BGPP Integreret Projekt

Udleveret torsdag 20. november 2014

Formål med projektopgaven

I projektet skal I analysere et givet problem og programmere et softwaresystem til at løse det. I skal præsentere *systemets formål, opbygning og virkemåde*, og I skal afprøve systemet, redegøre for om det virker som ønsket, og vurdere i hvilken grad jeres afprøvning understøtter en sådan konklusion.

Projektet involverer udvikling, test og dokumentation af et Java-program med grafisk brugergrænseflade og brug af en relationsdatabase.

Emne for projektopgaven

I projektet skal der udvikles et softwaresystem til brug for den *ekspedient* der betjener billetlugen og telefonen i en mindre biograf. En *kunde* er en person der står foran billetlugen eller som ringer ind på telefon. Løsningen skal understøtte ekspedientens arbejdsopgaver (tasks) i forbindelse med kundehenvendelser, for eksempel:

- En kunde vil gerne have to pladser ved siden af hinanden til en bestemt forestilling (film og tidspunkt) i dag, helst bagest i salen.
- En kunde vil gerne have 17 billetter til filmen *Interstellar* kl 19:00 en hvilken som helst dag i den kommende uge.
- En kunde ringer og vil gerne reservere 4 billetter til en hvilken som helst film kl 19:00 i morgen, men helst ved siden af hinanden og så langt fremme i salen som muligt.
- En kunde vil gerne have yderligere 2 billetter føjet til den reservation på 4 som han har i morgen kl 19:00.
- En kunde ringer ind og vil gerne have flyttet sin bestilling af 6 billetter fra kl 19:00 forestillingen i morgen til kl 21:30 forestillingen med samme film.
- En kunde ringer ind og afbestiller de 6 billetter han bestilte i går.
- En kunde vil både købe 1 billet til *Nightcrawler* ved forestillingen om 25 minutter, og reservere 5 billetter lørdag eftermiddag til *Stjerner uden Hjerner*.
- En lang række varianter kan tænkes: Der er ikke nok ledige pladser ved siden af hinanden; der er slet ikke nok ledige pladser; filmen går ikke på det ønskede tidspunkt; filmen går slet ikke i denne biograf; osv.
- Lav selv en liste af sandsynlige arbejdsopgaver som ekspedienten skal kunne løse ved hjælp af systemet.

Løsningen skal kunne håndtere reservation af pladser, men ikke salg; dvs. billetpriser og betaling skal ikke implementeres. Ved reservation skal der opgives identitet, fx telefonnummer eller navn.

Biografen har kun én billetluge og tillader ikke reservation over nettet, så løsningen behøver ikke håndtere samtidige opdateringer til databasen.

Biografen har flere sale og viser forskellige film på forskellige tidspunkter af dagen i de forskellige sale. En kombination af film, sal, tidspunkt og dag kaldes en *forestilling*.

De forskellige sale kan have forskelligt antal stolerækker og antal stole per række. Antallet af sale og salenes antal stolerækker mv skal fremgå af databasen og må ikke være indkodet i selve billetlugeprogrammet. For enkelheds skyld kan det antages at alle rækker i en given sal har lige mange stole, og at der ikke er nogen midtergange eller krumme stolerækker.

For at biografen kan håndtere skiftende forestillingsprogrammer skal de aktuelle forestillinger fremgå af databasen; de må altså ikke være indkodet i selve billetlugeprogrammet. Det er ikke en del af opgaven at lave en administratorbrugergrænseflade til at redigere disse forestillingsoplysninger. I kan altså bare opdigte nogle sale og forestillinger og lægge dem ind i jeres database - for eksempel ved at benytte MySQL WorkBench.

Til inspiration findes en skitse af en mulig brugergrænseflade i figur 1. I må meget gerne lave jeres brugergrænseflade anderledes, lav den ikke mere indviklet, hellere simplere.

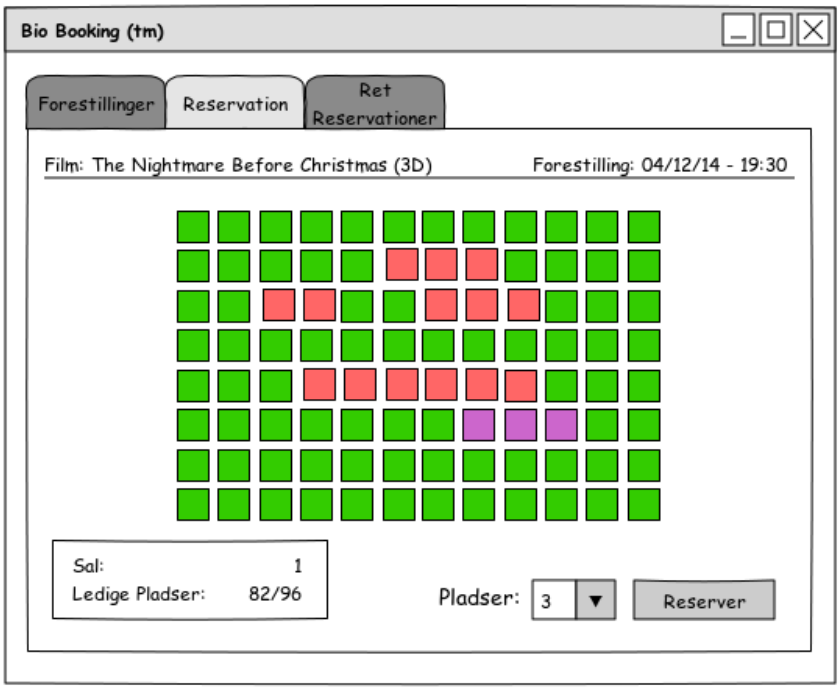

Figure 1: Skitse af mulig brugergrænseflade. Ledige sæder er grønne, røde sæder er reserverede og de lilla sæder er valgt som en del af en ny reservation.

Regler for projektarbejdet

- Opgaven udleveres ved forelæsningen torsdag 20. november og projektrapporten skal afleveres senest **onsdag 17. december kl 14:00 gennem LearnIt**. Det er jeres ansvar at aflevere til tiden. Hvis I afleverer for sent, kan I ikke gå til eksamen. I bør afsætte rigeligt tid til at gennemgå og uploade rapport samt kodefiler mv, for at sikre korrekt og rettidig aflevering.
- Projektet udføres i grupper à 3 personer.
- I skal selv danne grupper og meddele Jesper Madsen (jmad@itu.dk) navn og email på gruppens 3 medlemmer senest torsdag 27. november kl 17:00.
- Systemet skal programmeres i Java. Man kan frit vælge databaseserver, men MySQL er en oplagt mulighed. Man kan ligeså frit vælge mellem Java Swing og JavaFX til brugerfladen, husk dog at separere brugerflade, programlogik og data således programmet kan testes. Design af brugerflader i JavaFX er gennemgået i en kursusvideo på LearnIT.
- Om rapportens form og indhold, se vejledningen *Udformning af rapporter* der findes på kursets LearnIT side.
- Projektrapporten (eksklusive bilag) skal have et omfang på ***maksimalt*** 15-18 normalsider (dvs. sider uden figurer).
- Rapportens forside skal oplyse: `kursus`, `projekttitel`, `forfatternes navn` og `ITU-email`, `dato`. ***Alle afleveringer på ITU skal benytte ITU's standard-forside, som kan findes på ITU's intranet.***
- Der er instruktorhjælp til projektet tirsdag 13:00-16:00 og torsdag 13:30-15:30 fra og med tirsdag 25/11 til og med tirsdag 16/12.
- Hver projektgruppe har mulighed for korte vejledermøder med Jesper Madsen. Send en mail med et agenda[1] for mødet samt foreslåede tidspunkter. Spørgsmål af teknisk karakter henvises til TA's, vejledermøder er udelukkende ment til generel vejledning omkring strukturering af projektarbejde.

[1] Kort opsummering af hvad vi skal diskutere.

- Brug meget gerne kursets LearnIt side til at stille spørgsmål af almen interesse, fx om fortolkning af denne opgavetekst.
- *Plagiat er forbudt*. I må ikke bruge andres tekst, illustrationer eller programkode uden udtrykkelig kildeangivelse. Plagiat kan resultere i udelukkelse fra eksamen, enddog bortvisning.
- Studerende der ønsker at lave projektet alene, jvfr. Eksamensbekendtgørelsen BEK nr. 231 af 22/03/2006, §4 stk 3, skal bekendtgøre deres ønske om dette ved at sende mail til Jesper Madsen (samme tidsfrist). I er dog ***stærkt*** anbefalede, grundet projektets omfang, ikke at lave projektet alene.

Vurdering af projektarbejdet

Hovedvægten i vurderingen lægges på projektrapporten. Rapportens formål er at dokumentere at de enkelte delproblemer er løst *og at programmets funktionalitet er testet i form af unit tests*. Rapporten bør indeholde:

- En præsentation, præcisering og afgrænsning af krav til systemet, fx i form en liste af de arbejdsopgaver (tasks) og de data som systemet skal håndtere.
- Begrundede designbeslutninger vedrørende brugergrænseflade, databasens design, og programmets interne struktur (klasser og objektstrukturer - hvilket igen med fordel kan grupperes i moduler).
- Teknisk orienteret beskrivelse af databasedesign og programmets interne struktur.
- Begrundet afprøvning af det resulterende system, og vurdering af i hvilken grad systemet opfylder sit formål. Afprøvningen kan vedrøre brugergrænsefladens funktionalitet og unit test af klasserne.
- Kort brugervejledning til det resulterende system.
- Kort konklusion om i hvilken grad produktet opfylder de opstillede krav, herunder en mangelliste.
- Reflekterende beskrivelse af arbejdsprocessen, fx om I har nået projektets læringsmål.
- Bilag:
  - kondenseret projektdagbog, og endelige versioner af arbejdsblade[2]

Derudover forventes en ZIP pakke med Java-kildeteksten til det udarbejdede softwaresystem. Husk at fjerne irrelevante filer (IDE projektfiler, kompilerede Java (.class) filer med videre).

Der lægges vægt på at brugergrænsefladen understøtter hyppigt forekommende arbejdsopgaver godt, og at den programmeringsmæssige løsning er velstruktureret, veldokumenteret og vedligeholdsvenlig og ikke indeholder overflødige dele.

Læg mærke til at projektrapporten både omhandler *produktet*, altså softwaresystemet og dets opbygning, og *processen*, altså hvordan I har arbejdet, dog med mest vægt på omtalen af produktet.

Gode råd

Løs hellere nogle delproblemer godt end alle overfladisk

Det er *meget* bedre udtrykkeligt at beslutte at et specielt delproblem ("samme film vises i to forskellige sale samtidig") slet ikke håndteres i systemet, og eventuelt skrive det på en mangelliste, end at programmere en uigennemtænkt, uafprøvet og udokumenteret løsning til delproblemet. Sådan en delløsning kan komplicere den samlede softwareløsning, reducere dens brugbarhed og vanskeliggøre videreudvikling.

Lad være med at opfinde komplicerede softwareløsninger på opgaver som ekspedienten meget bedre kan løse visuelt, fx "finde fire ledige sæder ved siden af hinanden helst foran i salen men ikke ude til siden". Brug hellere opfindsomheden på at lave en god visuel understøttelse til ekspedienten, eller på at give mulighed for at bruge tastaturet i stedet for musen, der kan være meget belastende ergonomisk.

[2]Med arbejdsblade menes der hvad end måtte findes af noter med overvejelser, skitser eller lignende, som er udarbejdet før løsningen implementeres.

Divide and Conquer

Lad være med at lave alt på en gang. Forsøg at se på hvert delproblem for sig, og lav arbejdsblade til hvert delproblem, så I kan huske hvad I har overvejet og besluttet og hvorfor:

- Eksempel: Lav listen af ekspedientens arbejdsopgaver færdig.
- Eksempel: Undersøg om jeres brugergrænsefladedesign kan understøtte alle arbejdsopgaverne.
- Eksempel: Gennemtænk databasedesignet, opdigt nogle eksempeldata, og se om alle nødvendige data er til stede for at den tænkte brugergrænseflade ville kunne fungere.
- Eksempel: Find ud af hvordan en Java-komponent kan tegne en sal med givne dimensioner (antal stolerækker og antal stole per række), og vise de ledige, reserverede og solgte sæder med forskellige farver.
- Osv.

Start simpelt

Hvis noget virker meget uoverskueligt, så *Solve a Simpler Problem First*. Lav en løsning der er åbenlyst utilstrækkelig, men dog et skridt i den rigtige retning. Derefter er I meget klogere, og kan tage et nyt skridt. Fx kan man starte med kun én sal i biografen, eller kun én stolerække, eller med at man kun kan bestille én billet ad gangen.

Behandl ikke rapporten som en eftertanke

Husk at arbejde på rapporten undervejs i projektet, og at indberegne rapportskrivning i jeres tidsestimater. Arbejdsbladene udfærdiget undervejds i projektet kan med fordel benyttes i rapportskrivningsprocessen, således alle begrundelser for designbeslutninger og tests bliver reflekteret i den endelige rapport.

Husk at bruge tegninger og tabeller, ikke kun tekst, når I skriver rapport - rapporten skal kunne give et fyldestgørende overblik af jeres arbejde og tanker herom.

Behandl ikke tests som en eftertanke

Husk at designe jeres program således I kan teste dets funktion. Programmet kan nemmere testes såfremt I deler det op i logiske, velafgrænsede enheder - for eksempel ved at separere brugerfladen (view) fra programlogikken (controller) og den underliggende data (model).

Test inkrementelt

Det er en fordel at teste koden, efterhånden som I skriver den - på den måde er I bedst stillede til at identificere de interessante corner-cases som skal testes og I får samtidig sikret jer, at færre fejl kommer til at påvirke andre moduler som kodes sidenhen.

Lad arbejdsopgaverne diktere jeres GUI

En god GUI er optimeret til at de hyppigste arbejdsopgaver er hurtige og lette at klare. Desuden er den konsistent, dvs. prøv altid at vise den samme data, fx, kundelisten i det samme vindue, for ikke at forvirre brugere. Overvej hvorledes I kan lave så få vinduer som muligt. Tænk længe over, hvordan disse bedst kan vise den gemte data og hjælpe med at visualisere arbejdsopgaverne; f.eks. pladsreservation og oversigt over film med ledige billetter den næste uge.

Lad opdigtet eksempeldata drive GUI- og databasedesign

Det er en fordel at opdigte eksempeldata ud fra de forskellige arbejdsopgaver, og benytte dette til inspiration til hvorledes databasen skal se ud og hvad den skal gemme, samt hvordan GUI'en skal vise den gemte data.

Inden aflevering

Inden aflevering bør I sikre at alle krav, som er beskrevet i dette dokument, er opfyldt. Gennemgå gerne dokumentet flere gange.

I kan bruge følgende *ukomplette* checkliste til at sikre at jeres aflevering har:

- ☐ En velskrevet rapport som:
  - ☐ Indeholder en generel introduktion til programmet i form af en brugermanual
  - ☐ Er logisk sammenhængende, f.eks ved ikke at blande mange emner i samme afsnit og ikke beskrive ting der ikke introduceres før senere i rapporten.
  - ☐ Indeholder en *reflekterende* beskrivelse af arbejdsprocessen
  - ☐ Indeholder en sektion, der beskriver den overordnede test-strategi for hvert komponent i systemet.
  - ☐ Indeholder samtlige test cases i appendix
  - ☐ *I øvrigt imødekommer vurderingspunkterne beskrevet i "Vurdering af projektarbejdet"*
- ☐ En ZIP pakke med alle Java kodefiler samt andre filer/komponenter der indgår i jeres program. Husk at rydde op i strukturen, fjern kompilerede Java (.class) filer, IDE-relaterede filer med videre.
- ☐ Beskrivelse af hvordan programmet (JAR fil) skal køres af andre. Det er jeres opgave at sikre at programmet kan køres af lærere og censor. Det skal være muligt for eksaminator og censor at køre projektet som en del af bedømmelsen af projektet. Hvis det er nødvendigt at have login og password til databasen for at køre programmet skal disse fremgå tydeligt af rapportens forord.
- ☐ En (eller flere) ZIP filer indeholdende:
  - ☐ Rapport i pdf format.
  - ☐ Kode filer.
  - ☐ JAR fil til jeres program.