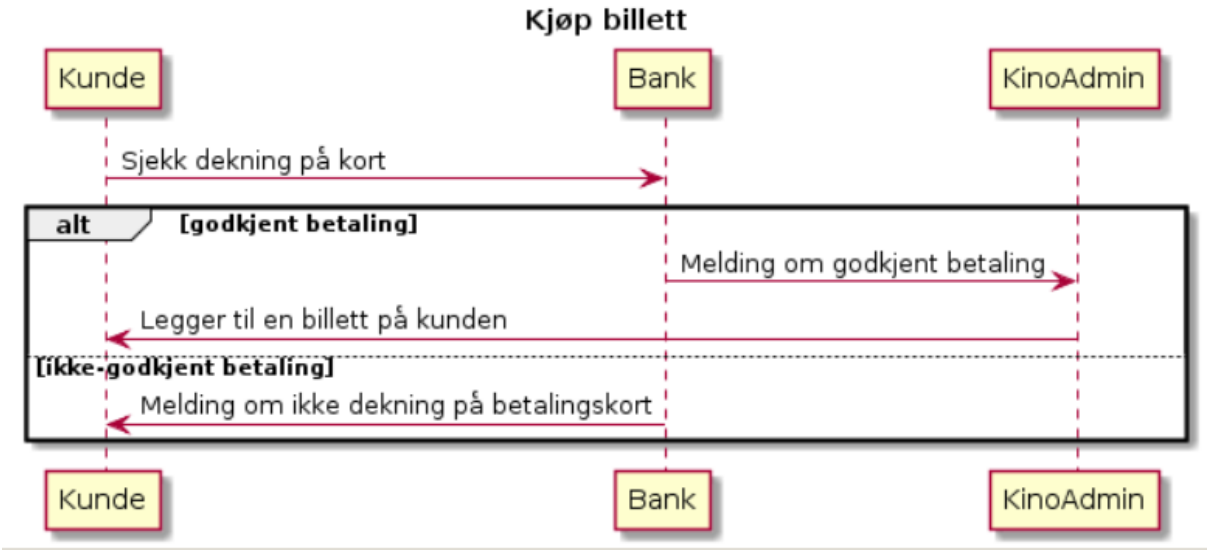
Selg billett



Figur 6.6: Sekvensdiagram for usecase selg billett. Laget av: Axel Grefslie

Overordnet er det personaen Kino Admin (Kino) som selger billettene for både sine egne kinoforestillinger og andre arrangementer. Billetten blir lagt til i kundens billettliste. Et eksempel på en slik liste visuelt for kunde vil være en "handlekurv". Som tidligere nevnt i krav #08.1 så er det ikke utviklet noe logikk for trekking av penger fra en kundes bankkonto. Men det er her dette ville blitt utført. Det blir sendt en forespørsel til (Bank) om tilgjengelig saldo er ok, deretter blir billettens pris trukket fra balansen. Et arrangement(Event) har ofte et begrenset antall billetter. (Event) reduserer antallet billetter med 1 hvis billetten er godkjent. Hvis det viser seg at billetten er av type Kino så skal (Kino) ha 100% av fortjenesten hvis det er den som har lagt ut filmforestillingen. Er det en annen arrangør som har solgt billetten og fått pengene fra (Kunde) sin (Bank) så skal 80 % av fortjenesten gå til denne arrangøren og 20 % til (Kino), siden det er de som administrerer det hele.

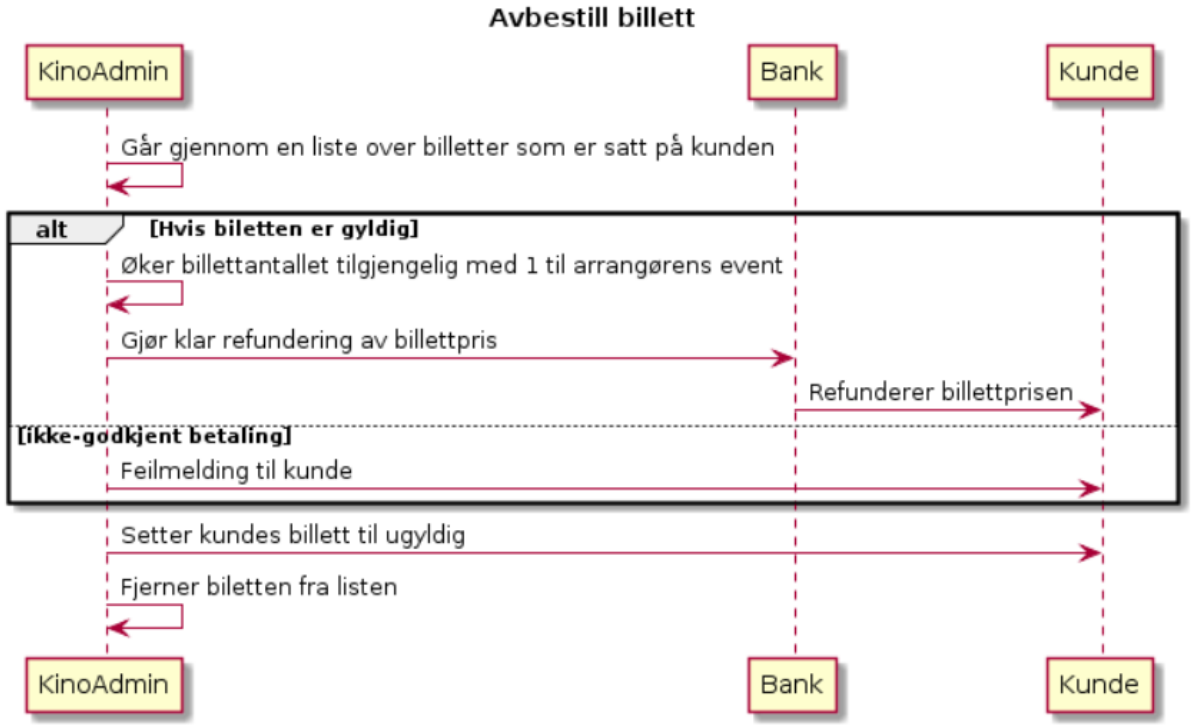
Kjøp billett



Figur 6.7: Sekvensdiagram for usecase kjøp billett. Laget av: Camilla Kolstad

Når (Kunde) utfører et billettkjøp, så vil (Bank) først sjekke om det er dekning på betalingskort før kjøpet går igjennom. Hvis det det er dekning på kortet, så vil kjøpet bli godkjent og (KinoAdmin) vil få melding om godkjent betaling. Deretter vil (Kino admin) legge til en billett på kunden. Det andre alternativet er at kjøpet ikke blir godkjent, fordi (Kunde) ikke har dekning på betalingskortet. (Kunde) vil da få en melding fra (Bank) om det ikke er dekning nok på betalingskortet. Denne sekvensen gjenspeiles i krav #08 og #08.1.

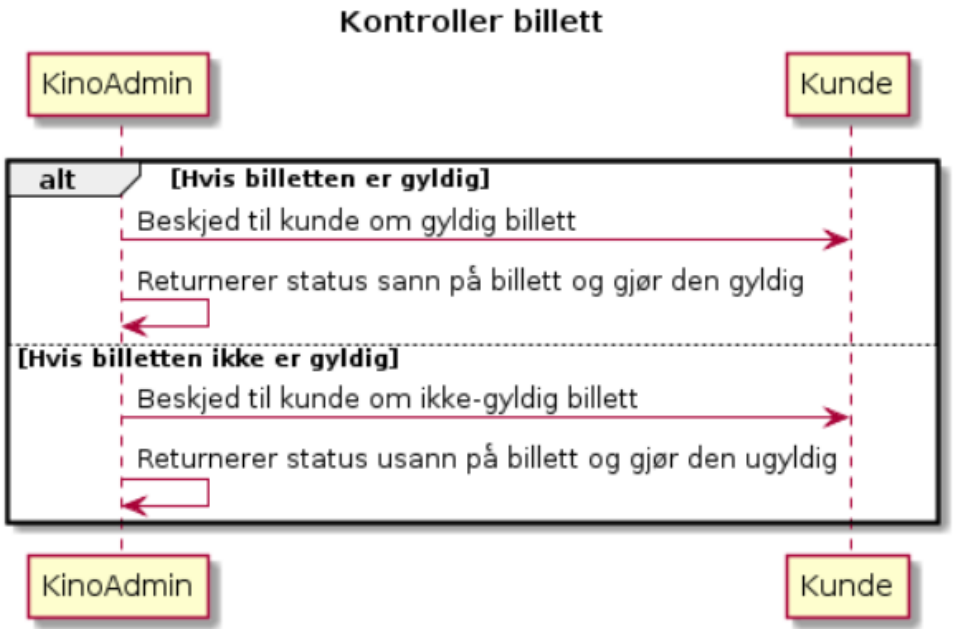
Avbestill billett



Figur 6.8: Sekvensdiagram for usecase avbestill billett arrangør. Laget av: Daniel Kallak

Dette kravet har løpenummer #02. Her er det (KinoAdmin) som er hovedaktøren og den som foretar mesteparten av handlingene. Under avbestilling av en billett rulles det gjennom en liste over billetter som er satt på den aktuelle (Kunde). Hvis billetten er kjøpt og er gyldig økes antallet tilgjengelige billetter med 1 på arrangørens event, samtidig går det en melding til kunden (Bank) om refundering. (Bank) foretar refundering av billetten til (Kunde). Hvis det ikke er lov å kansellere billetten (f.eks. hvis det ikke er refunderingsmulighet) går det en feilmelding til (Kunde). Etter dette scenarioet er satt, og kunde har fått en refundering blir billetten satt til ugyldig og billetten blir fjernet fra listen.

Kontroller billett



Figur 6.9: Sekvensdiagram for usecase kontroller billett. Laget av: Geir Kirksæther

Kontrollering av billetten henviser til kravspesifikasjon #06. I dette sekvensdiagrammet er det (Kinoadmin) som utfører selve kontrollen av billetten. Det er også gitt i kravet #06.1 at denne kontrollen kan utføres av en arrangør, som også er beskrevet om personaen arrangør og i Use-case diagrammet Kontroller billett. (Kinoadmin) utfører en kontroll av billett, typisk med en håndscanner. Når billetten scannes utføres en kontroll i systemet som sjekker om billetten er gyldig mot riktig arrangement. Er billetten gyldig får den status sann (true) og kontrollør får beskjed om billett er gyldig og gir beskjed til kunde. Er den ikke gyldig får billetten status usann(false) og beskjed til kunde om ikke-gyldig billett.

Kjente svakheter og problemer med prototypen

I dette punktet nevner vi noen kjente svakheter og problemer som er kjent for oss i prototypen. De kjente svakhetene blir listet opp med beskrivelser som gir en nærmere forklaring om hva problemet tilsvarer i prototypen.

Betalingsløsning

Det er ikke noen kobling mot en betalingsløsning. Det er kun skrevet en "stub-kode" som tar for seg den eksterne avhengigheten i kravet #08.1.

Bruker av applikasjon

Det er ikke noen kontroll av hvilken type bruker som benytter applikasjonen. Et fremtidig system må naturlig nok ta høyde for dette for å verifisere korrekt tilgang og login.

Kontroller billett

Det finnes en svakhet ved kontrollering av billett. Vi sjekker kun om billetten er gyldig eller ikke, men ikke basert på for eksempel alder, dato eller noe annet som kan sjekkes mot arrangementet.