System design document for scheduling application

Markus Grahn, Oliver Andersson, Christian Lind, Victor Cousin, Moa Berglund

23/10 2020

version 2

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.

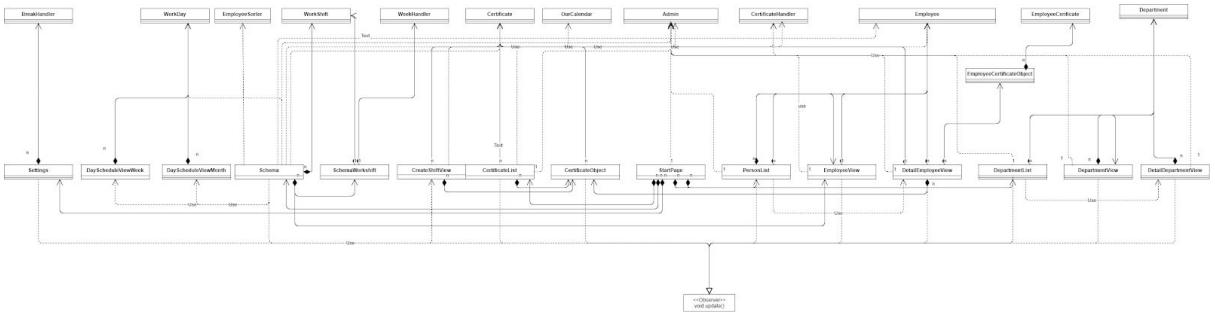
När man väl startat framgår en vy med ett schema med alla dagar i den nuvarande månaden samt att de olika dagarna är röda om alla arbetspass inte är bemannade med personal och gröna om de är bemannade. Längst upp finns flikar för olika vyer att hantera schemat, anställda, certifikat, avdelningar och 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.

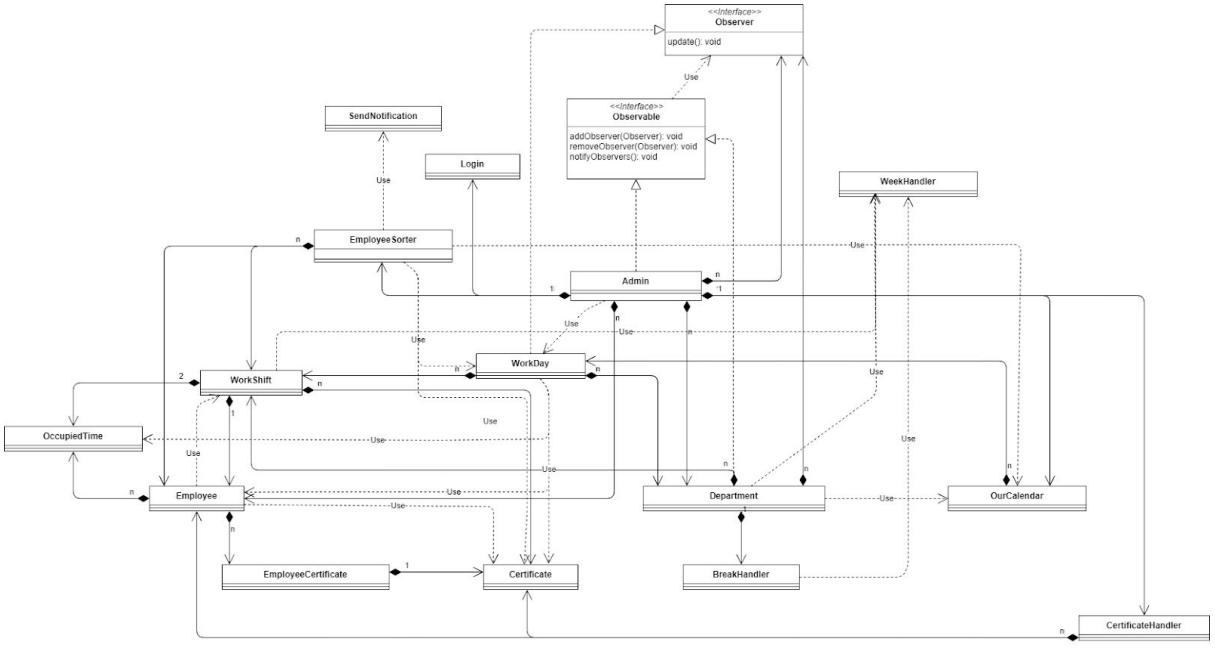
3. Systemdesign

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.

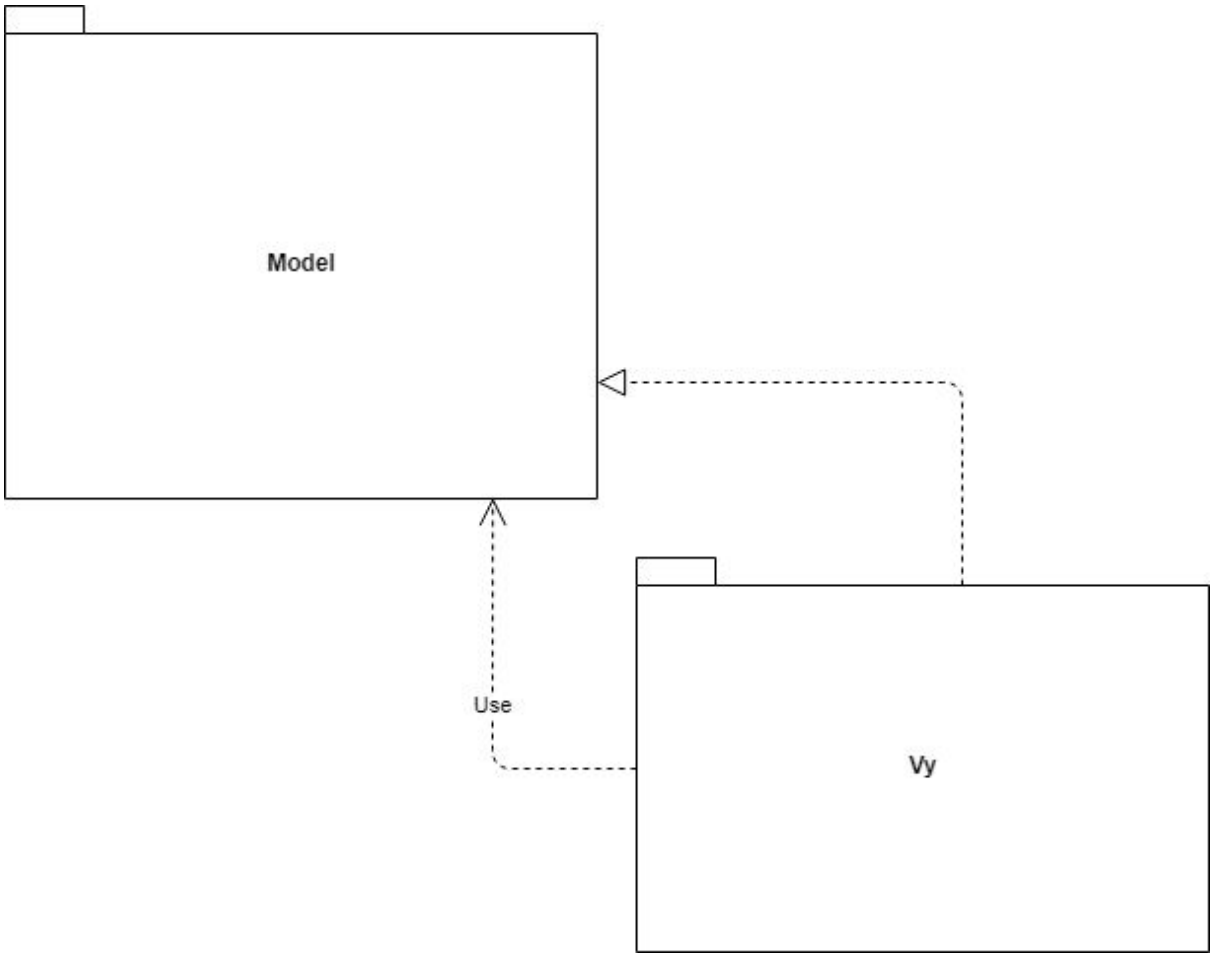
View:
Använder sig av observer pattern för att behålla mvc. Admin har också en getInstance()-funktion för att kunna skapa nya viewklasser utan att ändra tidigare skapade klasser.



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