Restaurangbokning

Laboration 2, Grafiska användargränssnitt

Simon Åkerblom

12 May 2022

Introduktion

Ett platsbokningssystem för en restaurang. Programmet består av en förenklad version av restaurangens planlösningen och en kölista. Borden är dynamiska och ändrar färg när man klickar på dem för att markera om de är lediga/upptagna. Programmet ska användas vid entrén och har ett grafiskt gränssnitt anpassat efter stressig miljö och hög personalrotation. Kösystemet är en hjälp så att personal enkelt kan hålla koll på hur många som väntar på bord och hur många de är i respektive sällskap.

Klicka på ett bord för att markera det som ledigt/upptaget. Lägg till ett sällskap i kölisten genom att klicka på "Add"-knappen, ange namn och antal personer i sällskapet, sen "Ok". Klicka på en post i kölistan för att ta bort den, bekräfta genom att klicka på "Ok".

Grafisk Design

En av designprinciperna som har använts som riktlinje under designprocessen är tagen från kurslitteraturen och har samma namn som designprincipen: "Don't make me think", av Krug (2014). Nedan beskrivs varför designprincipen är användbar i det här projektet.

"Don't make me think"

Principen innefattar ett brett spectrum av frågor man bör ställa sig vid val av design och täcker in mycket som är önskvärt ur ett användbarhetsperspektiv. Eftersom personalrotationen är hög och arbetsmiljön kan vara stressig är det extra viktigt att personalen som ska använda sig av systemet inte behöver lägger extra energi på att försöka lista ut hur man ska använda programmet och inte distraheras av onödig/överflödig information. Den mentala och

kognitiva belastningen ska vara minimal för att låta personalen ta hand om det som är viktigt: att serva gästerna.

Den andra designprincipen som varit en ledstjärna under designprocessen är "Match Between the System and the Real World", formulerad av Nielsen (1997). Kaley (2018) har vidare förklarat innebörden av principen i artikeln "Match Between the System and the Real World: The 2nd Usability Heuristic Explained" med tydliga exempel.

"Match Between the System and the Real World"

Det finns en viss överlappning mellan den här principen och "Don't make me think" eftersom det handlar om att få en snabb överblick av systemet och intuitivt förstå hur det fungerar; men här är fokus mer på det visuella och placering av grafik/komponenter. Eftersom restaurangen har en detaljerad planlösning finns det en möjlighet att skräddarsy det grafiska gränssnittet och bygga ett system som matchar, eller i alla fall efterliknar, verkligheten. Metaforer/symboler för ingångar/utgångar, kök, badrum osv, kan användas för att ge användaren en bra bild av bordens nummer och placering i relation till verkliga ting.

Med dessa två designprinciper i åtanke tog en enkel skiss form, se figur 1 nedan. Som sedan utvecklades till programmets bakomliggande gränssnitt som visas i figur 2. Programmet är uppdelat i två delar: Patio som är uteplatsen och Restaurant som är inomhus. Visuella metaforer finns med för att ge personalen en bra översikt.

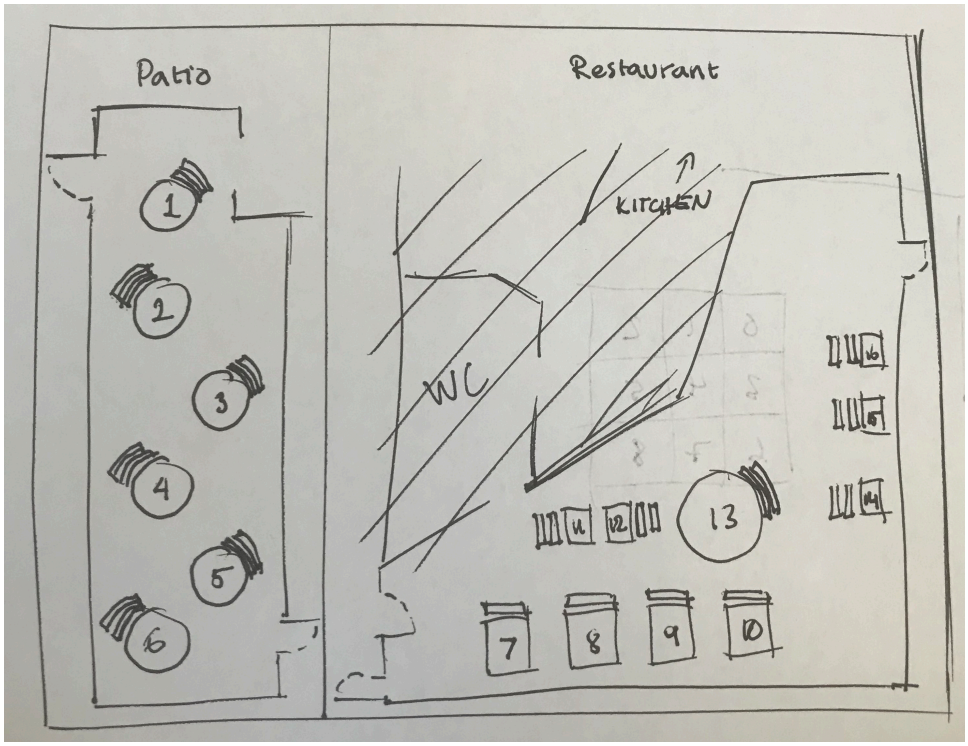

Fig 1. Skissen som användes som utgångspunkt för den grafiska designen.

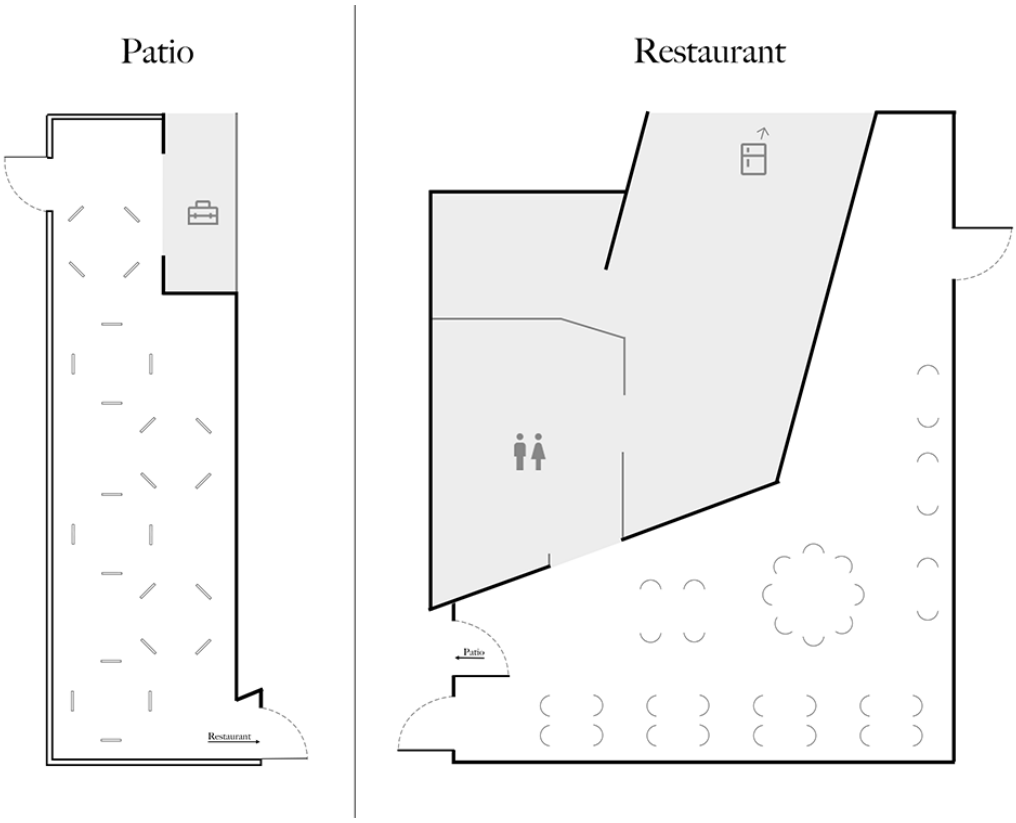

Fig 2. Förenklad planlösning som visar antal platser vid varje bord, ingångar/ utgångar, och metaforer.

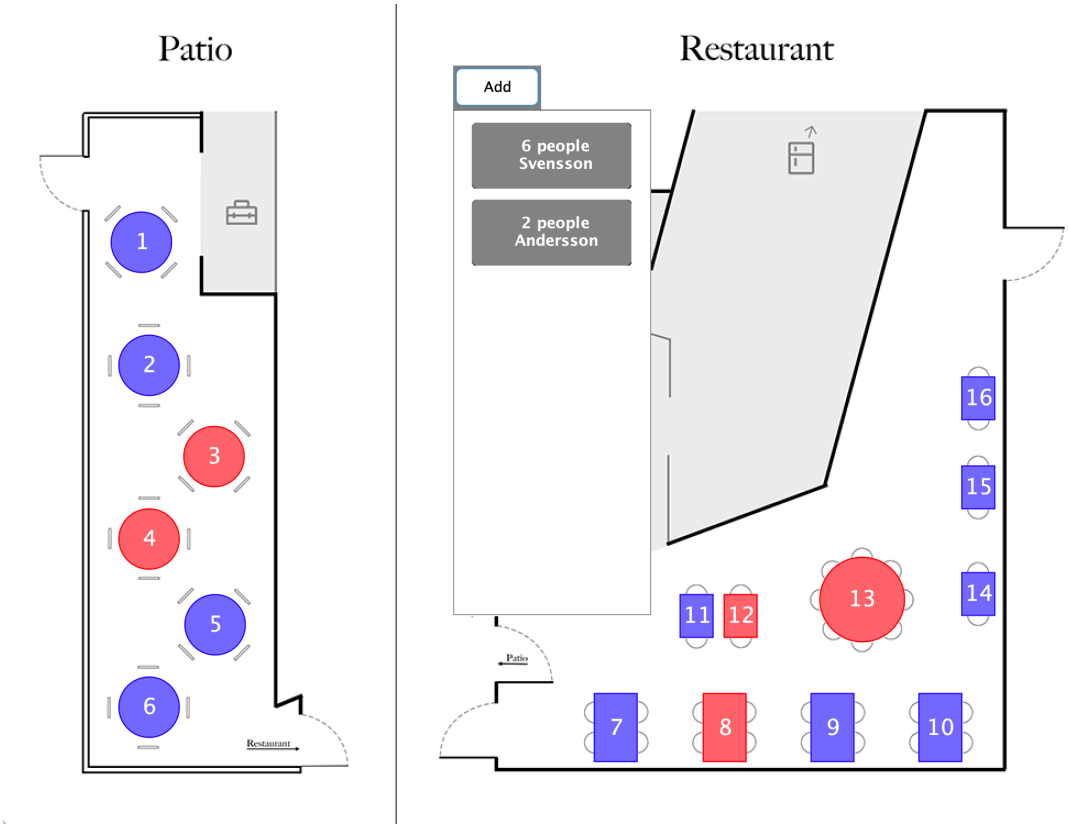

Fig 3. Programmets GUI efter implementation.

Slutresultatet är väl anpassat efter touchskärm (se figur 3). Alla klickbara komponenter: bord, “Add”-knappen, kö-poster, har stor träffyta och väl tilltagen marginal med syftet att minska den motoriska belastningen.

Programdesign

Programmet är uppdelat enligt designmönstret Model-View-Controller (MVC) och består således av en View-klass, en Controller-klass och en Model-klass. Model kommunicerar med klasserna Table och QueueEntry, View med Floorplan, QueueEntryPanel och QueueEntryTile (se figur 4).

Model har hand om bordens placering och storlek och lagrar dessa i en ArrayList, samma med QueueEntry.

Controllern förser View med objekten som behövs för att rita upp grafiken och ser till att den data som är sparad i Model synkroniseras och uppdateras mot gränssnittet som visas utåt mot användaren.

View ritar som sagt upp gränssnittet och visar dialogrutor vid behov. Några exempel på dialogrutor: inmatningsruta för att lägga till en post i kölisten, verifiering av borttagning av post, felaktig inmatning.

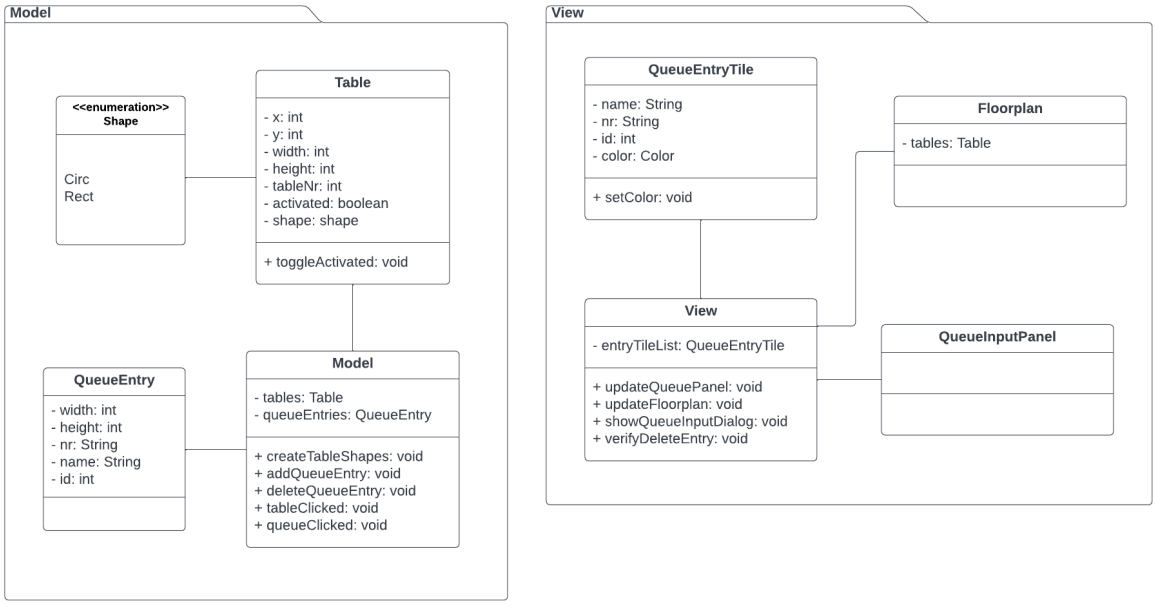

Fig 4. En abstraktion av systemet, uppritat enligt UML-notation.

Eftersom uppgiften lägger stor vikt vid gränssnittsdesignen, och inga större krav på funktionalitet, var det rimligt att börja implementeringsprocessen i vyn och inkludera logiken där först. Och när programmet väl var färdigbyggt, både gränssnitt och funktionalitet, kunde kodblock brytas ut och placeras i Controller- och Model-klasserna.

Implementation

Model

Attribut

tables: ArrayList<Table> - Här sparas alla Table-objekt.

queueEntries: LinkedList<QueueEntry> - Här sparas alla QueueEntry-Objekt.

Metoder

createTableShapes(int, int): void - När programmet startar tar denna metod emot bredd och höjd på gränssnittet och skapar alla Table-objekt, och sparar i tables.

addQueueEntry(String, String): void - Tar emot två strängar, en för namn och en för antal personer i sällskapet, och skapar en ny kö-post (QueueEntry) och lägger till i QueueEntries.

deleteQueueEntry(int): void - Tar emot id:et för den kö-post som klickats på och plockar bort den posten från queueEntries.

tableClicked(Table): void - Tar emot ett Table-objekt från Controller och anropar metoden toggleActivated() i det Table-objektet.

Table

Attribut

x: int - X-koordinaten som bestämmer var i gränssnittet bordet ska ritas upp.

y: int -Y-koordinaten som bestämmer var i gränssnittet bordet ska ritas upp.

width: int - Bordets bredd.

height: int - Bordets höjd.

tableNr: int - Bordets nummer.

activated: boolean - Bestämmer om bordet är upptaget eller inte. True om det är upptaget (activated), annars false.

shape: shape - Bestämmer om bordet ska ritas om som en oval eller rektangel.

Metoder

toggleActivated(): void - Ändrar värdet på activated-attributet. False om true och true om false.QueueEntry

Attribut

width: int - Kö-postens bredd.

height: int - Kö-postens höjd.

nr: String - Antalet personer i sällskapet.

name: String - Namn på personen som representerar sällskapet.

id: int - Ett unikt id för att identifiera vid borttagning.

View

Attribut

entryTileList: ArrayList<QueueEntryTile> - En lista med QueueEntryTile-objekt som fylls på varje gång kölistan ska ritas upp i gränssnittet.

Metoder

updateQueuePanel(LinkedList<QueueEntry>): void - Tar emot en lista med QueueEntry-Objekt, skapar nya grafiska QueueEntryTile-komponenter, och lägger till i kö-panelen.

updateFloorplan(): void - Riter om grafiken för Floorplan-objektet. Anropas efter att ett bord har klickats på och fått ett nytt värde på Activated-attributet.

showQueueInputDialog(): void - Visar en input-ruta där användaren behöver anger namn och antal person i sällskapet. Anropas efter att användaren klickat på “Add”-knappen.

verifyDeleteEntry(): void String - Visar en dialogruta där användaren måste bekräfta borttagning av en kö-post. Anropas efter ett användaren klickat på en kö-post i kölistan.

QueueEntryTile

Attribut

name: String - Name på personen som representerar ett sällskap som står i kö.

nr: String - Antal personer i sällskapet.

id: int - Id på komponenten. Används vid borttagning för att identifiera vilken kö-bricka som klickats.

color: Color - Bakgrundsfärgen på kö-brickan.

Metoder

setColor: void - Ändrar bakgrundsfärgen på kö-brickan. Anropas när användaren hovrar med musen eller håller fingret nedtryckt på ett bricka.

Slutsats

Designprocessen är snårig och tar tid. Det är viktigt att ha en klar bild av systemet innan implementering. Man behöver sätta sig in i hur slutanvändaren kommer använda programmet, och även arbetssituationen. Ett tilltalande gränssnitt som är användarvänligt, visuellt tilltalande, och roligt att använda, och som dessutom tillför ett verkligt värde är utmanande och kräver ett noggrant förarbete.

I en alternativ version hade man kunna använda sig av en GridBagLayout för att placera ut komponenterna, istället för LayeredPane, och på så vis göra programmet responsivt. Då hade man kunnat skapa Constraints-objekt och se till att grafiken förhåller sig till skärmytan. Bilden som fyller bakgrunden av programmet hade då kunnat ritats ut med paintComponent()-metoden. Fördelen är då att programmet hade fungerat på vilken enhet som helst, även på små skärmar som smartphones.

Det tog ungefär 8 timmar att ta fram den grafiska designen av programmet, och ca. 20 timmar att implementera det. Java kan vara klurigt och ibland kör fast på småsaker. Det är enklare och roligare att programmera grafiska användargränssnitt om man får direkt visuell feedback när man kodar, speciellt när man använder flera olika typer av layouter och konstruerar egna komponenter.

Referenser

Kaley, A. K. (2018, July 1). Match Between System and Real World: 2nd Usability Heuristic Explained. Nielsen Norman Group. https://www.nngroup.com/articles/match-system-real-world/

Nielsen, J. N. (1994, April 24). 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

Krug, S. (2014). Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability (3rd Edition) (3rd ed.). Pearson Education.