Grupp 1 - New Five, Markus Grahn, Oliver Andersson, Christian Lind, Victor Cousin, Moa Berglund

Slutrapport

RAD

23/10 2020

1. Introduktion

På arbetsplatser är schemaläggning en central arbetsuppgift för att verksamheten ska fungera. Detta är ofta en mycket tidskrävande uppgift, samtidigt som att den kan vara repetitiv. Därav är detta projektets syfte att skapa en applikation som schemalägger personal utifrån vad verksamheten kräver, och därav minskar det administrativa arbetet på ett företag. Användaren fyller själv i företagets uppgifter såsom anställda, avdelningar etc. Applikationen sätter sedan schemat utefter certifikat för olika arbetspass som användaren bestämt och tillgänglighet hos de anställda. Programmet är tillgängligt för olika branscher då innehavaren själv kan bestämma vilka krav som verksamheten har. Den förväntade användaren arbetar med schemaläggningen, och har ansvar för schemat. Applikationen begränsar sig till enbart denna användare, och erbjuder ingen funktionalitet eller gränssnitt för personalen som blir schemalagd.

1.1 Ordlista

Certifikat - arbetsrelaterad licens som en anställd kan bli tilldelad

2. Krav

2.1 User stories

ID är utsatta efter prioritet (01= högst). ID markerat med lila är implementerade, grönt är inte implementerade
"Som x, vill jag kunna y, för att z."

“ID 01: Planering as programstruktur
Som programmerare vill jag tänka igenom strukturen av projektet innan jag börjar koda för att undvika onödiga komplikationer längre in i utvecklingen.”
Acceptans:

- *Det finns genomtänkt domänmodell*
- *Det finns genomtänkt designmodell*
- *Det finns UMLdiagram för klass/paket/modul-nivå*

“ID:02 Anställa/Avskeda personal
Som användare
vill jag kunna lägga till och ta bort personal
för att hålla programmet uppdaterat om de som anställda just nu.”
Acceptans:

funktionell:

- *Det finns en sida där det går att lägga till personal via personuppgifter (textfält) i formulär*
  namn, tel, mail, personnr, typ av anställning, certifikat
- *Personalen som läggs till syns i någon form av lista.*
- *Det går att ta bort personal från listan*

"ID:03 Hantera anställdas certifikat
Som användare
vill jag kunna lägga till och ta bort certifikat hos de anställda
för att hålla programmet uppdaterat om kunskapsbanken just nu."
Acceptans:
funktionell:
- *Det går att tilldela certifikat till anställd*
- *Det går att skapa nya certifikat*
- *Det går att se vilka certifikat en anställd har*
- *Det går att se vilka anställda som har ett visst certifikat*
- *Det går att ta bort certifikat från anställd genom att trycka på någon knapp någonstans? I listan?*
- *Det går att ta bort certifikat från "möjliga certifikat"*

"ID:04 Kolla bemanning på avdelningar
Som användare
vill jag kunna veta att arbetsdagen bemannad utan att jag manuellt behöver göra det,
för att frigöra tid till annat för mig."
Acceptans:
funktionell:
- *Programmet genererar visuellt schema*
- *Schemat visar hur bemanningen är en viss tid*
- *användaren kan lägga in vilka pass en arbetsdag kräver för att vara bemannad korrekt*

ickefunktionell:
- *Användaren behöver inte lägga tid på att skapa ett schema annat än när programmet efterfrågar*

"ID:05 Krav på avdelningar
Som användare
vill jag kunna se till att alla avdelningar är bemannade med rätt certifikat
för att verksamheten ska fungera på bästa sätt."
Acceptans:
funktionell:
- *Certifikat för berört arbetspass bestäms av ägaren*
- *Endast behörig personal med rätt certifikat får arbeta på ett pass som kräver vissa certifikat*
- *För ett arbetspass går det att lägga till/ta bort certifikat när det kommit nya krav*

"ID: 06 Schemahantering
Som programmerare vill vi kunna hantera schemat på så sätt att vi kan hämta dagar och tider för att lägga in pass.

Funktionell:

- Hämta tider
- Lägga in pass

“ID:07 GUI ver 1
Som programmerare vill jag skapa ett körbart “GUI-skal” med javaFX för att ha någon form av gränssnitt att utgå och utveckla från”.
Acceptans:

Funktionell:

- *Det finns något körbart av GUI*

“ID:08 Notis vid inkomplett schema
Som användare
vill jag bli underrättad ifall det skulle bli brist på personal någon dag
för att manuellt fylla luckan.”
Acceptans:

Funktionell:

- *“Notis” visas vid brist av personal någon dag*
- *Berörd dag visar en varningstriangel i schemat*

“ID:09 Välja hur ofta anställda jobbar
Som anställd
vill jag inte kunna ha olika pass tätare än x timmar
för att ha en hälsosam livsstil.”
Acceptans:

funktionell:

- *Ägaren av programmet kan bestämma x antal timmar som de anställda garanteras vara lediga innan nästa arbetspass*
- *Programmet gör anställda vars lediga tid är mindre än x “otillgängliga”.*

“ID:10 Garantera alla rast
Som användare
vill jag veta att synkronisering av raster fungerar på så vis att bemanning fortfarande är intakt för att ha ett bra arbetsflöde.”
Acceptans:

funktionell:

- Användaren kan fylla i hur mycket rast som garanteras efter y timmar
- Användaren kan fylla i hur många som inte får ha rast samtidigt per avdelning
- Schemat visar vilka raster man har

“ID: 11 Ledighet
Som schemaläggare vill jag inte att någon ska kunna schemaläggas som har fått godkänd ledighet."
Acceptans:

funktionell:

- Det går att göra en person "upptagen" vid bestämd tid/datum så att det ej går att schemalägga vid ledighet

“ID:12 Hantera arbetspass
Som användare
vill jag kunna lägga till och ta bort arbetspass utefter behov
för att det kan hända oväntade företeelser som gör att vi behöver mer/mindre bemanning.”
Acceptans:

Funktionell:

- *Det går att ta bort arbetspass*
- *Användaren kan skapa nya arbetspass (valfri tid och avdelning)*
- *Det går att ha olika scheman olika veckodagar och tider*
- *Ändra bemanning på redan upplagda pass*

“ID:13 Inga felaktiga pass
Som användare
vill jag kunna byta personal på arbetspass i programmet
för att de anställdas scheman ska stämma även när förändringar görs.”
Acceptans:

funktionell:

- *Användare kan byta pass mellan personal*
- *Programmet låter inte samma person göra två arbetspass samtidigt*
- *Programmet låter inte obehörig personal arbeta på certifikat-krävda pass*
- *Programmet låter inte byta pass med någon som är ifylld “inte tillgänglig” den dagen*
- *Det går inte att byta ett pass till ett annat som ligger mindre än x timmar mellan den berördas redan tilldelade pass*

“ID:15 Visuell separation mellan avdelningar
Som användare
vill jag kunna urskilja i schemat vilka som arbetar på vilken avdelning
för att enkelt kunna uppsöka berörd personal under arbetstid.”
Acceptans:

Funktionell:

- *Schemat är visuellt synligt och särskiljer avdelningarna*
- *Avdelningarna särskiljs mha olika färger*

“ID:16 Kommentera koden
Som deltagare i ett gemensamt programmeringsprojekt vill jag att koden ska kommenteras väl för att enkelt kunna förstå vad de övriga har kodat.”
Acceptans:

Icke-funktionell:

- *Koden täcks övergripande av javadoc*

“ID:17 Exceptions
Som programmerare vill jag att de ska finnas tydliga felmeddelanden när något inte går som det ska för att förstå vad som är fel.”
Acceptans:

Funktionell:

- *Exception finns för när någon metod returnerar null*

“ID:18 Junit
Som programmerare vill jag kunna testa stor del av programkoden för att undvika buggar.”
Acceptans:

Icke-funktionell:

- *Junit är implementerad*

“ID:19 Separerade testklasser
Som programmerare vill jag ha en test-klass för varje klass i modellen för att få en bättre överblick av testerna.”
Acceptans:

Icke-funktionell:

- *Testmetoder är placerade i testklasser för respektive klass som testas*

“ID:20 Maven
Som programmerare vill jag implementera maven i projektet för att möjliggöra build automation.”
Acceptans:

Icke-funktionell:

- *Maven är implementerad*

2.2 Definition of Done

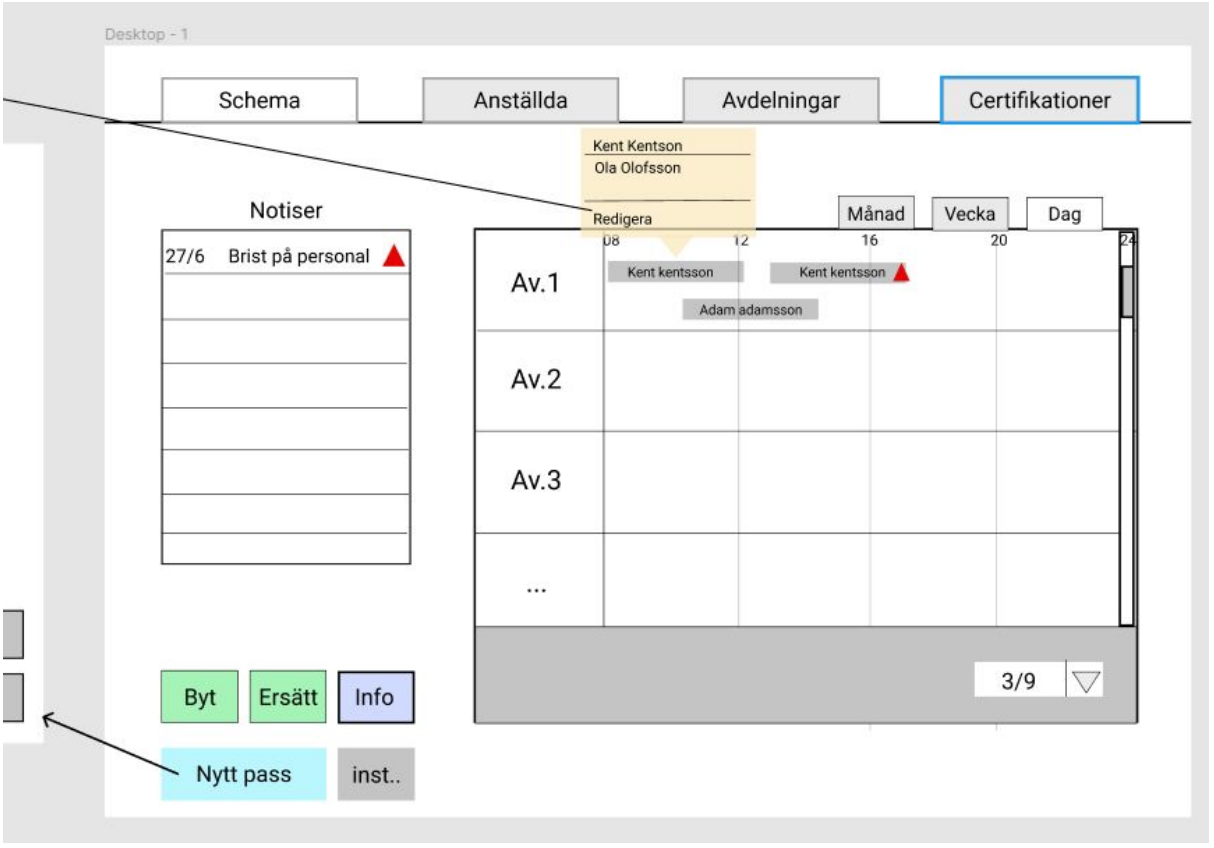
funktionell:

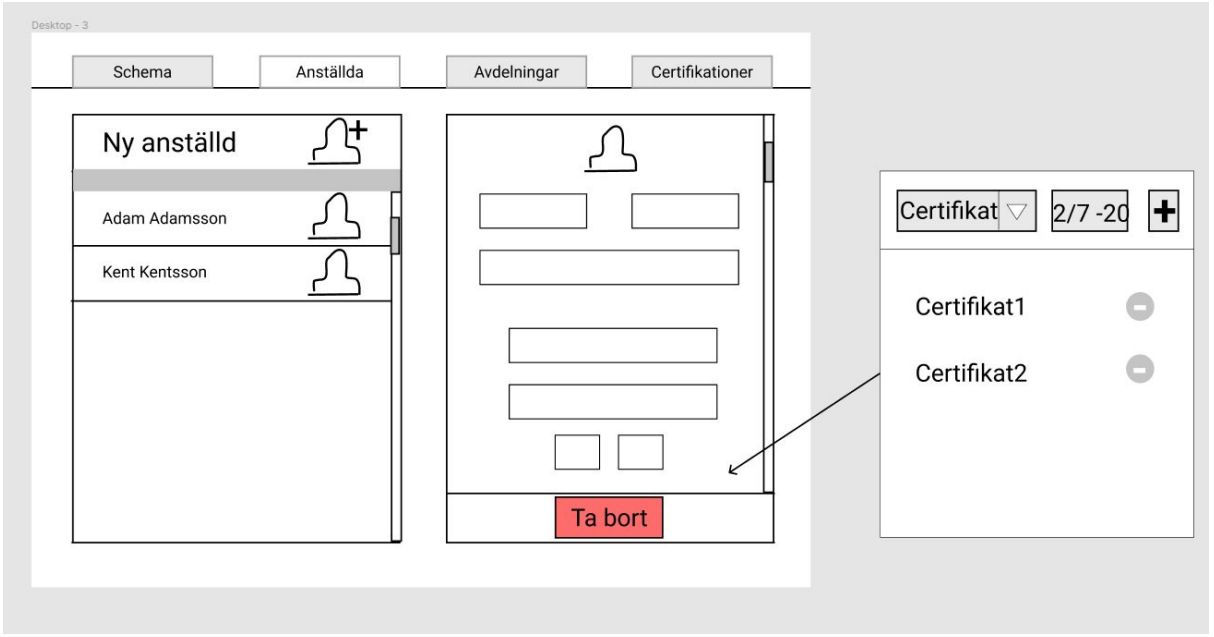
- Arbetsgivare kan mata in företagsspecifika uppgifter såsom anställda, certifikat, avdelningar, x(se ovan) och pass
- visuellt schema genereras
- det är möjligt att uppdatera företagsuppgifter
- det är möjligt att byta pass
- det är möjligt att ta bort och lägga till pass
- schemat visar vilka som jobbar, vilken tid och avdelning

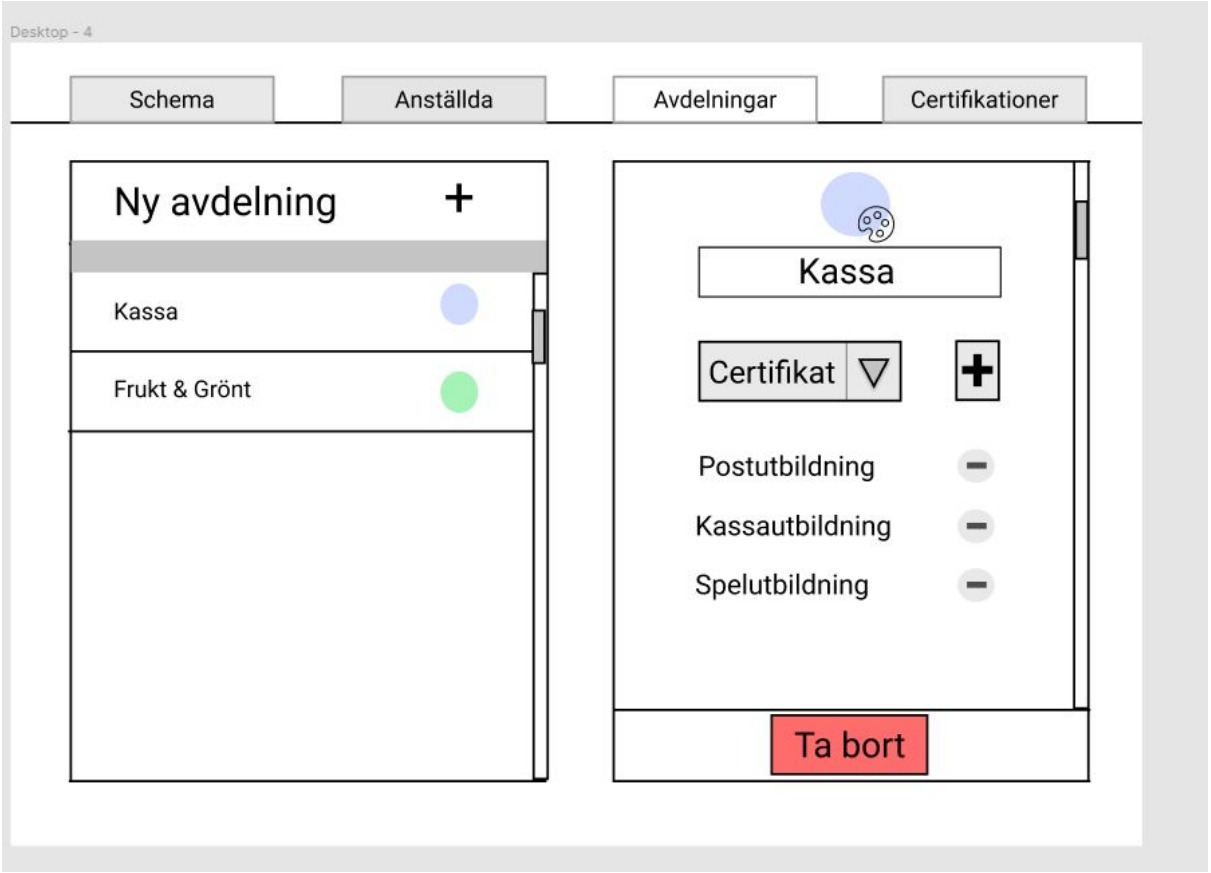
icke-funktionell:

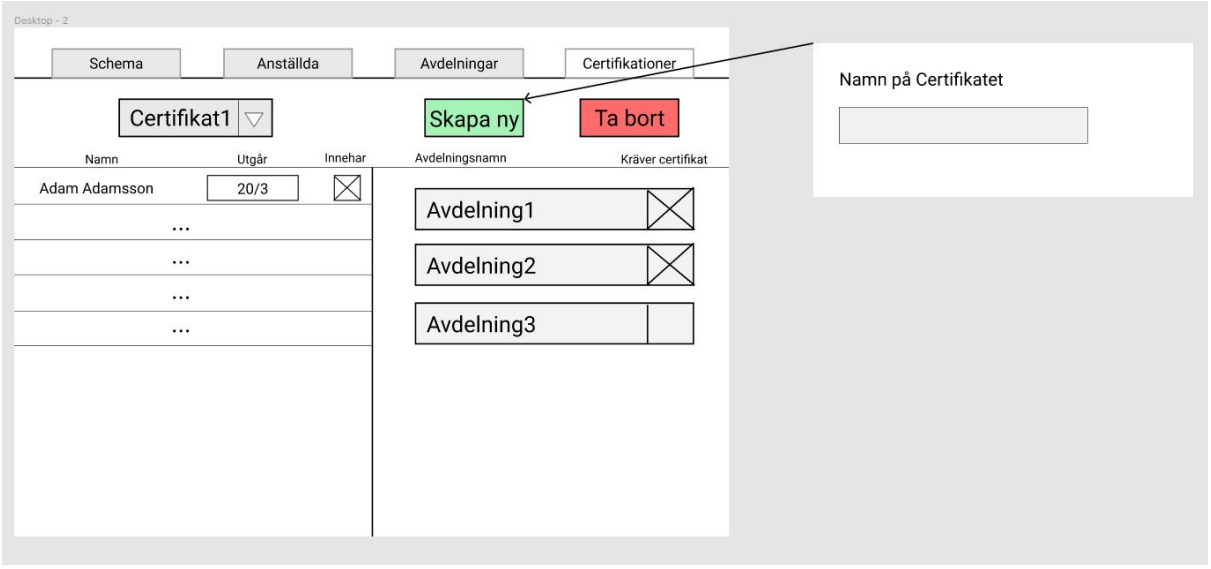
- programmet är utförligt testat med tydlig struktur av tester
- koden är kommenterad med javadoc
- alla klasser innehåller header comments innehållande author, responsibility, used by, uses
- Användaren behöver inte lägga tid på att skapa ett schema annat än när programmet efterfrågar

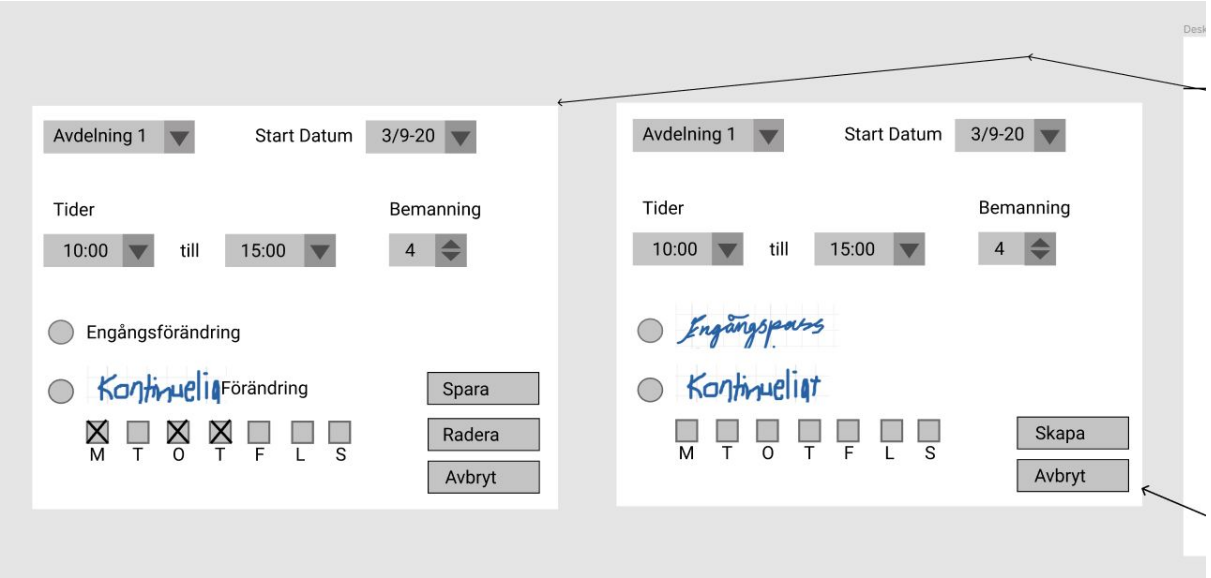
2.3 User Interface



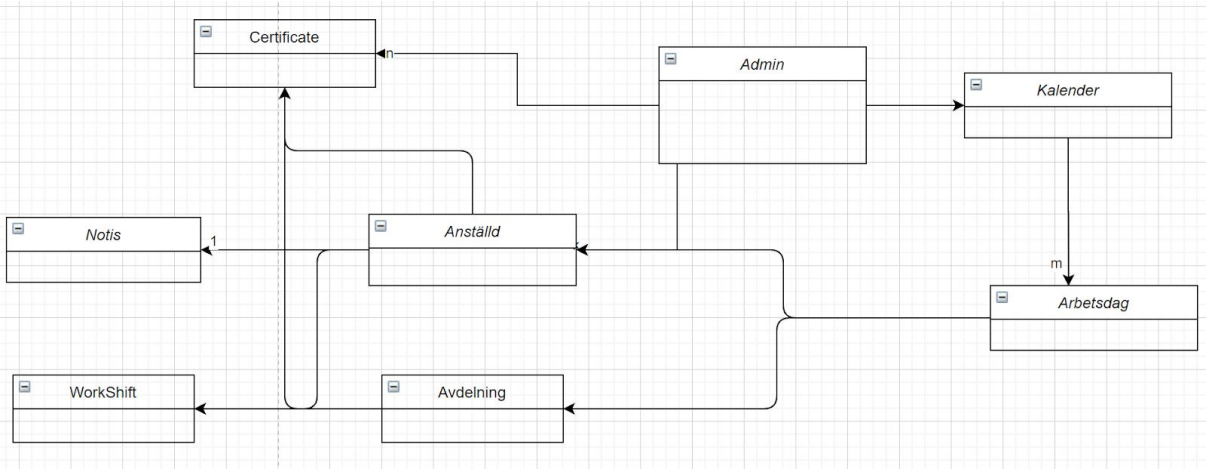



Desktop - 4
Schema
Anställda
Avdelningar
Certifikationer
Ny avdelning
+
Kassa
Frukt & Grönt
Kassa
Certifikat
Postutbildning
Kassautbildning
Spelutbildning
Ta bort


Desktop - 2
Schema
Anställda
Avdelningar
Certifikationer
Certifikat1
Skapa ny
Ta bort
Namn
Utgår
Innehar
Avdelningsnamn
Kräver certifikat
Adam Adamsson
20/3
...
...
...
...
Avdelning1
Avdelning2
Avdelning3
Namn på Certifikatet



3. **Domänmodell**



3.1. Klassernas ansvarsområden

- Admin: Huvudklassen som ansvarar för kommunikation mellan klasserna. Den delegerar olika uppgifter till de olika klasserna för att inte ha kommunikation/beroenden mellan många olika klasser.
- BreakHandler: Hanterar längden av raster
- Certificate: Representerar certifikat som arbetspass kan kräva av personalen för att kunna jobba på just det arbetspasset.
- CertificateHandler: Hanterar certifikat, vilket innebär att den både förmedlar och tar bort certifikat från anställda (via Admin).
- Department: Representerar en avdelning med alla dess planerade arbetspass. Avdelningen ser till att alla som jobbar samtidigt på avdelningen blir tilldelade raster utifrån hur avdelningen minst måste vara bemannad.
- Employee: Representerar en anställd med dess personuppgifter och tilldelade certifikat. Har även koll på när den inte kan jobba, alltså är beviljad ledighet eller när den redan arbetar.

- EmployeeCertificate: Representerar certifikat som en anställd kan bli tilldelad. EmloyeeCertificate kan göras tidsbestämda.
- EmployeeSorter: Sorterar ut alla anställda på arbetspass utifrån vilka certifikat de har så att anställda inte blir schemalagda på en arbetspass som kräver andra certifikat. Notifierar admin om schemaläggningen resulterar i att pass inte går att fylla med lämplig personal. *Just nu tar programmet inte hänsyn till vilken typ av anställning personal har (ex 50% osv), utan den schemalägger så den anställde med minst ockuperad tid blir tilldelad i första hand om den är lämplig (är inte upptagen och har rätt certifikat).*
- ImportServicePackage: Hårdkodad data som modellen hanterar som om den kom från en laddad fil om användaren väljer att “ladda in fil” i GUI
- Login: Möjliggör att användaren måste skriva in användarnamn och lösenord för att komma till startsidan av applikationen
- OccupiedTime: Representerar vilka tider en anställd inte är tillgänglig för nya arbetspass. Admin väljer hur lång ledighet som lovas efter ett arbetspass.
- OurCalendar: Representerar en kalender fylld med arbetsdagar(Workday) som är möjlig att hantera i månader, veckor och dagar.
- SendNotification: Skickar ett mail till anställdas mailadress när de blivit tilldelad ett pass och mail till admins mailadress när ett pass inte gick att fylla vid schemaläggning.
- WeekHandler: Har statiska metoder som möjliggör att enkelt addera tid till ett date-objekt utan att varje gång gör om till millisekunder.
- WorkDay: Representerar en arbetsdag på ett visst datum som håller koll på alla arbetspass på den specifika dagen, vilka avdelningar dessa tillhör och om de är fyllda eller inte. Möjliggör byten mellan olika arbetspass och förändring av bemanning om de är kvalificerade.
- WorkShift: Representerar ett arbetspass som kan upprepas om så är valt. Detta har själv koll på om det är bemannat eller ej och kan tilldela anställda till det.

4. Referenser

- JavaFX https://openjfx.io/
- Maven https://maven.apache.org/
- IntelliJ stöd för Dependency Analys
- Javax.activation https://docs.oracle.com/javase/8/docs/api/javax/activation/package-summary.html
- Javax.mail https://javaee.github.io/javamail/

SDD

23/10 2020

1. Introduktion

Att skapa ett väl fungerande schema är en mycket viktig del för att få till en stabil arbetsmiljö. Men scheman och personalen som ska följa schemat är dynamiska och kan förändras snabbt vilket kan rubba stora delar av schemat då alla krav kan vara svåra att fylla upp. Det är därför NF (new five) har utvecklat en schemaläggare som snabbt och effektivt ska kunna placera ut anställda på ett schema efter deras behörigheter och erbjuda ett lätt sätt för att snabbt kunna strukturera om ett schema.

1.1 Ordlista

Certifikat - arbetsrelaterad licens som en anställd kan bli tilldelad

2. Systemarkitektur

Det första som händer när applikationen startar är att man måste logga in med användarnamn och lösenord för att sedan få upp ett prompt om handlingar; “Ny fil”, “Ladda fil”, “Spara & Avsluta”. Väljer man “ny fil” körs programmet utan argument och man får en blank kanfas att arbeta på. Väljer man “Ladda fil” så läser programmet in all data från ett servicepaket med hårdkodad data, för att visa på hur man skulle kunna hämta gammal data.

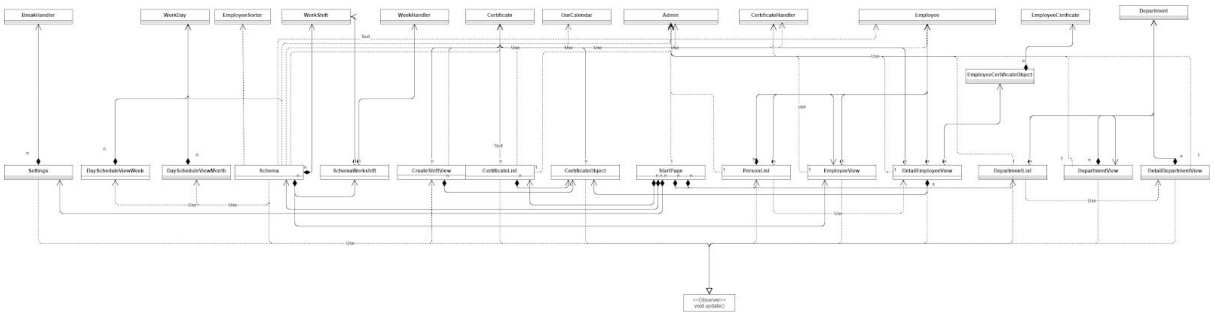
När man väl startat framgår en vy med ett schema med alla dagar i den nuvarande månaden samt att de olika dagarna är röda om alla arbetspass inte är bemannade med personal och gröna om de är bemannade. Längst upp finns flikar för olika vyer att hantera schemat, anställda, certifikat, avdelningar och inställningar i. Det är möjligt att skapa nya avdelningar och därefter skapa arbetspass i dessa. Arbetspassen kan tilldelas att kräva olika certifikat av anställda för att kunna arbeta på det specifika arbetspasset. Sedan kan man delegera arbetare som har rätt utbildning i form av certifikat på dessa arbetspass. Detta går att göra både manuellt av schemaläggaren men också automatiskt med hjälp av en sorterare som finns i programmet. Denna sorterare har en algoritm som ser till att alla pass fylls med behörig personal i största möjliga mån. Sorteraren schemalägger en vecka framåt när användaren efterfrågar det genom att klicka på en knapp i schemavyn.

Denna sorterare har en enkel algoritm som börjar med att kolla vilka anställda som är behöriga att arbeta på vilka arbetspass. Sedan sorteras denna lista efter vilka pass som har minst personal som kan jobba där. Efter det kollar den efter det pass med lägst potentiella arbetare och delegerar till det arbetspasset efter vilka anställda som har minst arbetspass sedan innan. Om den inte lyckas fylla alla arbetspass skickar den ett mail till användarens mailadress med vilka arbetspass som saknar anställda. När anställda blivit tilldelade arbetspass skickas mail till dessa för att informera.

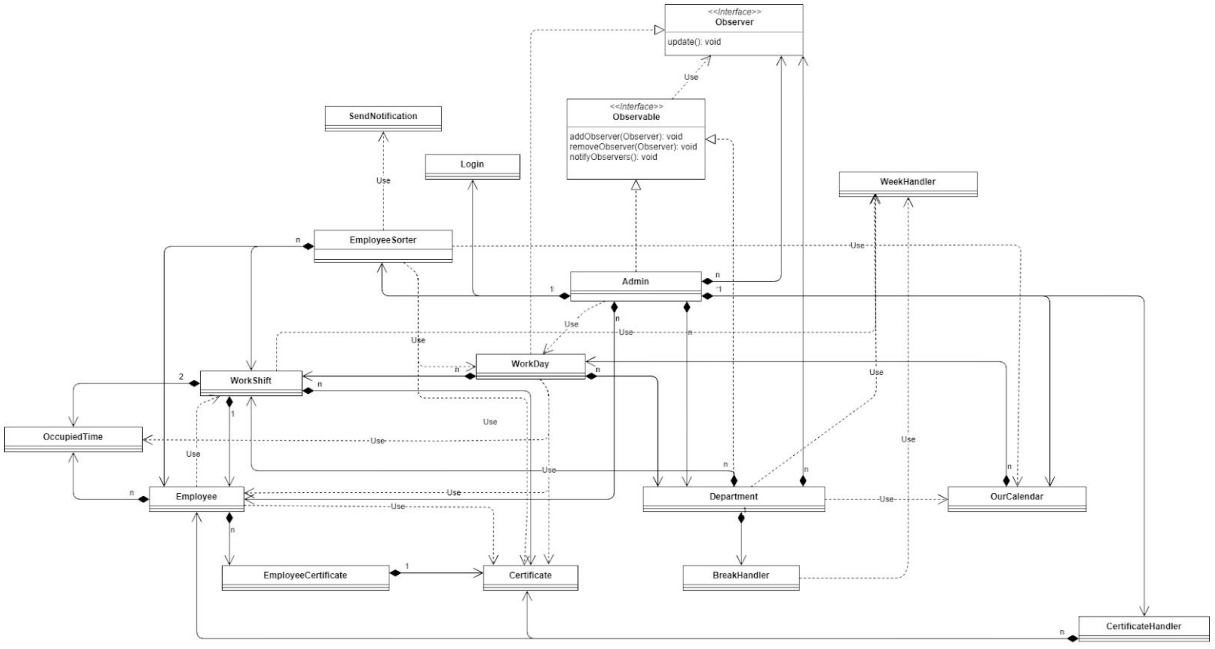
3. Systemdesign

MVC designen i projektet fungerar på så vis att modellen sköter all backend medans det är en integrerad controller inuti view, då det bara finns kontrollers som både agerar som en kontroller men samtidigt visar information. Detta gör att om man byggt vanligt MVC så hade man behövt skicka från vyn till kontroller och sedan till modellen.

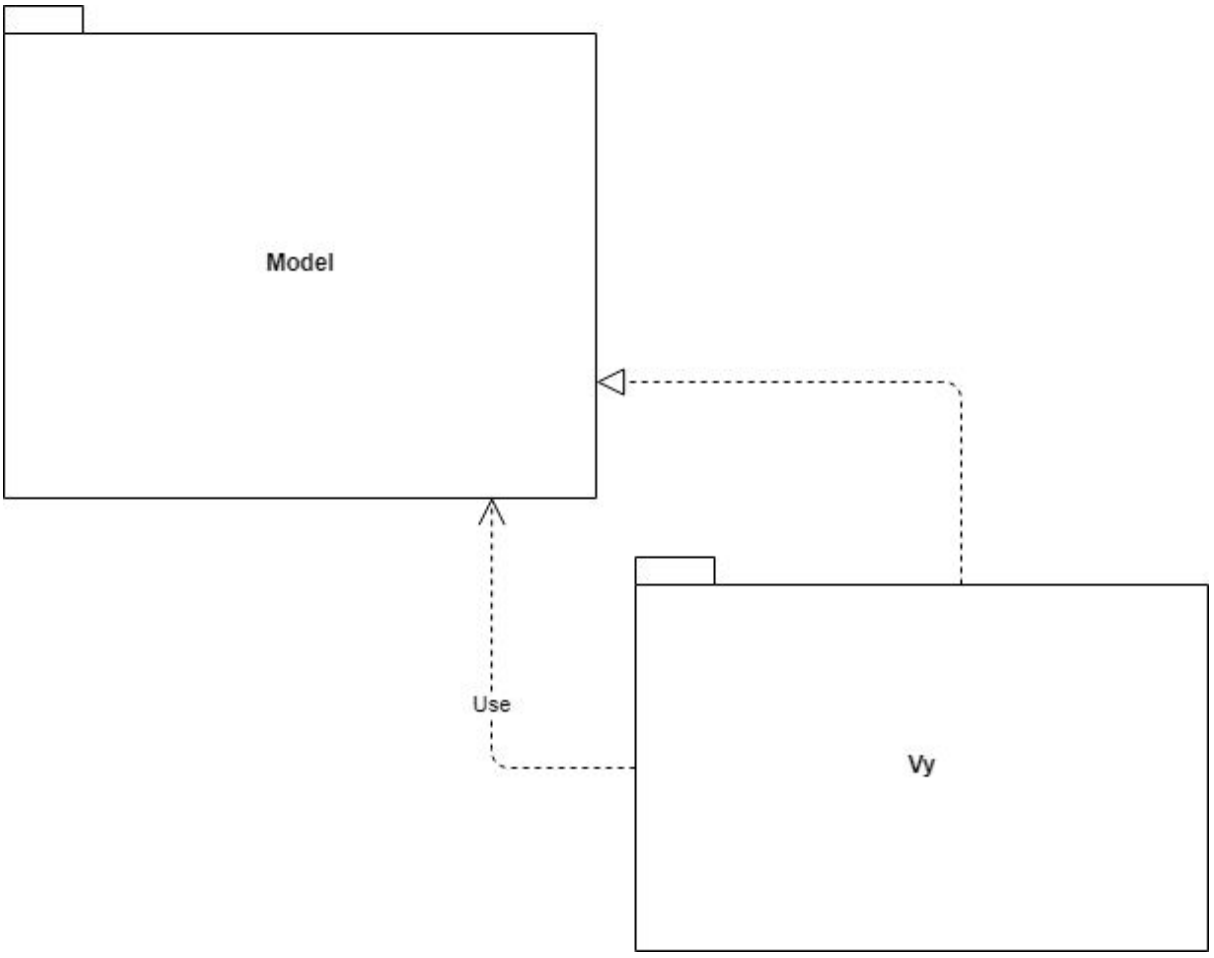
View:
Använder sig av observer pattern för att behålla mvc. Admin har också en getInstance()-funktion för att kunna skapa nya viewklasser utan att ändra tidigare skapade klasser.



Model Package:



Module-level:



Design Patterns:

- **Singleton** - används då Admin, OurCalender, CertificateHandler, BreakHandler enbart går att skapa en gång, för att alltid använda enbart den instansen
- **Observer** - för att kunna uppdatera vyn utan att vyen ska behöva fråga efter uppdatering från model- den uppdateras när uppdatering finns då modellen notifierar vyn om det. Dessutom utnyttjar Department och Workday mönstret så att alla WorkDays observerar Department efter nya WorkShifts som ska läggas till eller tas bort från alla WorkDays. Förut behövdes en metod som man måste kalla på i varje workday men om man kallade på metoden fler gånger så sattes dubbletter av WorkShifts in och det var svårt att veta vilka som ska tas bort om vissa work shifts endast ska vara med en dag vilket nästan gjorde det omöjligt att ta bort ett WorkShift från flera WorkDays.

4. Datahantering

Eftersom fokus inte legat på att skapa en databas så är detta ett område där applikationen kan förbättras. Nuvarande så finns ett servicepaket med hårdkodad information så att det ska gå att demonstrera hur det hade fungerat om man laddat data från en .zip.

För att i framtiden på ett smidigt och enkelt sätt kunna lagra data så hade det valts att alla listor (ArrayList och List) skulle lagras i varsin textfil (.txt) eftersom man i förväg aldrig vet hur stora listorna är och det kan bli svårt att hitta information om allt ligger i samma fil. När det kommer HashMap och Map så hade det valts att det ligger en fil bland de andra textfilerna med varje nyckel. Detta ligger i nummerordning vilken gör det lätt att skapa en ny mapp som har samma namn som variabelnamnet på HashMap. Detta hade gjort det enkelt att hitta vilken lista eller variabel som korresponderar till nyckeln. Eftersom det finns andra variabler med känd längd från start så hade det valts att det skulle finnas en textfil per Javapaket där de finns ordnade i en logisk struktur.

För att kunna ladda gamla scheman så skulle allt sparas i en krypterad .zip där all data lagts in. För att enkelt hitta zip-filen skulle den döpas till dagens datum och tid. t.ex. 11-06-2020-09-29. Detta skulle göra det enkelt att ladda den .zip man vill. När man valt en .zip som man vill använda så dekrypterar och unzippar applikationen filen och lägger den i applikationens mapp så att den kan nå all data och sedan omzippa den nya datan.

5. Kvalitet

För nästan varje klass så skapas en tillhörande testklass under en testmapp med samma namn + test. Målet med dessa testklasser är att testa varje funktion av den tillhörande klassen. Problemet är att man inte kan förutspå när eller om det kommer fallera i ett gränsfall då gränsfall inte är lätt definierade när variablerna är så ickelinjära.
Dependency-analys finns i git-projektet under dependencyAnalysis/“dependency” + versionsnummer

Har vi några kända problem?
En svaghet i programmet är att schemaläggningsalgoritmen just nu utgår från att den anställde med minst ockuperad tid blir tilldelad i första hand om den är lämplig (är inte upptagen och har rätt certifikat). Alltså kommer den person med mest ledighet inte bli tilldelad pass i första hand, eftersom ledighet också är ockuperad tid. Detta är en svaghet för att applicera programmet i verkligheten men duger som prototyp.

5.1 Åtkomstkontroll och säkerhet

När man först startar applikationen så kommer man till en logga in sida för att all information inte ska vara lättåtkomlig. Anledningen till att en säkerhetskontroll såsom logga in är åtråvärt kan vara att vi hanterar personlig information samt att en obehörig ej ska kunna ändra i schemat, t.ex. en anställd som inte är nöjd med sitt schema. I nulägen finns det en krypteringsalgoritm som krypterar lösenordet till användare, men i framtiden när man vill kunna ta in känsliga data ska även detta kunna krypteras på samma sätt som lösenord.

Vi har gjort koden säkrare genom att arbeta med icke-muterbara listor genom att det aldrig gå att få tag på en hel lista, utan man får tag på objekten via metoder med index för listan. Att göra objekt immutable leder till att beroenden på dessa objekt blir säkrare och man undviker alias-problematik.

6. Referenser

- JavaFX https://openjfx.io/
- Maven https://maven.apache.org/
- IntelliJ stöd för dependency analys
- Javax.mail https://javaee.github.io/javamail/
- Javax.activation
  https://docs.oracle.com/javase/8/docs/api/javax/activation/package-summary.html

Peer Review

- Do the design and implementation follow design principles?
    - Does the project use a consistent coding style?

Det är lite utspritt om man har skrivit ett interface i en ny fil eller om man skrivit dem i samma fil som en klass (Subscription har både en klass och två interface medan ISearch är ett eget interface).

    - Is the code reusable?

Mycket av koden skulle kunna användas även till andra projekt, t.ex. så skulle man kunna använda BarcodeScanner även till andra saker man kan scanna. Det som är lite fel är att man även här kollar om det är en ISBN på samma ställe istället för att lägga det i modellen vilket kan skapa problem om man vill använda den i en annan applikation.

    - Is it easy to maintain?

Då koden följer OCP genom att den är abstraherad och inte beror av något hårdkodat gör att den är enkel att underhålla.

    - Can we easily add/remove functionality?

Då det inte är mycket “hårdkodat” och paketen i ui har low coupling till varandra är det inte något problem att varken ta bort eller lägga till delar av funktionaliteten. Exempelvis är det inget (mer än Firebase) som beror av login-mappen och den skulle smidigt kunna tas bort.

    - Are design patterns used?

Adapter pattern används på ett flertal ställen på korrekt sätt eftersom man eftersöker RecyclerViews funktionalitet vilket kräver en adapter.

- Is the code documented?

I modellen är metoderna generellt väl kommenterade, men saknas info i header comments (used by och uses). Klassen Subscription har fel header comment (kopierad från klassen Ad). Konstruktorn i Subscription har felaktig javadoc då alla variabler inte sätts till null (userID och isbn initialiseras inte). Klassen User saknar javadoc helt. UI är inte väl kommenterad, inte heller mappen cameraUtil.

Kommentarer i Subscription.java angående userId är inte konsekventa.

- Are proper names used?

variabelnamn är vaga, exempelvis namn som “c”, “v” hade kunnat hållas till att alltid istället heta “context” och “view” som de gör på vissa ställen och är mycket tydligare. Se exempelvis skillnad mellan klassen BookItem med tydliga variabelnamn, och BookItemAdapter med otydligare benämningar till samma typer av variabler. Det är tydliga metodnamn och klassnamn genom all kod

- Is the design modular? Are there any unnecessary dependencies?

Det verkar vara low coupling mellan modulerna, vilket är bra för att få en modulär struktur på sin kod. Det är samtidigt t.ex. high cohesion inne i search mappen, vilket även det är bra för att få en modulär kod.

- Does the code use proper abstractions?

Ja, de klasser som utnyttjar abstraktion behöver det för att utnyttja vy komponenter. Men i modell så behövs inga abstraktioner då varje klass är självständig och inte behöver brytas ner i mer detaljerade klasser.

- Is the code well tested?

2 av 4 modellklasser testas samt en kontrollklass. Kontrollklassen testas helt medan de andra saknar några tester som kan vara nödvändiga att testa. Dock så testas den viktigaste funktionaliteten hos modellklasserna och kvarlämnar för det mesta getters men även några andra viktigare metoder. Dessutom behövs lite omstrukturering bland testklasserna då alla 3 ligger i model-testmappen, trots att Search-klassen inte ligger i Model. Dessutom testas vyn (ui) men en vy ska inte behöva testas med junit (tyder på att vyn är för smart, om man förhåller sig till MVC).

- Are there any security problems, are there any performance issues?

Man har valt att göra objekt immutable man tar och kopierar objekt för att sedan använda det nya objektet med setters och getters. Exempel på detta är Book och BookItem där Book endast har getters vilket gör att man inte kan ändra på den, detta kan man istället göra via

BookItem där man även lägger till setters. Förutom kameran som redan påpekats vara långsam(SDD) så var det inga problem med att köra appen.

- Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

Svårt att förstå sig på då UML inte är av samma version som koden (vyn saknas). Controller i UML innehåller klasser som istället ligger under ui i koden. I kontrollermappen i koden finns det ett ISearch interface och en klass Search, vilket inte stämmer överens med UML diagrammet. Mappen som heter ui tolkas vara View och i nuläget ser det ut som om den också har hand om funktionalitet som hör till Controller och Model. Exempelvis klassen AddAdFunctionality inuti ui har metoder som checkInputs(...) vilket snarare borde höra till en isolerad Model.

- Can the design or code be improved? Are there better solutions?

Implementera MVC med förhållningssättet att UI borde vara “dum” och inte ha så mycket funktionalitet som den har nu (ex AddAdActivity). Strukturera tester, kommentera fullständigt även header comments.