
IT1901 Prosjektrapport

Gruppe 13

Espen Bugten
Susanne Gustavsen
Vegard Harper
Pål-André Myrland
Bjørn Tunesvik
Linn Vikre

Innhold

1	Introduksjon	3
1.1	Om faget, oppgaven og gruppen	3
1.2	Beskrivelse av prosjektet	3
1.3	Valg av prosessmetode	4
1.4	Tidsestimering	4
1.5	Ansvarsområder	5
1.6	Kunnskapsområder	5
1.7	Risiko	5
2	Arkitektur forklaring	5
2.1	MainFrame-klassen	6
2.2	CustomerWidget	6
2.3	OrderGUI	7
2.4	Kitchen	8
2.5	Delivery	9
2.6	SettingsWidget	11
2.7	Database	11
2.8	Diverse Hjelpklasser	12
3	Prosessbeskrivelse	12
3.1	Arbeidsfordeling	12
3.2	Møter	12
3.3	Sprinter	13
3.4	Kildekode deling	15
4	Implementasjon	15
4.1	MainFrame.java	15
4.2	CustomerWidget.java	15
4.3	OrderGUI.java	16
4.4	Kitchen.java	16
4.5	Delivery.java	16
4.6	SettingsWidget.java	17
4.7	(Database) DB.java	17
5	Verktøy	18
5.1	Rammeverk - QT Jambi	18
6	Forbedringer, mangler og bugs	19
6.1	Eventuelle forbedringer	19
6.2	Testing	20
6.3	Bugs og mangler	21
7	Brukermanualer	22
7.1	Ikke-teknisk brukermanual	22
7.2	Teknisk brukermanual	29

1 Introduksjon

1.1 Om faget, oppgaven og gruppen

Gruppe 13 består av Espen Bugten, Susanne Gustavsen, Vegard Harper, Pål-André Myrland, Bjørn Åge Tungesvik og Linn Vikre. Det var ikke alle som kjente hverandre fra før, men vi ble fort kjent og er alt i alt fornøyde med inndelingen.

Oppgaven vi fikk gikk ut på å lage et system til en restaurant for å registrere bestillinger som mottas per telefon. Vi var veldig fornøyd med å få en jobbrelevant oppgave. Gjennom dette prosjektet er hovedmålet å lære å samarbeide, og det å ta beslutninger i fellesskap som en gruppe.

1.2 Beskrivelse av prosjektet

Systemet vi har laget er en desktop-applikasjon i Java som er designet for en “take-out” restaurant, i vårt tilfelle en pizzarestaurant.

Eksempler på bruksområder for systemet

Systemet er lagt opp slik at brukerne av systemet kan utføre alle oppgaver. Det er lagt inn restriksjoner slik at en bruker ikke kan gjøre flere ting samtidig. Eksempelvis hvis det er få på jobb, er det lett for kokken å ta imot bestillinger, eller for brukeren å skrive ut kvitteringer. Alle deler av systemet er lett tilgjengelig. Dette kan i mange sammenhenger være nyttig, men for å utnytte dette må alle ansatte ha opplæring på hele systemet, slik at man unngår at det gjøres kritiske feil. For eksempel at en bruker feilaktig registrerer en bestilling som er ferdig. Under kommer typiske bruksområder for systemet.

1. En kunde ringer opp restauranten for å bestille middag til seg og familien. Den restaurantansatte registrerer kunden med navn, bostedsadresse, telefonnummer, etc. i systemet dersom det er en ny kunde. Eksisterer kunden fra før skal den ansatte søke opp kunden.

Bestillingen registreres ved at den ansatte henter fram og legger inn de forskjellige rettene kunden ønsker. Deretter registreres det om kunden ønsker maten levert, eller skal hente selv. Totalprisen for bestillingen regnes ut og en regning/kvittering skal kunne skrives ut. Merverdiavgift er spesifisert på regningen/kvitteringen. Det samme er et eventuelt leveringsgebyr.

2. Kokken bruker systemet til å få opp alle motatte bestillinger som enda ikke er effektivert. Når bestillingen er levert skal den ikke lenger være tilgjengelig i denne delen av systemet.
3. Nye retter kan enkelt legges inn i systemet, med navn, pris og kommentar. Kommentarfeltet brukes til å legge inn informasjon om hva produktet inneholder.
4. Systemet brukes også av personen som foretar leveringen. Når personen bruker systemet skal han enkelt få fram et kart, som viser hvor leveringsadressen befinner seg.

Kravspesifikasjon

Brukeren (personen som tar imot bestillinger) skal kunne legge inn nye kunder eller søke opp eksisterende kunder. De skal kunne legge inn bestillinger på pizza i tillegg til kommentarer på de aktuelle rettene (allergier osv). Kunden skal også kunne bestemme når maten deres skal være ferdig, så brukeren må da kunne sette dato og klokkeslett.

Kokken skal se hvilke retter som skal lages, og her også i prioritert rekkefølge etter tidspunkt. I tillegg skal kokken kunne legge til nye retter, og kunne fullføre ordre.

Sjåføren skal se bestillinger som skal leveres, og skal også få et kart med adressen til den aktuelle kundens adresse. Han skal kunne registrere at bestillingen er levert, og skrive ut kvitteringer.

I systemet skal leveringspris og grense for gratis levering enkelt kunne endres.

Tekniske spesifikasjoner

1. Det skal være en Desktop-applikasjon utviklet i Java.
2. Koden skal dokumenteres ved bruk av Javadoc.
3. Standard kodekonvensjoner for Java skal følges : Code Conventions for the Java Programming Language.
4. Lagring av data gjøres enten i en database eller i vanlige filer.

1.3 Valg av prosessmetode

Under dette prosjektet har vi valgt å benytte metodikken scrum som vi fikk en introduksjon til i oppstarten av dette kurset. Ingen av oss hadde noen erfaring med dette fra før, så det var noe vi måtte sette oss inn i.

Vi brukte scrumtavlen for å få oversikt over framgangen i prosjektet. Slik fikk vi oversikt over hva som var under prosess og hva som lå og ventet på behandling. Oversikten viste hvilke(n) oppgave(r) som måtte prioriteres til neste gang. Scrum benyttes først og fremst når man skal jobbe tett opp mot en kunde, slik at kunden hele tiden er med på prosessen. Dette har ikke vært tilfellet i dette prosjektet, så vi valgte å ikke oppdatere scrumtavlen like ofte, som vi ville gjort med en aktiv kunde.

1.4 Tidsestimering

Vi bestemte oss for å ha to ukers sprinter, og møtes en gang i uken. Vi lagde en “work breakdown structure” som delte prosjektet inn i mindre deler (*se vedlegg 1*). Dette gjorde det lettere å beregne hvor mye tid hver enkelt del ville kreve. Det ble etterhvert klart at å møte en gang i uken ikke var tilstrekkelig, så vi begynte å møtes to ganger i uken der det lot seg gjøre.

1.5 Ansvarsområder

Vi bestemte oss for at ingen skulle ha en spesiell rolle, som for eksempel programmeringsjef eller leder, i stedet tok vi et kollektivt ansvar for å få ting gjort. Vi delte likevett opp i mindre ansvarsområder, som for eksempel kontakt med studentassistenten, ansvar for database og skriving av møte referater.

1.6 Kunnskapsområder

Gruppen består av seks personer med varierende kompetanse innenfor programmering. Fem stykker med varierende kunnskap innenfor Java programmering, samt en med god erfaring innenfor C/C++. Det var heller ingen som hadde spesielt mye kunnskap innenfor database håndtering, men det var noe vi hadde vært borte i tidligere i studiet, så litt behersket vi fra før.

1.7 Risiko

Se *vedlegg 2* for risikoanalysen vi gjorde i starten av prosjektet. Prosjektet gikk forholdsvis smertefritt, bortsett fra noen få situasjoner:

Tap av kode

Mistet en del av arbeid angående den underliggende logikken i OrderGUI. Dette ble løst ved å skrive den tapte koden på nytt. Bortsett fra å sette arbeidet tilbake noen timer gjorde ikke dette noen store skader ved utviklingen av programmet.

Sykdom

Ved sykdom sørget vi for å fordele personens arbeid på de andre medlemmene i gruppen. Dette krevde en større arbeidsmengde på hver enkelt, men varighet av sykdom var kort.

Organisering av møter

Ved organisering av møter hadde vi problemer med å finne tidspunkter der alle hadde anledning til å møte. Dette ordnet vi ved å arrangere mindre møter uten alle tilstede, og kommunisere via sosiale medier, slik alle til enhver tid var klar over fremgangen i programmet. Denne løsningen sørget for en god utvikling, da det alltid var personer som kunne møte andre medlemmer ved ulike tidspunkter.

2 Arkitektur forklaring

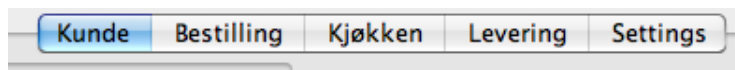
Programmet vårt er modulbasert hvor de fleste modulene er uavhengige av hverandre i direkte forstand. Modulbasert programmering sørget for en kontinuerlig testing av programmet, mens det ble kodet. Dette gjorde det enklere for alle i gruppen å jobbe på ulike deler av programmet, og teste dem uavhengig av hverandre for å oppnå hurtige resultater. Kommunikasjonen mellom

de ulike modulene (der det trengs) er dannet ved å sende signaler mellom klassene. Signalene gjør det mulig å sende ulike data mellom klassene, og utføre oppgaver ved motatt signal.

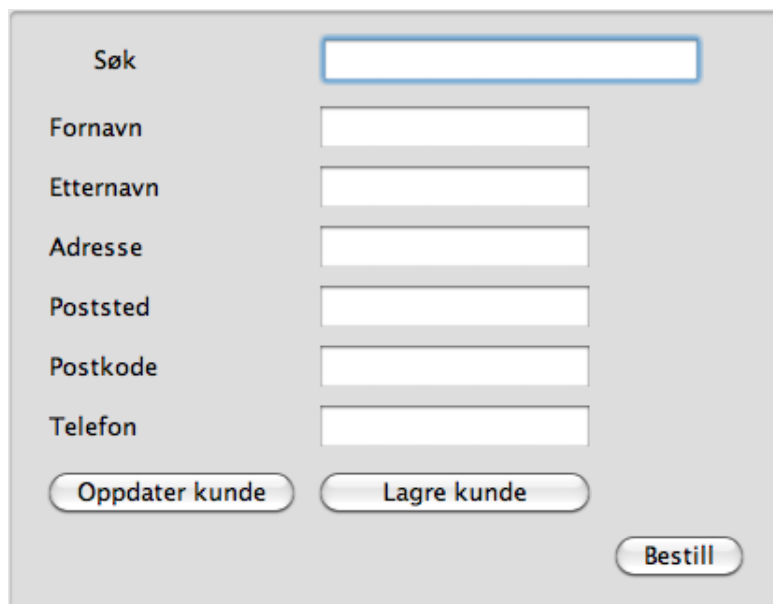
Den eneste modulen som alle de andre modulene har et direkte forhold til, er databasen, som inngår som den underliggende logikken.

2.1 MainFrame-klassen

Vi har tenkt at alle delene av bedriften blir strukturert sammen i ett vindu, dette pga at bedriften antageligvis ikke er så stor, og derfor ikke har store krav til at hver ansatt har bare en spesifikk oppgave. Kan for eksempel tenke oss at kokken må ta bestillinger i tillegg til å lage mat. Om dette er nødvendig, har han lett tilgang til å gjøre det uten å måtte forflytte seg til et annet rom eller en annen terminal. Vi har løst dette som på bildet under, og plassert hver del av programmet i hver sin tab.



2.2 CustomerWidget

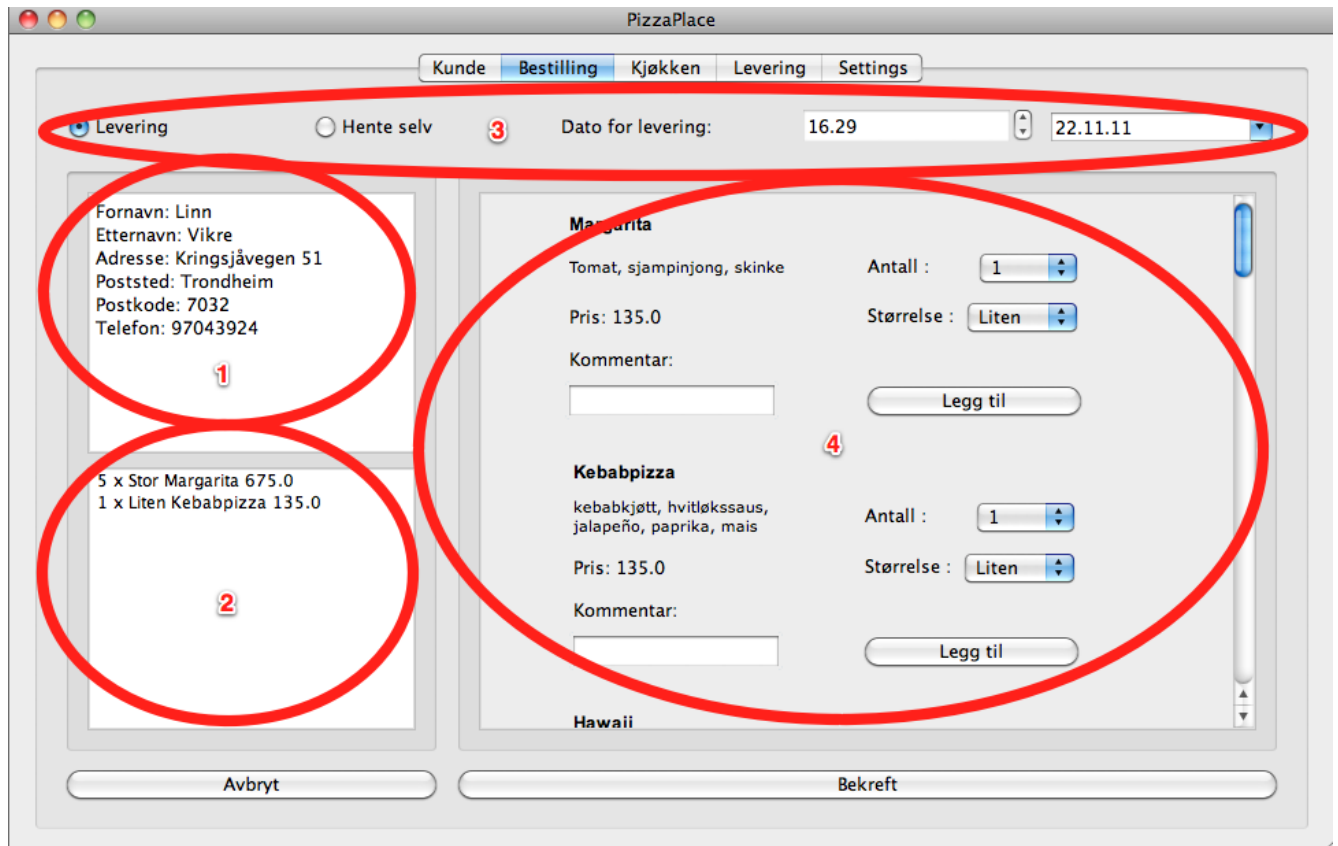


CustomerWidgeten er enkel og inneholder et søkefelt og flere tekstfelt. Hvis man finner ett treff i databasen vil all info bli lagt til i feltene. Er det flere treff i databasen kommer alternativene opp og man velger. Om man vil oppdatere info på kunden kan man bare gjøre endringen og trykke oppdater knappen. Hvis man derimot ikke finner kunden i databasen, oppretter man en ny kunde ved hjelp av lagre knappen.

For at vi skal forsikre oss om at all kundedata er rett, så har vi valgt å deaktivere OrderGUI helt til man har søkt opp eller registrert ny kunde, og trykt bestill. Når vi trykker på bestillknappen så sender vi videre customerID, som vi igjen trenger for å opprette en ordre. Vi gjør dette for

at man ikke skal kunne tukle med å legge til produkter i en ordre, uten at vi har en kunde på plass.

2.3 OrderGUI



Punkt nr. 1 på bildet over vises kundeinformasjonen for å være sikker på at det er riktig kunde vi legger produktene til. Nødvendigheten er ikke så stor, men siden man ikke har muligheten til å endre kundeinformasjonen i OrderGUI, er det mest for å være 100% Det er mulig å endre adressen til kunden etterpå. Da kan man enten fullføre bestillingen for så å endre adressen i CustomerWidget, eller eventuelt trykke avbryt, endre adressen i CustomerWidget og legge ordren inn på nytt.

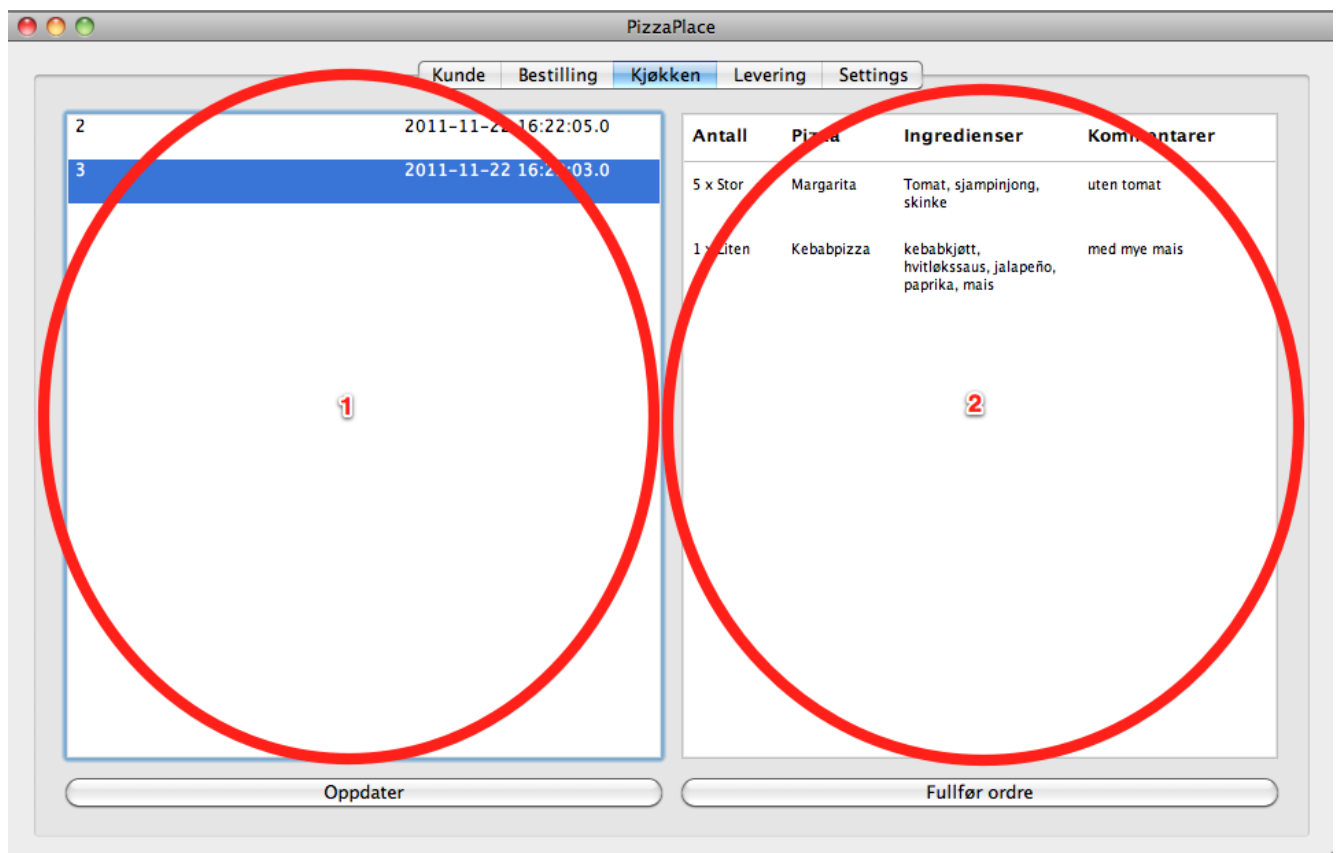
Punkt nr. 2 viser produkter som er lagt til i ordren. Ved å dobbeltklikke på et produkt fjerner du dette fra listen. Produktlisten er en samling av produkter som inneholder informasjon som skal bli lagt til i ordren i databasen. Er listen tom, når en trykker på bekreft, vil programmet respondere med en feilmelding. Dette er fordi knappen er direkte tilknyttet operasjoner mot databasen. Tomme ordre mot databasen gir ingen mening, derfor blir dette behandlet ved at programmet ikke går videre. En ordre vil først bli opprettet når brukeren bestemmer seg for å fullføre en ordre, samt at ordren inneholder produkter.

Avbryt knappen er lagt til for at brukeren skal kunne angre valget ved opprettelse av ordre, dette kan være i situasjoner ved for eksempel valg av feil kunde. Brukeren kan da lett forkaste en ordre, og bli sendt tilbake til kunde registreringen.

Punkt nr. 3 er for å legge informasjon om tidspunkt og dato for henting/levering. Når man åpner siden vil det være kryssset av for levering og tidspunktet for leveringen er satt til en time etter nåværende klokkeslett. Valget er gjort fordi man forventer at de fleste kunder som ringer inn vil ha levering og at de vil ha ordren levert så fort som mulig. Vi har satt en time som minimum. Er ikke dette tilfelle gjør brukeren de nødvendige endringer etter ønske fra kunden.

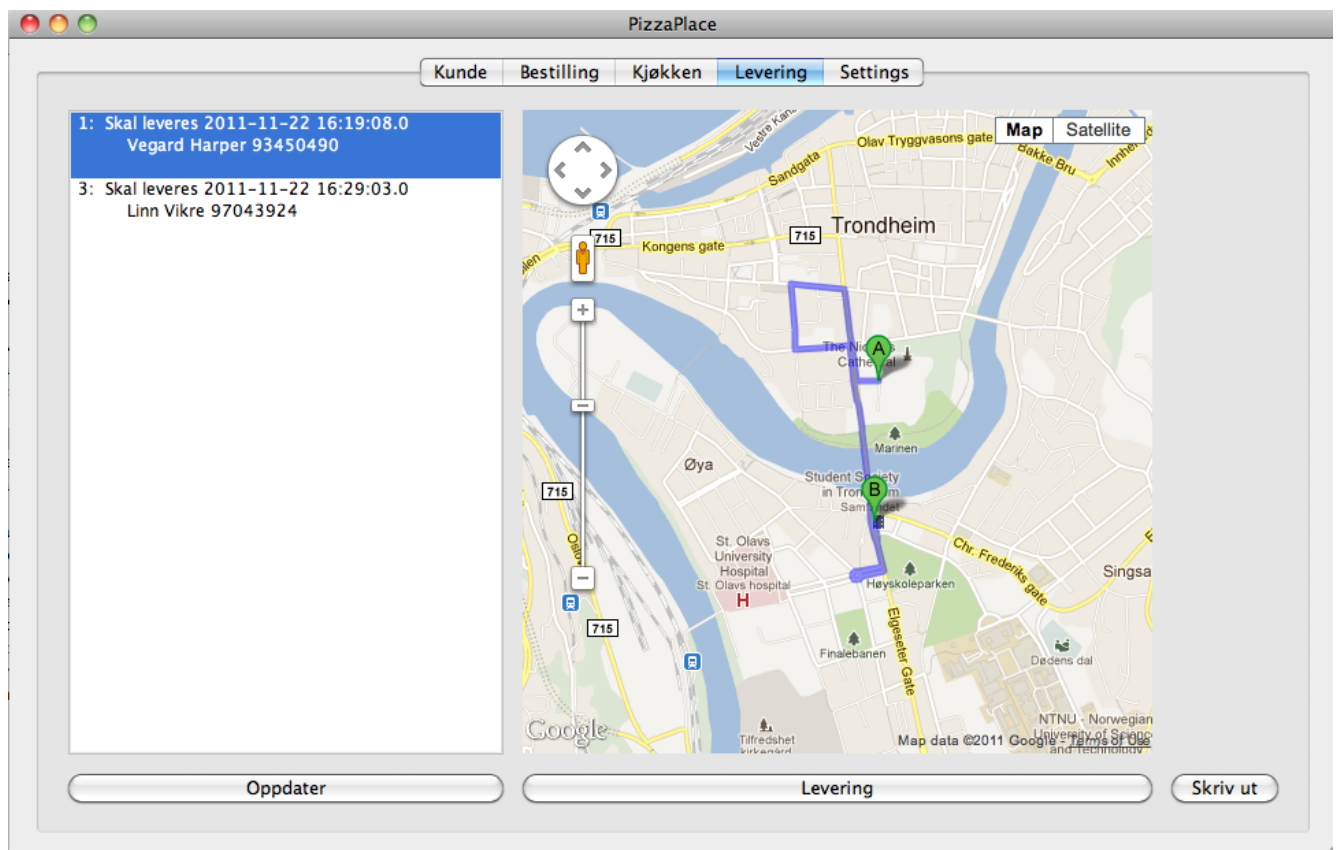
Punkt nr. 4 brukes for å velge produkter. Her kan man velge antall, størrelse og legge til kommentarer. Når dette er gjort trykker man legg til.

2.4 Kitchen



Kitchen er brukergrensesnittet for å vise ordrer som skal lages. **Punkt 1.** i figuren er listen over aktive ordrer som skal lages. Kokken trykker på ett av elementene i lista og får opp hva ordren inneholder **ipunkt 2.** Når en ordre er ferdig velger man elementet i lista og trykker fullfør ordre. Oppdater knappen har vi lagt til for at man skal kunne bruke systemet på flere forskjellige maskiner samtidig. Programmet vårt er lagt opp slik at det internt oppdaterer kjøkkenet og levering når en ordre blir lagt til, fullført eller levert. Dette ville vi ha løst på en annen måte om vi hadde hatt en server med et program til å styre alt sammen. Men på grunn av nivået på gruppa og kravet til prosjektet er det ingen nødvendighet, men det ville ha løst problemet bedre enn oppdater knappen. Problemet er løst på samme måte i Delivery klassen.

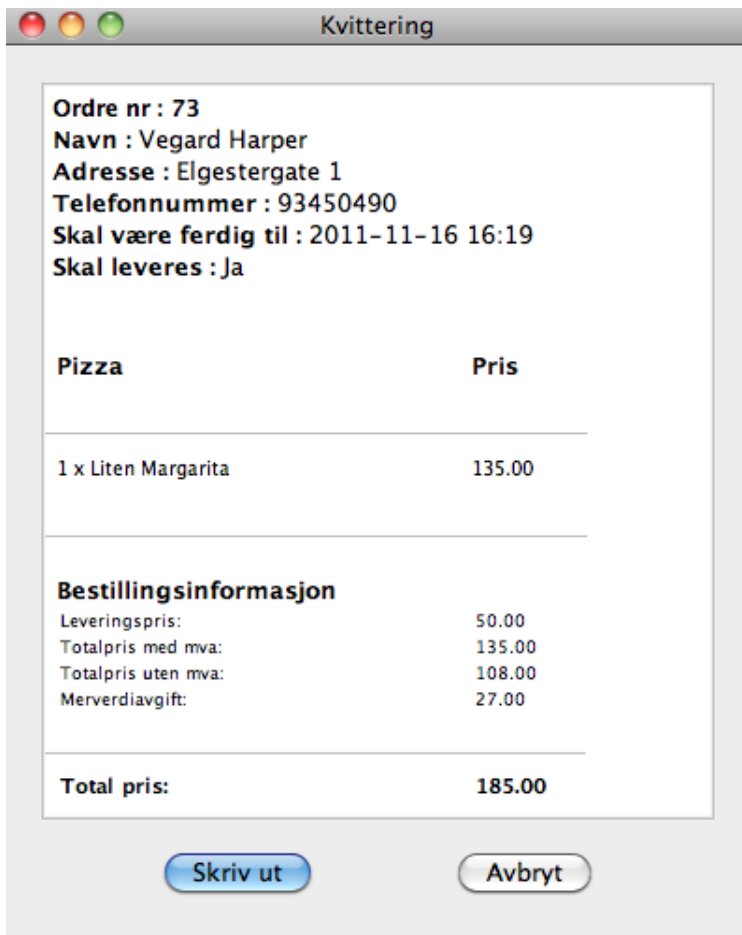
2.5 Delivery



Delivery klassen er bygd opp på samme prinsipp som kitchen. Her er vinduet delt inn i to kolonner, en liste og et webview. Første kolonne inneholder informasjon om de ordrene som skal leveres. Andre kolonne er et QWebView som inneholder et kart som viser en kjørerute mellom bedriftens- og kundens adresse.

Kartet vises ved at sjåføren velger en ordre i listen. Basert på adressen, vil en kjørerute mellom bedriftens -og kundens adresse vises på kartet. Velger kunden å hente bestillingen selv, vil informasjonen vise dette, og det vil bli vist et standard kart med bedriften i sentrum.

Brukeren har muligheten til å skrive ut en kvittering ved å markere en ordre i lista og så trykke skriv ut. Da vil det dukke opp et eget vindu med informasjon om kunde, hvilke produkter som er bestilt og totalpris.



Kvittering

Ordre nr : 73
Navn : Vegard Harper
Adresse : Elgestergate 1
Telefonnummer : 93450490
Skal være ferdig til : 2011-11-16 16:19
Skal leveres : Ja

Pizza	Pris
1 x Liten Margarita	135.00

Bestillingsinformasjon

Leveringspris:	50.00
Totalpris med mva:	135.00
Totalpris uten mva:	108.00
Merverdiavgift:	27.00

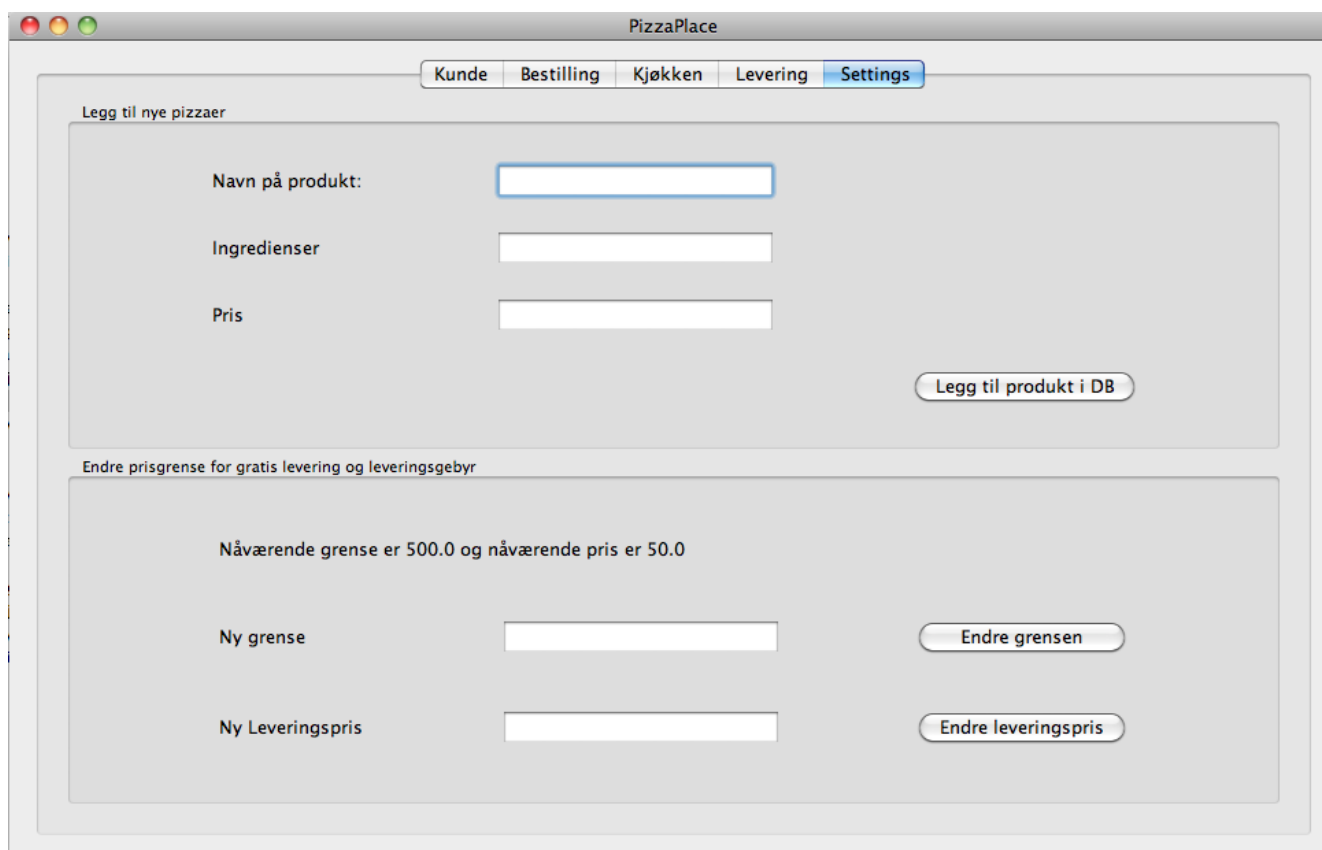
Total pris: 185.00

Skriv ut **Avbryt**

Valget av kartløsningen falt på bruk av Googles Map API, siden dette er gratis, lett å jobbe med, og de fleste på gruppa er kjent med dette produktet. Fordelen med dette kartet er at man kan få kjørerute fra bedriftens -til kundens adresse, forstørre/forminske kartet, og vise "streetview". Alt dette fordi vi har muligheten til å koble QWebViewet opp mot en nettside som kjører et kall til Google Maps, og returnerer en HTML-side som viser en rutebeskrivelse.

Ulempen er at når man laster inn Google Map i applikasjonen vår, vises ikke stedsmarkeringene i begynnelsen. De vises derimot hvis du forstørrer/forminsker kartet en gang.

2.6 SettingsWidget



SettingsWidget er en fane vi har for å legge til nye produkter, endre prisen på levering og endre grensen for hvor mye man må handle for før man får leveringen inkludert. Man slipper dermed å logge seg inn på databasen for å legge til produkter. Når det gjelder leveringsgebyr og grensen for gratis levering blir disse satt til 50 kr og 500 kr første gang man bruker programmet. Om du endrer disse verdiene blir en fil opprettet slik at neste gang du starter programmet blir denne filen lest inn og endringen vil fortsatt være der.

2.7 Database

Databasen er konstruert som en uavhengig modul i programmet. De andre modulene henter ut ønsket funksjonalitet ved å innkapsle den funksjonen de trenger. I første omgang gjør dette valget det lettere å sørge for at en applikasjon kun kan ha én tilkobling til databasen. For det andre sørger dette for at de ulike modulene i programmet enkelt kan hente ut ønsket funksjonalitet fra databasen. For det tredje, sørger dette for en lettere håndtering av eventuelle feil som måtte skje under kjøring, samt muligheten for å levere mer spesifikke feilmeldinger. For det fjerde, vil de modulene som benytter seg av databasen, ikke alltid ha de samme formateringene av data. Ved å pakke inn funksjonene sørger vi for at vi kan behandle dataen i modulene før vi sender det mot databasen for behandling. Etterhvert gikk vi inn for en standard når det gjelder innkapsling av data. Vi gikk for en modell der all data fra og til databasen blir pakket inn i array av strings. Grunnen til dette valget var at det skulle være lettere for alle i gruppen å behandle dataen og samtidig unngå forvirring, ved å følge standarden.

Som vist i vedlegget brukes databasen til å lagre informasjon om kunden samt alle ordre denne

kunden måtte ha.

Databasen støtter innsetting av data og henting av data. Den er delt inn i flere metoder, basert på hvilken informasjon de skal behandle. Grunnen til dette er at det er vanskelig å lage generiske funksjoner for å bygge SQL spørringer. På bakgrunn av dette bruker de ulike metodene statiske spørringer eller halvveis genererte spørringer hvor det lar seg gjøre. Dataene som blir prosessert blir innkapslet i henhold til en fast struktur, slik at det er klart hva som er hva. Dette sørger for en ryddig samling av data, som spesifiserer hva som ligger hvor. På bakgrunn av dette vet modulene til en hver tid hvordan de skal prosessere data fra og til databasen.

2.8 Diverse Hjelpklasser

Klassene orders, product, customer og receipt er lagd lik tabellene i databasen. Disse klassene er lik navnet og feltene på tabellene i databasen, og inneholder bare en constructor der dataen blir satt. Java-konvensjonen ble ikke fulgt, siden klassene har små bokstaver. Dette tenkte vi ikke på før mot slutten av prosjektet. Og å endre databasen ville ført til at vi måtte gjøre mange endringer rundt omkring i klassene, spesielt i spørringene mot databasen. Dette hadde vi rett og slett ikke tid til å fikse så sent i prosjektet.

3 Prosessbeskrivelse

3.1 Arbeidsfordeling

I begynnelsen av prosjektet prøvde vi å fordele arbeidet fornuftig med tanke på hver enkelts ferdigheter. Vi kom fram til at parprogrammering var en fin samarbeids/lærings metode. Vi prøvde å sette de med mest erfaring i forskjellige par, da de med mindre erfaring kunne lære mye av dette. Etterhvert fikk vi dårlig tid og ble nødt til å gå bort fra dette. På grunn av forskjellige timeplaner var det ofte mer praktisk å programmere hver for seg. Konsekvensen av dette var at arbeidsfordelingen ble noe ujevn, men alt arbeid ble dokumentert, slik at alle hadde tilgang til det.

3.2 Møter

(Se vedlegg 4 for møtetreferat og vedlegg 5 for timelister)

Vi avtalte i begynnelsen at vi skulle møtes hver tirsdag, siden vi hadde veldig ulik timeplan så var det vanskelig å møtes andre dager. Etterhvert fant vi ut at vi måtte møtes hvertfall to ganger i uka, da det ble alt for lite med å bare møtes en gang.

På hvert av disse arbeidsmøtene oppsummerte vi først hva som hadde blitt gjort siden forrige gang vi møttes. På bakgrunn av dette planla vi hva som måtte gjøres videre, i henhold til sprinten. På begynnelsen av sprint 5 gikk vi over til rapportmøter, siden vi da for alvor skulle begynne å skrive på rapporten.

3.3 Sprinter

(Se vedlegg 6)

Vi klarte å holde sprintene greit i starten, men fant fort ut at det var vanskelig å estimere hvor lang tid oppgavene faktisk kom til å ta. Noen oppgaver ble vi ferdig med før tiden og andre tok lenger tid enn planlagt. I tillegg kom vi på ting som måtte gjøres underveis, som førte til små endringer i de aktuelle sprintene vi var i. I perioder var vi også nødt til å prioritere andre fag, noe som førte til noen utsettelse og endringer i gjennomføringer av enkelte sprinter.

Sprint 1

Første sprinten inneholdt mye planlegging om hvordan vi ville organisere prosjektet og arbeidet vårt. Vi bestemte tidspunkt og dag for når vi skulle møtes å jobbe med prosjektet hver uke, vi hadde møte med studentassistenten, vi bestemte oss for hvor lange sprinter vi skulle ha, og hvordan inndelingen av prosjektet skulle være.

Vi ble raskt enige om at vi måtte ha en database i bunn, og i begynnelsen hadde vi en lokal database, men gikk senere over til NTNU sin MySQL database som vi kunne lage selv. GUIet var også viktig å få opp, slik at vi kunne ha et utgangspunkt for applikasjonen vår. Derfor begynte vi å lage skisser på dette, for å finne en løsning vi var tilfreds med. Vi satte også opp funksjonelle- og ikke funksjonelle krav som applikasjonen skulle tilfredstille.

Vi valgte også å benytte et rammeverk som heter QT Jambi istedet for Java Swing. Skisser på hvordan vi ville at programmet skulle se ut ble laget, og vi listet opp noen av funksjonene programmet skulle inneholde.

Susanne dro også på scrum-kurs den 14.september for å lære scrum-prosessen bedre og videreformidle dette til resten av gruppen.

Sprint 2

I denne sprinten fortsatte vi på å bygge databasen, da dette tok litt lengre tid enn vi hadde planlagt. Vi fortsatte også å se på QT Jambi, og fant ut at vi kunne bruke den innebygde designeren til å lage skisser til GUIen. Det viste seg at designeren genererte kjørbare kode samtidig, men vi valgte å la være å bruke denne koden siden det ble utrolig rotete.

Vi bestemte oss for å begynne å parprogrammere, så vi delte oss i par; Vegard og Espen begynte med MainFrame GUIet, Susanne og Linn begynte med å lage logikken og GUIet for å bestille pizza i OrderGUI, mens Bjørn og Pål-André programmerte videre på Database-klassen.

Vi fikk også laget et ER-diagram som utgangspunkt for databasen, samt laget risiko-analysen slik at vi hadde det i orden.

Sprint 3

Vi fant ut at vi måtte ha en SCRUM-tavle lett tilgjengelig for å kunne se hva alle gjorde, hva som måtte gjøres, hva som var i prosess og hva som var ferdig. Derfor benyttet vi oss

av Googledoc, slik at alle hadde tilgang til dokumentet til en hver tid og alle kunne redigere dokumentet samtidig.

Siden de oppgavene vi satte opp på forrige sprint var såpass omfattende, regnet vi med at de måtte fortsettes på også i denne sprinten, noe som stemte bra siden de var såpass generelle. Videre så fikk vi tid til å implementere deler av leveringsGUIet, med kart.

I denne sprinten hadde vi også midtveispresentasjon som vi måtte forberede oss litt til, så litt av tiden gikk til det. Første prioritet gikk på å få et sammensatt program som fungerte å gå igjennom uten at vi brydde oss om logikken i bakgrunn. Det var altså GUIen som var viktigst å få på plass.

Sprint 4

I denne sprinten begynte vi å bli så og si ferdig med hoveddelene av programmet og det eneste vi manglet var å kunne skrive ut kvitteringen. Men det skrev vi forholdsvis raskt og fikk implementert det i programmet vårt. Vi hadde fortsatt litt igjen som måtte fikses på programmet, men dette så vi på som mindre endringer som vi kunne ta mens vi begynte på rapporten.

Vi begynte derfor å sette opp hva vi skulle ha med i rapporten, hvor vi fordelte punktvis hvem som begynte å skrive om hva, slik at det ikke var noen som skrev mye mer enn alle andre. Vi planla ett møte med student assistenten for å få litt tips til hvordan vi skulle skrive rapporten.

Sprint 5

Vi klarte så og si å holde sprinten fra forrige gang. Vi var ferdig med de store programmeringsdelene av prosjektet. Og nå var det bare litt småting som manglet.

Vi fortsatte å diskutere hvilke punkter vi skulle ha med i rapporten og fikk en bedre oversikt over hva som allerede var skrevet og hva som manglet. Deretter satt vi opp en plan over hva som måtte skrives, slik at vi kunne ta på oss det som manglet å skrive. Dette satt vi opp i google dokumentet vårt.

Studentassistenten var ikke tilgjengelig for å hjelpe oss med oppsettet av rapporten. Vi hadde derfor et møte med den vitenskaplige assistenten, for å få litt hjelp med hvordan vi skulle strukturere rapporten og litt klarheter i noen ting som skulle være med i rapporten.

Sprint 6

I denne sprinten har vi hovedsaklig jobbet med rapporten, samt gått igjennom programmet vårt for å fjerne litt unødvendig kode og gjøre de siste testene av programmet. Vi laget også de diagrammene vi manglet som vi skulle ha med i rapporten, siden det var mange og vi visste at det kom til å ta såpass mye tid å lage dem.

3.4 Kildekode deling

I begynnelsen av prosjektet fikk vi vite at vi skulle få tildelt et filområde slik at vi kunne oppbevare all kode og materiale til prosjektet der. Men det ble ikke tilfellet, derfor valgte vi å bruke GitHub til å dele koden.

GitHub er veldig greit å bruke når flere brukere skal skrive på samme kode. Man oppretter et “repository”, et prosjekt, som man kan dele med flere og alle kan da redigere og oppdatere til nye endringer som blir gjort i koden. Dette fungerte veldig bra for vår del, bortsett fra at vi hadde litt problemer i starten med å sette opp Git på de forskjellige maskinene våre.

4 Implementasjon

(Se vedlegg 11 for klassediagram og vedlegg 10 for usecase diagram)

4.1 MainFrame.java

MainFrame klassen er ansvarlig for å opprette alle instanser av modulene som kreves av programmet. Den gjør alle moduler operative, og oppretter kommunikasjon mellom dem der det kreves. Kommunikasjonen mellom dem blir brukt til å sende over signaler, slik at mottakeren kan utføre oppgaver ved mottatt signal. I denne klassen handler det i første omgang om å overføre kontrollen over til andre moduler. Det skal imidlertid nevnes at denne klassen også styrer linker mellom andre moduler. Dvs. at linken blir dannet her. Grunnen til dette er at modulene (objektene), som skal linkes, må være tilgjengelig i samme fil for å opprette en link.

Databasen, som også er en modul, blir også startet her. Databasens funksjonalitet blir gjort tilgjengelig ved å sende ved referansen med i alle moduler som krever denne. Dette gjør at databasen er lett tilgjengelig innenfor alle modulene.

4.2 CustomerWidget.java

(Se vedlegg 7 for flow-chart)

CustomerWidget er implementert for å støtte søk, lagring, og oppdatering av kunder. Den bruker i stor grad databasen som underliggende logikk for å utføre dette. Lagring av en kunde blir utført ved å hente ut kundedata fra GUIet, og pakke dette sammen i et string array. Data blir sendt til databasen ved at en “wrapper”-funksjon blir kalt.

Oppdatering av kunde er bygget opp med det samme prinsippet. Siden vi bruker en mellomlagret kunde-id, så kan vi bruke denne til å bygge opp en query mot databasen. Selve operasjonen mot databasen følger det samme prinsippet som ovenfor. Data blir hentet fra GUIet, for så å bli sendt til databasen ved hjelp av en “wrapper”-funksjon.

Søk er implementert noe annerledes. Konseptet med event er der bygd opp på en helt annen måte. Hver gang teksten endrer seg vil det bli sendt et signal som resulterer i et kall mot

databasens søkefunksjon. Resultatene vil så bli listet opp i en event-basert liste under søkefeltet, hvor man kan velge kunde ved å trykke på det korrekte valget.

4.3 OrderGUI.java

(Se vedlegg 7 for flow-chart)

Denne klassen skiller seg noe ut, med at den i stor grad inneholder en egen logikk del. Den bruker fortsatt databasen, men den behandler/lagrer mindre deler data internt før den benytter seg av databasen. OrderGUI benytter seg av to hjelpe klasser, Pizza og PizzaList, for å behandle data. OrderGUI'et oppretter et PizzaList objekt, og PizzaList henter data om alle produktene fra databasen og lager en liste i form av et scrollArea. Dette gjør at man bare trenger å skifte ut lista hver gang man legger til et nytt produkt. Pizzaobjektet sender ut signal med all data om produktet til OrderGUI'et, som igjen behandler mottatt data.

Videre blir mottatt data lagt inn i en produktliste, samt i en liste som opptrer som et speilbilde av denne listen. Forskjellen mellom dem er at den originale listen viser en formatert tekst over produkter, mens den speilede listen inneholder all faktisk informasjon. Programmet oppretter først en ordre i databasen, deretter går programmet igjennom den speilede listen og setter inn produktene til ordenen.

4.4 Kitchen.java

(Se vedlegg 8 for flow-chart)

Kitchen er implementert for å hente ut informasjon om orderene fra databasen. Informasjonen blir lagret i en liste som inneholder en formatert tekst om ordenen, mens den speilede listen inneholder all informasjon om den faktiske ordenen.

Når man trykker på et element i listen, vil programmet hente produktene som skal lages til ordenen. Teksten blir formatert og vist i vinduet til høyre med mer informasjon om produktet som skal lages. Når en ordre er ferdig så henter programmet ut orderID fra den speilede lista og oppdaterer ordenen i databasen til fullført.

4.5 Delivery.java

(Se vedlegg 9 for flow-chart)

Ved å klikke på en leveranse i listen får man opp leveringsadressen på kartet til høyre. For å hente ut kart i Google Maps må man bruke en "http-request", hvor man sender inn adressen der sjåføren er og adressen han skal til. Det kreves et spesielt format på adressene, hvor man tar hensyn for "æ, ø og å", som vi har tatt hensyn til ved å bytte ut disse bokstavene med "ae, oe og aa". Når vi da henter kartet fra "http-requesten" legger vi dette på et QWebView.

Delivery-klassen er avhengig av PrintReciept som generer all funksjonalitet for å skrive ut kvittering. PrintReciept arver fra klassen TheReciept. The Reciept setter opp en enkel layout bestående hovedsaklig av tekst og knapper. Knappene er koblet til to metoder som arves fra QDialog, hvor vi overskriver metoden accept() for å ha mulighet til å skrive ut kvitteringen.

4.6 SettingsWidget.java

I denne klassen har vi satt stortsett tenkt på funksjonalitet, så denne var satt sammen for å lett kunne endre verdier og legge inn produkter. Nye produkter kan enkelt bli lagt inn i databasen og vises i produktlista ved ordrebestilling ved å legge inn informasjon og navn, ingredienser og pris og deretter trykke Legg til i DB". Produktet vil da vises som siste element i lista over produkter i bestillingsvinduet.

Når det gjelder leveringsgebyr og grensen for gratis levering blir disse satt til 50 kr og 500 kr første gang man bruker programmet. Om du endrer disse verdiene blir en fil opprettet slik at neste gang du starter programmet blir denne filen lest inn og endringen vil fortsatt være der. Hvi du velger å endre verdiene igjen vil de gamle verdiene bli overskrevet, og de nye blir gjeldende.

4.7 (Database) DB.java

(Se vedlegg 12 for ER-diagram)

Som nevnt er databasen bare en rekke med "getter"-og "setter"-metoder mot databasen. Disse metodene blir "wrappet" inn i modulene hvor de skal brukes. De bruker Java sql driveren for å kontakte databasen. Dette API'et støtter diverse metoder for å sende spørringer mot databasen. Spørringene er for det meste statiske, det vil si at de blir bygd for en bestemt situasjon.

Det er imidlertid to metoder som ikke følger dette mønsteret, det er metodene Search() og insert(). Disse metodene baserer seg på input for å determinere hvilke spørringer de skal bygge opp. Insert()-metoden bruker klasser som "templates" for innsetting i databasen. Feltene i hjelpeklassene gjenspeiler de nøyaktige navnene på feltene i tabellene. Instansene av klassene blir brukt for å bygge opp spørringene automatisk, ved bruk av "Java reflection".

Search()-metoden er noe mer statisk enn dette, ved hjelp av hjelpefunksjonen determineType() bygger den betingelsesdelen av spørringen ved å studere funksjonens input. Basert på input kan den bygge opp søk etter: nummer, fornavn, etternavn, eller fornavn og etternavn.

Resten av metodene i databasen er bygd opp med statiske spørringer, som ikke i særlig grad avhenger av input.

5 Verktøy

Noen av verktøyene vi har brukt for å fullføre prosjektet er listet under:

- Eclipse Helios og Indigo, som texteditor for programmeringen samt generering av Java-doc.
- yEd graph Editor, for ER-diagram og usecase diagram.
- LucidChart, for å lage flowcharts.
- Qt Jambi 4.5.2_01, rammeverket vi brukte for å kode prosjektet.

5.1 Rammeverk - QT Jambi

I dette prosjektet valgte vi å bruke et rammeverk som heter QT Jambi i stedet for Java sin Swing. Dette valgte vi både fordi ingen på gruppen vår hadde noen særlig erfaring med Swing programmering fra før av, men også fordi QT Jambi har en egen designer som vi kunne bruke til å lage skisser til hvordan vi ville at de forskjellige delene av programmet skulle se ut. Designeren til QT Jambi genererer også Java kode som kan kjøres, men koden ble veldig rotete så vi valgte å skrive koden selv. I startfasen av prosjektet var designeren veldig nyttig da vi kunne få laget flere skisser som så bra ut, og ikke håndtegnede skisser hvor det kan være vanskelig å se hva ting egentlig er.

QT jambi har en del effektive hjelpemidler som "signals" og "slots". Dette gjør at kommunikasjonen mellom forskjellige objekter blir lettere. For eksempel når man trykker på en knapp så blir det sendt et signal til et annet objekt med slot som er koblet til signalet, dermed mottar objektet et signal om at det har skjedd en endring. Dette synes vi er enklere enn måten det gjøres på i Swing, som benytter seg av "call-back"-funksjoner.

På en annen side var det ikke like lett å generere en runnable jar fil som kunne kjøre på alle operativsystem, siden hvert operativsystem krever forskjellige bibliotek. Det viste seg også at vi måtte være veldig forsiktige med skaleringen av programmet, da det ble store forskjeller i for eksempel Windows i forhold til Mac OS X. Dette var noe vi ikke forutså i begynnelsen og måtte derfor ta hensyn til dette når vi lagde programmet vårt.

6 Forbedringer, mangler og bugs

6.1 Eventuelle forbedringer

- En mulig forbedring kunne vært å ha hurtigtaster. Det ville gjort det enklere og raskere å bruke systemet. Dette var noe vi diskuterte underveis, men ikke fikk tid til å implementere.
- En annen forbedring kunne vært om kokken hadde mulighet til å markere en ordre som påbegynt. Da kunne for eksempel en påbegynt ordre vært vist med en annen farge i listen over ordrene. Dette ville gjort det enklere for kokken å holde oversikt i tilfeller der han/hun jobber med flere ordrer samtidig. Slik vi har det nå kan kokken bare fullføre ordrene. Om systemet faktisk skulle brukes, ville vi nok gjort dette annerledes, da det ikke er veldig hygienisk at kokken skal trykke på en datamaskin. Under leveringen ville vi hatt en lignende funksjonalitet, men der ville det vært om ordren er under utkjøring eller ikke.
- Databasen er ikke konstruert for å støtte skalering, det vil si at flere bedrifter bruker det samme programmet. Programmet og databasen klarer bare å håndtere en ordre om gangen, på grunn av måten vi har konstruert spørringen for å hente ut ordre nummeret. Hvis programmet skal støtte skalering må dette skrives om. Det skal imidlertid tid nevnes at dette ikke er et problem, siden programmet vårt i utgangspunktet er skrevet for en liten bedrift.
- Vi burde gjort alle metodene i DB-klassen til statiske metoder. Slik vi har det nå med å opprette en instans av DB klassen er grisete og løser det ikke elegant. Hadde metodene vært statiske kunne vi sluppet å sende referansen rundt omkring i programmet. Dette var rett og slett noe vi ikke tenkte på tidlig i utviklingen av programmet.
- Et innloggingssystem kan være lurt å ha, blant annet av sikkerhetsmessige årsaker, for å holde uvedkommende ute av systemet. Men det hadde også åpnet muligheten for å begrense tilgangen til enkelte deler av systemet. Det kan for eksempel tenkes at man ikke vil at alle ansatte skal kunne endre leveringspriser, legge til produkter osv. Men som vi argumenterte for tidligere kan det være praktisk i noen sammenhenger at alle har tilgang til hele systemet.

6.2 Testing

Vi har gjort regelmessig testing av programmet underveis. Vi har testet ut mange forskjellige input data, slik at vi får sjekket om programmet gir riktig output og/eller om det takler feil input. Vi har fått personer utenfor vår gruppe til å teste programmet, og har på denne måten også fått testet brukervennlighet og funnet noen feil og lite praktiske løsninger. Mange av de fikk vi rettet opp, men enkelte hadde vi ikke tid til fikse fordi de ble oppdaget for sent.

Noen problemer vi fikk fikset:

1. (a) Når man skulle lage en ny kunde og trykte lagre, kunne man ikke gå videre til bestilling. Da ville ingen kundeinformasjon dukke opp. Men brukeren ble lagt inn i databasen og kunne søkes på og deretter gå til bestilling.
(b) Ble fikset ved å ikke tømme feltene når man lagrer kunden.
2. (a) I SettingsWidget kunne man skrive inn bokstaver der man krevde tall, noe som førte til at programmet kastet exceptions.
(b) Ble løst ved at vi ga brukeren tilbakemeldinger på hva som var feil. De som testet dette for oss kunne ikke vite hva som var feil.
3. (a) Kommentarer og antall var ikke resatt mellom to forskjellige bestillinger.
(b) Tømme feltene etter hver fullførte bestilling.
4. (a) Når man trykket avbryt etter å ha lagt til produkter i bestillingsvinduet. Så ville neste kunde som skulle bestille ha de samme produktene i sin liste.
(b) Dette ble fikset ved å resette dette feltet på samme måte som når man trykker på "Bekreft"-knappen.
5. (a) Når man trykket avbryt etter å ha lagt til produkter. Så ville neste kunde ha de samme produktene i sin liste.
(b) Ble fikset ved å legge til at listen ble resatt på samme måte som når man trykker på "Bekreft".
6. (a) Når vi hadde flere kunder med samme navn i databasen og søkte på navnet, ble alltid den første kunden valgt. Oppdaterte vi kunden, så ble alle kunder med samme navn oppdatert.
(b) Dette løste vi ved at vi fikk opp en liste der man kunne velge kunde basert på fornavn etternavn og telefonnummer. Da ble også et unikt telefonnummer mye viktigere.
7. (a) Når vi lukket programmet uten at det lå produkter i bestillingen, ble det kastet exceptions. Det samme skjedde også hvis man prøvde å vise hva som skulle lages eller prøvde å skrive ut en kvittering. Dette var fordi programmet satt inn en tom ordre i databasen hver gang man trykket på bestill, etter å ha søkt opp en kunde. Denne blir ikke slettet om du lukker programmet, men når du trykker avbryt.
(b) Vi fikset dette ved å endre når ordren blir satt inn i databasen. Nå vil ordren bli opprettet når brukeren trykker "Bekreft" i bestillingsvinduet.
8. (a) Når vi la til et produkt i OrderGUI'et med større antall enn 1 så burde prisen blitt multiplisert sammen. Man legger for eksempel til 2 margarita til 135 kroner per stykk, da burde ordrelista vise 2 margarita 270 kr, og ikke 2 margarita 135 kr.

- (b) Formaterte teksten som ble satt inn i lista.

6.3 Bugs og mangler

- Bestillingstidspunktet som vi skal sette i databasen er av typen `current_time`. Det er et problem hvis har man lyst til å vite når kunden bestilte(refundering, klaging osv). Problemet er at dette tidspunktet fungerer som en klokke som angir tiden da man sist fikk resultatet fra databasen.
- Under Windows med nyeste Java versjon(1.7) ville ikke programmet kompileres. Vi fikk feilmelding om at det var noe som ikke matchet, selv om det gjorde det. Det fungerer på Linux, Mac og Windows med Java 1.6 installert.
- Vi fant ut at problemet med at kommentaren og antall ikke ble resatt når man fullførte bestillingen, også er tilfelle når man trykker avbryt. Dette viste seg å være vanskeligere å fikse, man i dette tilfellet må iterere over hele produktlista for å tømme feltene.
- I programmet har vi ikke vært konsekvent nok med navn på variabler og klasser. Et eksempel er `Pizza.java` som egentlig burde hatt navnet `Product.java`, Det er fordi vi bruker pizza til å lagre alle typer med produkter, som for eksempel leskedrikker og dressinger. Derfor hadde vært bedre om pizza var en subklasse av product der man kunne ha definert spesielle egenskaper til pizza.
- Hvis man trykker lagre på en kunde som allerede eksisterer i databasen og som er søkt opp, vil man få duplikat i databasen. Endrer man informasjonen på en av de med samme fornavn, etternavn og telefonnummer, vil man endre informasjonen på alle personene som hadde samme informasjon. Dette er på grunn av at vi søker i databasen etter fornavn, etternavn og telefonnummer, hvor vi burde inkludert kundenummer.

7 Brukermanualer

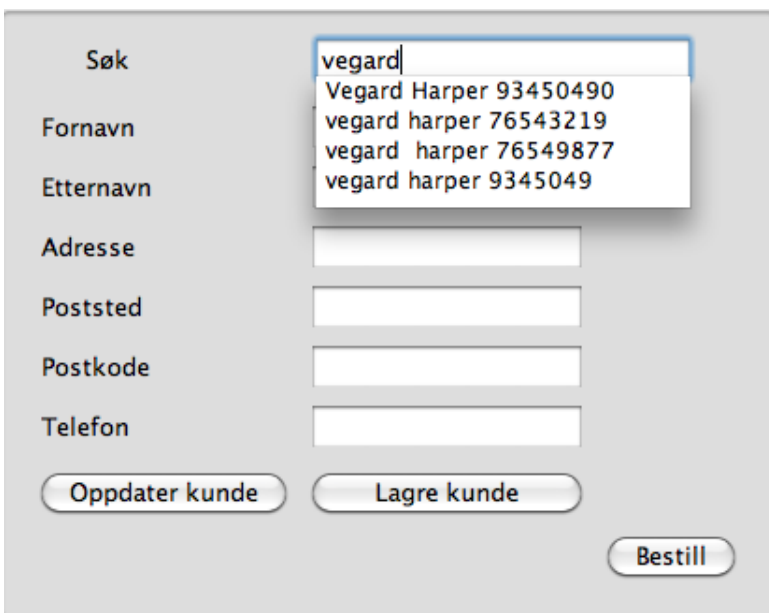
7.1 Ikke-teknisk brukermanual

Kunderegistrering



A screenshot of a web form for customer registration. The form has a light gray background. At the top left is a label 'Søk' next to a white search input field. Below this are seven more input fields, each with a label to its left: 'Fornavn', 'Etternavn', 'Adresse', 'Poststed', 'Postkode', and 'Telefon'. At the bottom left are two buttons: 'Oppdater kunde' and 'Lagre kunde'. At the bottom right is a button labeled 'Bestill'.

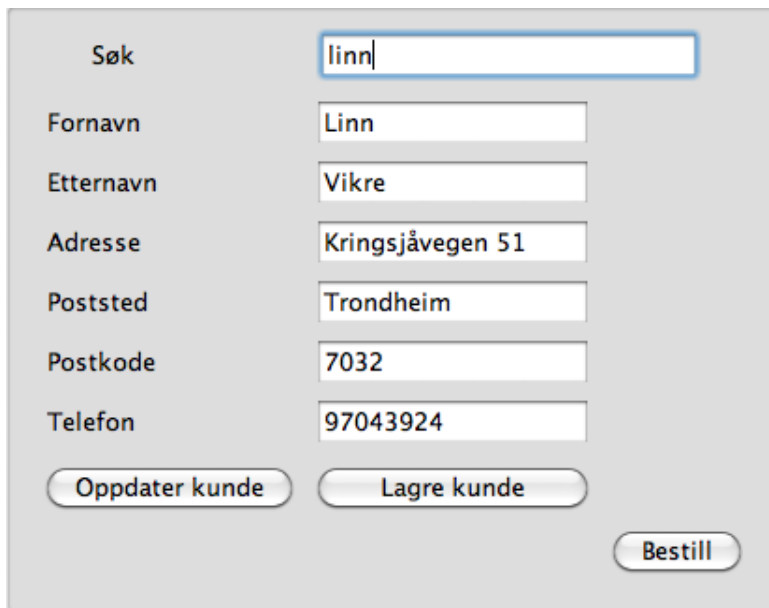
For å søke opp en eksisterende kunde, brukes søkefeltet øverst. Skriv inn det du ønsker å søke på i dette feltet, det kan enten være fornavn, etternavn eller telefonnummer.



A screenshot of the same web form, but with the search input field containing the text 'vegard'. A dropdown menu is open below the search field, displaying four suggestions: 'Vegard Harper 93450490', 'vegard harper 76543219', 'vegard harper 76549877', and 'vegard harper 9345049'. The other input fields and buttons remain the same as in the previous screenshot.

Hvis det finnes flere brukere i databasen med samme navn, vil du få opp en liste med forslag. Dette krever at man skriver inn fornavn. Når du har valgt den kunden som er riktig, vil all informasjonen vises i feltene under.

Hvis det finnes en unik bruker i databasen, vil informasjonen fylles ut i feltene under automatisk.

A web form for customer registration. It has a search bar at the top with the text 'linn'. Below it are input fields for 'Fornavn' (Linn), 'Etternavn' (Vikre), 'Adresse' (Kringsjåvegen 51), 'Poststed' (Trondheim), 'Postkode' (7032), and 'Telefon' (97043924). At the bottom are three buttons: 'Oppdater kunde', 'Lagre kunde', and 'Bestill'.

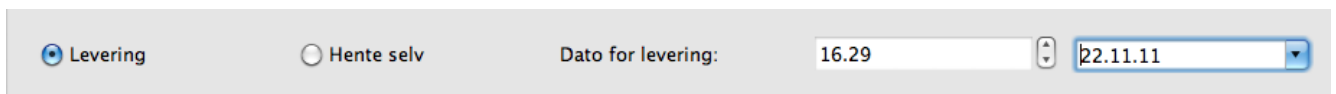
Hvis det er en ny kunde, fyller du informasjonen om kunden inn i feltene, og trykker på “Lagre kunde”. Da legges kunden legges inn i databasen.

Viktige ting angående input i feltene:

- Postkode må være siffer.
- Telefonnummer må være siffer.
- Resten kan være tekst.

Om kunden har byttet adresse eller telefonnummer, kan dette enkelt endres på enten før man bestiller eller senere. Da endrer du bare de aktuelle feltene som skal endres og trykker “Oppdater kunde”. Endringene vil da bli lagret.

Bestilling

A form for selecting order status and delivery date. It has two radio buttons: 'Levering' (selected) and 'Hente selv'. To the right is a label 'Dato for levering:' followed by a text input field containing '16.29' and a date dropdown menu showing '22.11.11'.

Når du kommer inn på bestillingssiden så har du muligheten til å endre på tidspunkt, dato og status om kunden skal hente selv eller få bestillingen levert hjem til seg, øverst i skjermbildet. Tidspunktet for levering er satt en time frem, fordi man antar det tar tid før en ordre blir produsert og blir kjørt ut. Her har du altså muligheten til å legge inn en bestilling i god tid før den skal leveres.

PizzaPlace

Kunde **Bestilling** Kjøkken Levering Settings

☒ Levering ☐ Hente selv

Dato for levering: 16.29 22.11.11

Fornavn: Linn
Etternavn: Vikre
Adresse: Kringsjåvegen 51
Poststed: Trondheim
Postkode: 7032
Telefon: 97043924

5 x Stor Margarita 675.0
1 x Liten Kebabpizza 135.0

Margarita
Tomat, sjampinjong, skinke
Antall : 1
Størrelse : Liten
Pris: 135.0
Kommentar:
Legg til

Kebabpizza
kebabkjøtt, hvitløkssaus, jalapeño, paprika, mais
Antall : 1
Størrelse : Liten
Pris: 135.0
Kommentar:
Legg til

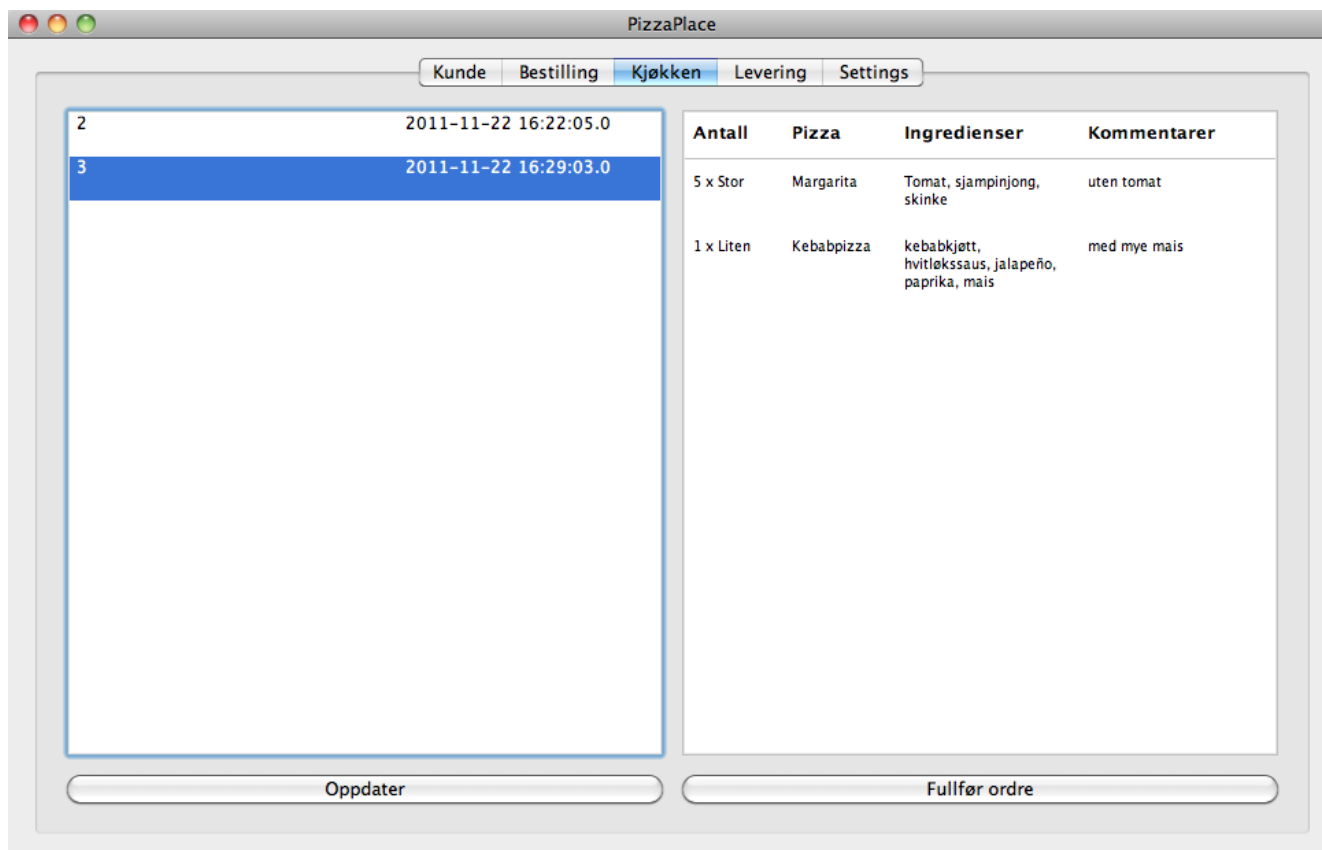
Hawaii

Avbryt Bekreft

For å legge til produkter blar du bare opp og ned i vinduet til høyre etter produktet kunden ønsker, velger antall, størrelse og eventuelt en kommentar og trykker legg til. Produktene som legges til blir lagt til i lista nede til venstre.

Vil du fjerne et produkt fra lista, gjør man dette enkelt ved å dobbeltklikke på elementet i lista. Når bestillingen er klar trykker du bare på “Bekreft”, om kunden ombestemmer seg trykker du på “Avbryt”.

Kjøkken



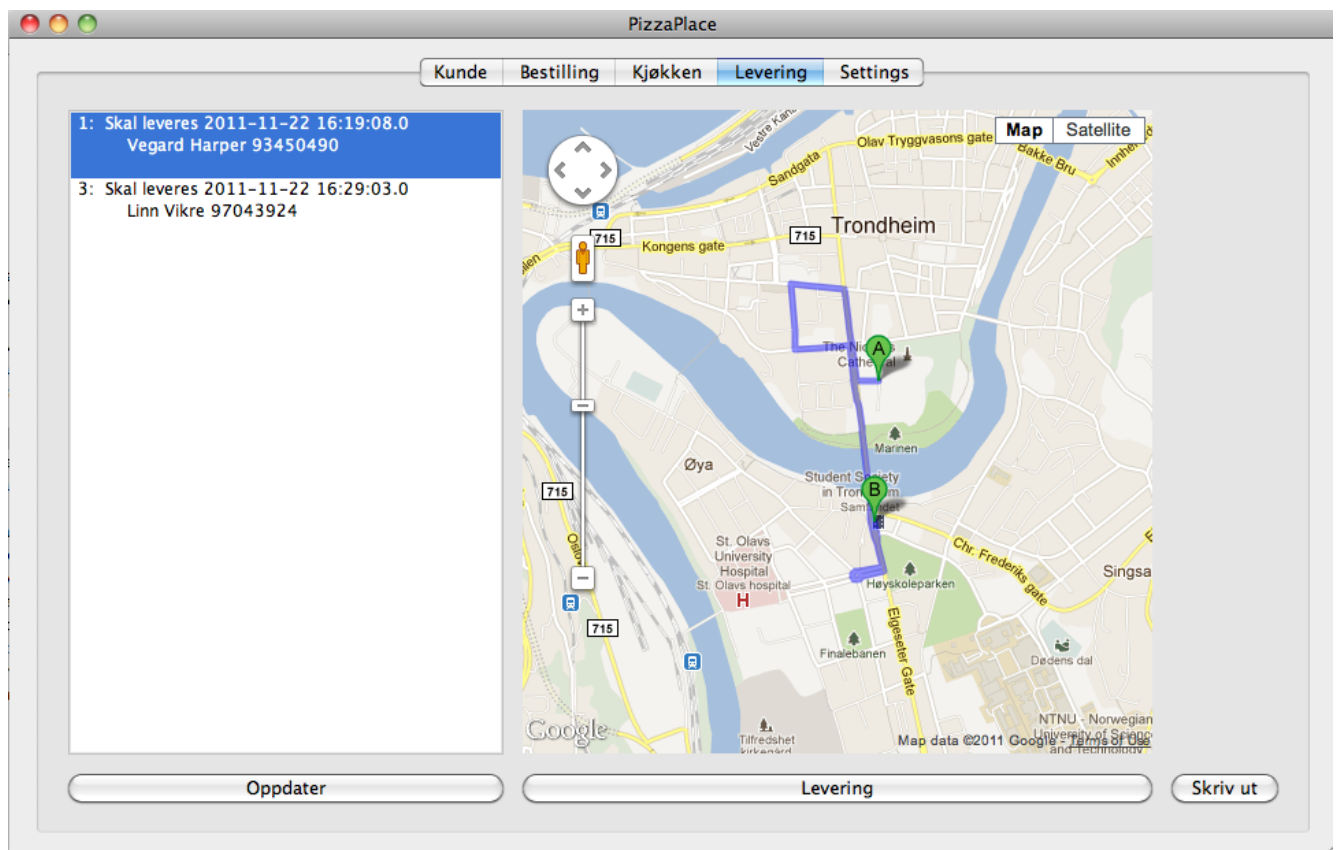
Kjøkkenvinduet består vinduet av to kolonner. Til venstre finner kokken ordrene listet opp med ordrenummer, dato og tid, listen er sortert etter hvilken ordre som skal være ferdigstilt først.

For å få opp informasjon om en ordre, velger kokken enkelt den ordren fra listen som han ønsker å få mer informasjon om. Da vil det i høyre kolonne dukke opp en oversikt over hvilke retter ordren inneholder, samt antall og eventuelle kommentarer.

Når en ordre er ferdig, kan kokken trykke “Fullfør ordre” og ordren vil da forsvinne fra listen.

For å oppdatere listen med ordrene, kan kokken trykke på “Oppdater”, og eventuelle nye ordre vil da bli sortert inn i listen over ordrene.

Levering



Leveringsvinduet er bygd opp på samme måte som vinduet i kjøkken. Listen til venstre viser ordrene, og det er presisert om de skal leveres eller ikke. Disse ordrene er sortert på tid, etter når de skal hentes/ leveres, hvor den som har kortest tid til levering/henting ligger øverst i lista.

For å prosessere en ordre velger du den ønskede ordenen fra lista til venstre. Da vil du få opp et kart i vinduet til høyre, som viser hvor ordren skal leveres. Hvis den skal leveres til en kunde, vil kartet vise ruten mellom bedriftens adresse og kundens adresse. Hvis kunden ønsker å hente ordren selv, vil det vises et kart med bedriftens adresse som midtpunkt.

Kvittering

Ordre nr : 73
Navn : Vegard Harper
Adresse : Elgestergate 1
Telefonnummer : 93450490
Skal være ferdig til : 2011-11-16 16:19
Skal leveres : Ja

Pizza	Pris
1 x Liten Margarita	135.00

Bestillingsinformasjon

Leveringspris:	50.00
Totalpris med mva:	135.00
Totalpris uten mva:	108.00
Merverdiavgift:	27.00

Total pris: 185.00

Skriv ut **Avbryt**

Når du har valgt en ordre kan du trykke på knappen “Skriv ut”. Da kommer det opp et vindu med all informasjon om hva bestillingen inneholder og totalpris. Hvis man ønsker å skrive ut kvitteringen trykker man bare på “Skriv ut”.

Når ordren er levert trykker man på “Levering”, ordren vil da forsvinne fra skjermen.

Hvis den som leverer ønsker å se om det har kommet flere ordre som skal leveres eller hentes, kan han trykke på “Oppdater”. Da vil ordre som kokken har ferdigstilt bli sortert inn i listen over ordre som skal leveres/hentes.

Settings

PizzaPlace

Kunde Bestilling Kjøkken Levering **Settings**

Legg til nye pizzaer

Navn på produkt:

Ingredienser

Pris

Legg til produkt i DB

Endre prisgrense for gratis levering og leveringsgebyr

Nåværende grense er 500.0 og nåværende pris er 50.0

Ny grense Endre grensen

Ny Leveringspris Endre leveringspris

Settingsvinduet består av to deler, øverst kan man legge inn nye produkter. Da skriver du inn navn, ingredienser og pris på det nye produktet. Når dette er gjort trykker du på "Legg til produkt i DB".

Nederst kan du endre grensen for hvor mye man må handle for før man får gratis levering. Og du kan også endre prisen på leveringen. Du skriver bare inn ønsket verdi og bekrefter med knappen til høyre for feltet. Skal du legge inn en verdi med desimaler må du bruke punktum. Eksempel: 50.00 er riktig og 50,00 er **feil**.

7.2 Teknisk brukermanual

Programmet vårt benytter seg som nevnt av QT Jambi rammeverket, som en direkte konsekvens er ikke dette direkte multiplattform. Det er imidlertid mulig å lage en .jar fil som tar høyde for dette, ved å pakke inn alle avhengigheter inn i en fil. Vi fant dessverre ikke tid til dette, da dette var en del over vårt nivå. Som en annen løsning har vi valgt å kompilere applikasjonen vår for hver plattform vi ønsker å støtte. Disse er som følger, Windows 32/64, Linux 32/64 og Mac OS X 32/64.

For hver plattform distribuerer vi en ferdig kompilert .jar fil som enkelt kan kjøres ved kommandoen `java -jar file_name.jar`. Det eneste som må merkes seg er at for å kjøre filen på Mac OS X må vi endre dette til `java -jar file_name.jar -d32 -XstartOnFirstThread`.

For å kjøre programmet krever det også at man er på NTNU sitt nettverk, enten via trådløst, kabel eller over VPN. Dette fordi vi bruker NTNU sin MySQL-database for lagring av informasjon, som igjen krever at man er på NTNU sitt nett.