Requirements and Analysis Document for StoreIT

Pär Aronsson William Albertsson Jonathan Eksberg
Hugo Stegrell Carl Lindh

25 oktober 2019

Innehåll

1 Introduktion

1.1 Bakgrund

Projektets idé uppkom från ett problem alla i gruppen kände igen. Då gruppens medlemmar är aktiva i föreningar, så är alla medvetna om att de förråd som finns är ostrukturerade och fyllda med olika föremål. Detta gör det svårt att hitta vissa föremål när de behövs, samt svårt att ha en överblick på vad som faktiskt finns.

Ytterligare en svårighet som finns mellan föreningar är hur lån av föremål ska gå till. Godkännande av förfrågningar och eventuellt kontrakt för betalning av föremål som gått sönder eller hanterats ovarsamt är några av de sakerna som hör till otydligheterna vid utlåning.

Som utvecklare ligger intresset i att utveckla en applikation som även skulle gynna gruppens medlemmar, då dilemmat finner sig för alla i vardagen. Denna applikation gynnar således främst olika föreningar som tillhör en viss organisation, där föremål ofta lånas inom organisationen, utan någon betalning.

1.2 Applikations funktionalitet

Detta projekt har riktat in sig på att lösa tidigare nämnda problem och skapa ett smidigare sätt för belåning och utlåning. Applikationen kommer även att ge en överblick över sitt eget lags inventarie, vilket kan vara en fördel för bland annat nya personer i de olika lagen. Applikationen kommer visa nödvändig information om de olika föremålen, såsom bokningsbara tider, föremålets förvaringsplats och i vilket skick det är i. Ett annat lags föremål kan då reserveras i ett valt intervall, varefter en användare i laget som äger föremålet kan bekräfta eller neka reservationen.

2 Krav och förväntningar

2.1 User Stories

Här är de user stories som projektet utgått ifrån. De är prioriterade i 4 kategorier, med viss inbördes ordning i Prioritet 1 och Prioritet 2. För user stories som är åtminstone påbörjade finns förväntad funktionalitet och beteende.

2.1.1 Implementerade User Stories

Nedan följer de user stories som har blivit implementerade i programmet:

Prioritet 1

Story Identifier: ID 1
User Story: Körbart program

Beskrivning
Som en utvecklare, vill jag ha ett körbart program på vilket applikationen kan byggas vidare på.

Funktionalitet
- Finns det ett GitHub repository som alla medlemmar kan ta del av?
- När programmet körs, visas det då någon form av GUI?
- Kompilerar och körs programmet på alla medlemmars datorer utan några fel?

Icke-funktioner
- Fungerar det som det är tänkt på alla medlemmars datorer och utan varningar?

Story Identifier: ID 2
Story Name: Användarstruktur

Beskrivning
Som en utvecklare, vill jag ha en tydlig struktur i modellen för olika organisationer, dess lag och användare för att programmet slutligen ska uppnå en bra koddesign.

Funktionalitet
- Finns det en lista med alla användare där man kan skilja dem åt?
- Finns det en implementation av lag som har en överblick på vilka som är medlemmar i laget?
- Har organisationen en överblick på de olika lag som är med i den?
- Har ett lag en lista med vilka föremål de äger?

Icke-funktioner
- Kan jag följa med i designen och förstå det logiska förloppet?
- Är det lätt att utveckla programmet med denna design?

Story Identifier: ID 3
Story Name: Se föremål

Beskrivning
Som en användare, vill jag kunna se de olika föremålen som finns för att få en bättre uppfattning om föremålets skick, förrådsplats etc.

Funktionalitet
- Har programmet en grafisk representation av föremål som visar nödvändig information?

– Om jag klickar på ett föremål, visas då en detaljvy av det föremålet?

Icke-funktioner
– Varje gång jag ser en representation av ett föremål, kan jag då komma åt detaljvyn?
– Är informationen presenterad på ett strukturerat och lättöverskådligt sätt?

Story Identifier: ID 4
Story Name: Se reservationer

Beskrivning
Som en användare, vill jag kunna se reservationer för att få en överblick på vad som är lånat/utlånat.

Funktionalitet
– Kan jag se en reservation, som är kopplad till ett föremål?
– Kan jag se viktig informatiom som ägare, lånare och tid?
Icke-funktioner
– Kan man välja att se reservationer för lånade och utlånade föremål separat?

Story Identifier: ID 5
Story Name: Skapa reservationer

Beskrivning
Som en användare, vill jag kunna skapa en reservation för att få tillgång till ett föremål under en viss tid.

Funktionalitet
– Finns det ett UI för att reservera ett föremål?
– Finns det en representation av ett föremåls reservatiner?
Icke-funktioner
– Blockeras min bokning om föremålet redan är bokat?
– Kan man se vilket lag jag tillhör när jag skapar reservationen?

Story Identifier: ID 6
Story Name: Se allt bokningsbart

Beskrivning
Som en användare, vill jag kunna bläddra mellan föremål för att se vilka föremål som jag kan låna, samt vilka föremål som tillhör de lag jag är med i.

Funktionalitet
– Ser jag den relevata information jag behöver när jag bläddrar bland alla föremål?
– Kan jag klicka på ett item för att då få fram en tydligare beskrivning av just det föremålet?
– Finns det ett GUI för att visualisera föremål?
– Är det möjligt för lag att bläddra bland sina egna föremål?
Icke-funktioner
– Är det lätt att åtskilja dem?

Story Identifier: ID 7
Story Name: Framework

Beskrivning
Som en utvecklare, vill jag ha ett grundligt ramverk där knappar till andra vyer samt annan funktionalitet kan läggas till för att underlätta under utvecklingen av applikationen.

Funktionalitet
- Finns det ett grundläggande grafiskt ramverk för applikationen?
- Finns det knappar för länkar till vidare funktionalitet såsom: ”profile”,”your inventory”, ”team” etc.?
- Finns det en ram för applikationen, med t.ex. en meny till vänster och en toppsektion med fler knappar?

Icke-funktioner
- Kan jag alltid navigera mig runt till programmets alla delar på ett smidigt sätt?

Story Identifier: ID 8
Story Name: Skapa nya objekt

Beskrivning
Som en användare, vill jag kunna lägga till nya föremål för mitt lag därför att vi vill kunna utöka vårt inventarie.

Funktionalitet
- Finns ett GUI för att skapa nya föremål?
- Kan all väsentlig information kan väljas?

Icke-funktioner
- Kommer jag åt knappen för att lägga till föremål på ett enkelt och logiskt sätt?

Story Identifier: ID 9
Story Name: Ändra lagmedlemmar

Beskrivning
Som ett lag, vill vi kunna lägga till och ta bort medlemmar då vi vill kunna utöka laget samt ta bort medlemmar som ej längre vill eller bör vara med.

Funktionalitet
- Är det möjligt att ta bort medlemma?
- Är det möjligt att lägga till medlemmar?
- Gäller det att den borttagna användaren inte ser saker som laget ser när denne är borttagen?
- Kan den tillagda användaren se de saker som laget ser?

Icke-funktioner
- Kan alla medlemmar i ett lag lägga till/ ta bort en annan användare?

Story Identifier: ID 10
Story Name: Inställningar

Beskrivning
Som en användare, vill jag ha separata vyer för användare och för lag. Jag vill även ha dem presen-

terade på ett trevligt sätt. Detta för att få ett logiskt upplägg i applikationen.

Funktionalitet
- Finns det separate inställningsvyer för lag och för din användare där jag ser all väsentlig information?

Icke-funktioner
- Ligger all funktionalitet på rätt ställe?
- Är det en logisk uppdelning på den funktionalitet som finns?
- Kan jag ändra all information?

Story Identifier: ID 11
Story Name: Logga ut

Beskrivning
Som en användare, så vill jag kunna logga ut för att se till att information synkroniseras mellan användare.

Funktionalitet
- Finns det en knapp för att kunna byta vy till logga in -sidan?
- Sparas den information som ska delas mellan användare?

Icke-funktioner
- Kan jag logga ut närsomhelst i programmet?
- Syns ingen information från den tidigare inloggade användaren?

Prioritet 2

Story Identifier: ID 12
Story Name: Datahantering

Beskrivning
Som en användare, vill jag att all information om items, bokningar etc. ska sparas mellan körningar för att jag vill kunna redigera dem vid ett senare tillfälle.

Funktionalitet
- Sparas alla data som förväntat mellan körningar?

Icke-funktioner
- NA

Story Identifier: ID 13
Story Name: En användares lag

Beskrivning
Som en användare, vill jag kunna vara med i flera lag och kunna ändra så att det relevanta laget är ansvarigt vid en ev. bokning.

Funktionalitet
- Kan jag, när val av lag är relevant, välja lag?

Icke-funktioner
- Sparas valet till senare val?

Story Identifier: ID 14
Story Name: Konto

Beskrivning
Som en användare, vill jag ha ett konto för att försäkra mig om att ingen utomstående kan ändra någon data eller komma åt min information.

Funktionalitet
- Finns ett personligt konto med användarnamn och lösenord som är specifikt för mig?
- Ger detta konto mig tillgång till all information som är relvant för mig?

Icke-funktioner
- Gör kontot att ingen annan kommer åt information enbart jag borde komma åt?

Story Identifier: ID 15
Story Name: Framework

Beskrivning
Som ett lag, vill vi kunna godkänna en bokning av vårt item för att se till att vi inte vill använda just det föremålet då.

Funktionalitet
- Kan jag se alla reservationer mitt lag inte bekräftat?
- Går det att bekräfta?

Icke-funktioner
- Är det bara föremålets ägare som kan bekräfta bokningar av föremålet?

Story Identifier: ID 16
Story Name: Icke-reserverbara föremål

Beskrivning
Som ett lag, vill vi skilja på föremål som vi vill och inte vill låna ut. Detta för att kunna ha hela vårt inventarie digitalt utan att låna ut det.

Funktionalitet
- Kan jag vid skapandet av föremål välja om det är publikt eller privat?
- Gör denna skillnaden att private föremål inte kan reserveras?

Icke-funktioner
- NA

Prioritet 3

Story Identifier: ID 17
Story Name: Terms

Story Identifier: ID 17
Story Name: Terms and Conditions
Beskrivning
Som ett lag, vill vi ha ”terms and conditions” för det item vi har, som en användare måste läsa och godkänna före denne kan skicka en bokningsförfrågan. Detta för att se till att våra items sköts ordentligt.

Funktionalitet
- Måste användare godkänna terms innan den kan skicka en bokningsförfrågan?

Icke-funktioner
- NA

2.1.2 Icke-Implementerade User Stories

Nedan följer de user stories som inte har blivit implementerade i programmet sorterade efter vilken prioritet de hade:

Prioritet 2

- Som en användare, vill jag ha en sökfunktion för att kunna lätt leta rätt på items mitt lag har och items jag kan låna.
- Som en användare, vill jag ha en tydlig överblick på alla avvaktande bokningsförfrågningar för att lätt kunna godkänna/avböja dessa förfrågningar.
- Som en användare, vill jag ha olika kategorier att välja emellan för att kunna begränsa min bläddring.
- Som en användare, vill jag ha ”in-app”-notifikationer för att se till att jag inte missar viktig information.
- Som ett lag, vill vi kunna avboka items som vi har lånat och/eller lånat ut. Detta då oförutsagda saker kan ske, t.ex. att vi inte längre vill låna föremålet eller så går ej vårt item att använda längre.
- Som en användare, vill jag kunna ta bort items från vårat inventarie då de har försvunnit/gått sönder.

Prioritet 3

- Som en användare, vill jag ha en tydlig överblick över relevant information (dashboard), för att t.ex. snabbt kunna uppdatera mig om bokningar som skett.
- Som en användare, vill jag kunna ändra representationen av items (rutnät, lista, etc.) för att det ska passa mina preferenser.
- Som en användare, vill jag ha en kalender för att visualisera när jag kan boka ett visst item.
- Som en användare, vill jag kunna skapa nya ”tags” för att det ska passa för mitt användningsområde.
- Som en användare, vill jag ha kontaktinformation till andra lag för att kunna ställa frågor om deras föremål och reservationer.

Prioritet 4

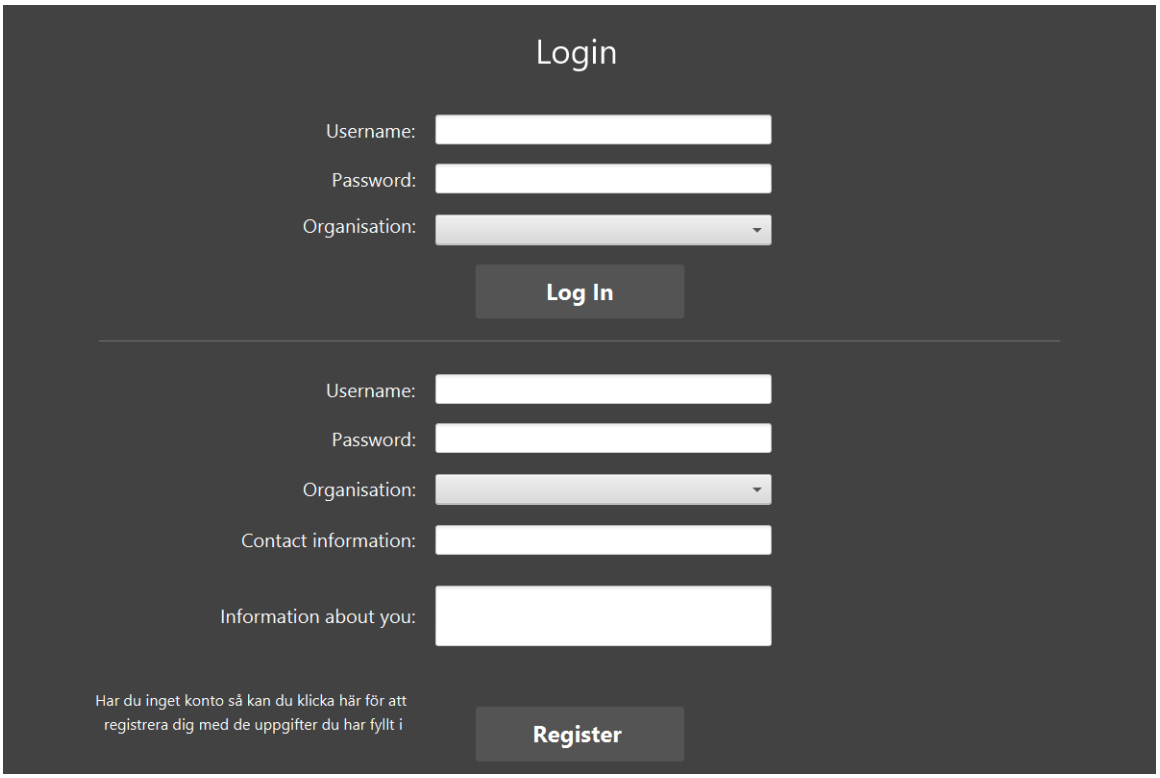
- Som en användare, vill jag få email-notifikationer om viktiga uppdateringar i applikationen för att kunna notifieras även om jag inte har programmet igång.
- Som en användare, skulle jag vilja kunna använda applikationen på vilken dator som helst, med samma organisation och ha all data synkroniserad mellan datorer.

2.2 Defintion of Done

Nedanstående punkter ska alla vara uppfyllda för att en user story ska kategoriseras som klar, men undandtag kan göras om skäl finnes och gruppen är överrens.

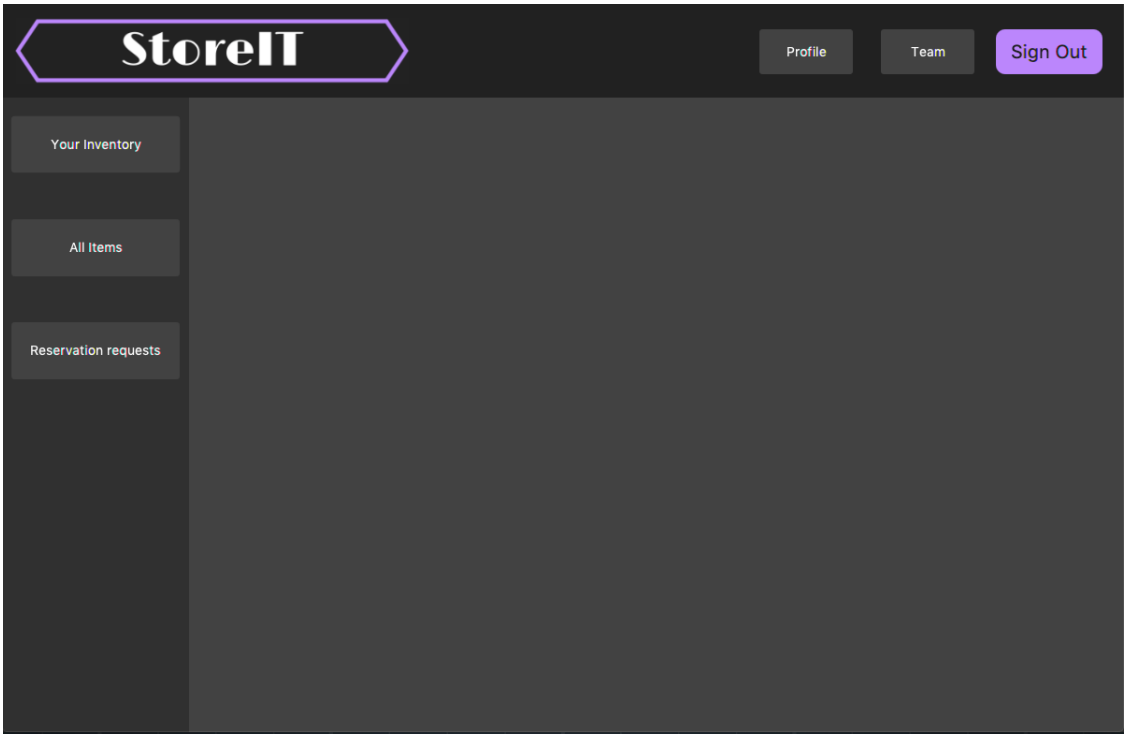
- Gruppen är överens om att det inte saknas några tasks i user storyn.
- Alla tasks är klara
- Koden har genomgått användartest av 3 personer
- Koden har genomfört JUnit tester av sådant som inte är möjligt eller svårt att testa vid användartest
- Allt som finns i filen är relevant(ingen utkommenterad kod etc)
- Koden är väldokumenterad i Javadoc
- Detta har kontrollerats av två andra i en Pull Request.
- Koden är pushad till master på git

2.3 Användargränssnitt



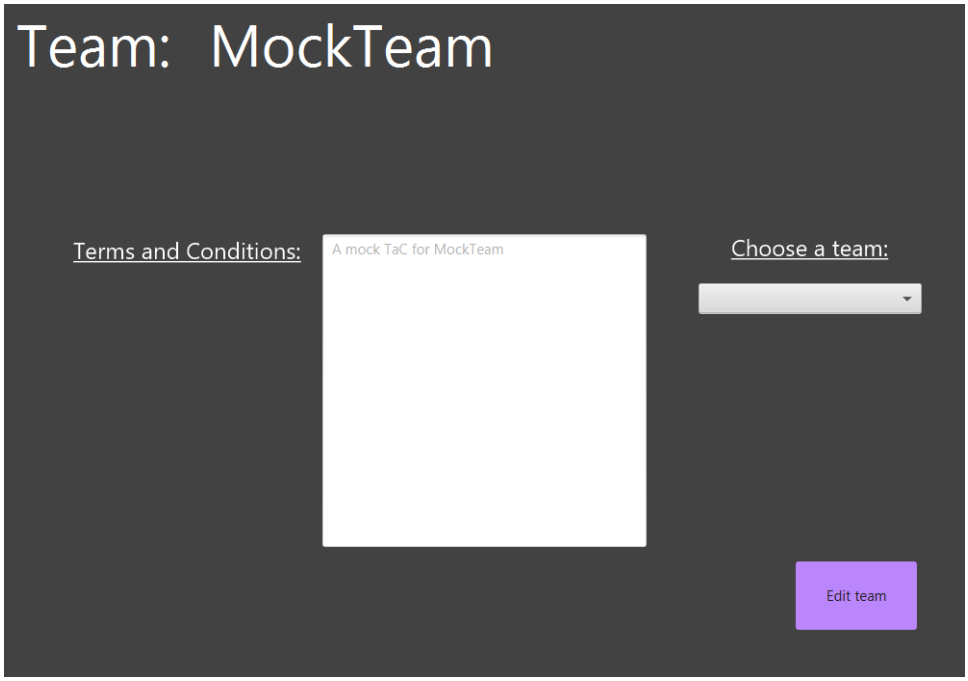
Figur 1: Hur inloggningsvyn faktiskt ser ut i programmet.

Det första som sker när man först startar applikationen är att det visas en inloggningsskärm (figur 1). Har man inget konto kan man även registrera sig där och sedan logga in. När användare trycker på knappen ”Log in” och informationen stämmer, tas den vidare in i programmet.



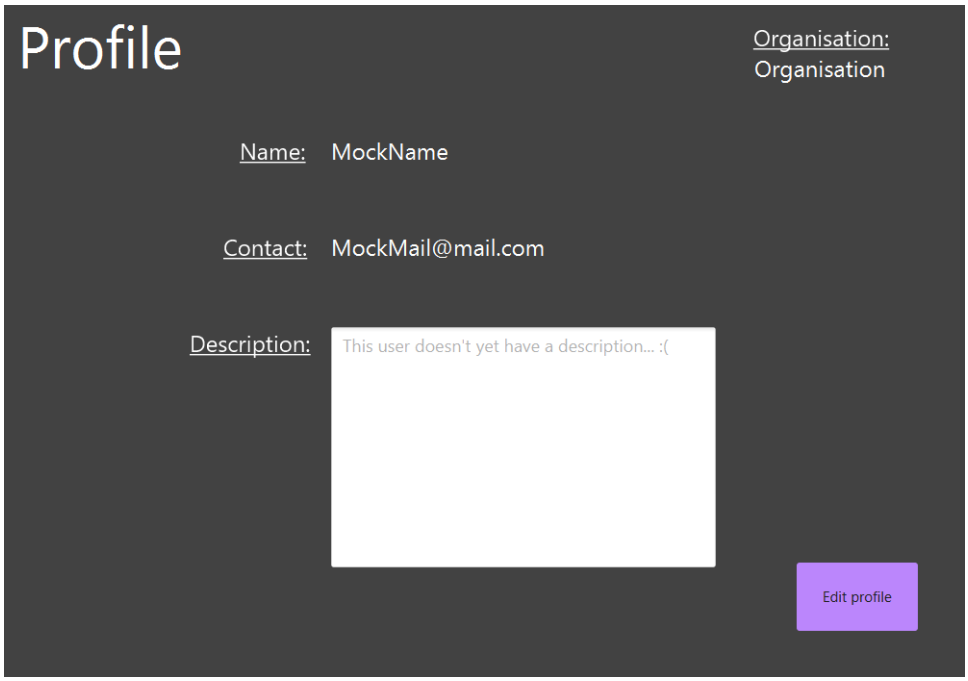
Figur 2: Hur ramverket för applikation ser ut i programmet.

Det första användaren möts av när den loggar in är applikationens ramverk som består av en header längst upp, en huvudpanel i mitten samt en sidomeny till vänster (se figur 2). Headern och sidomenyn är statiska och fungerar som navigationsmedel till de olika vyerna. Huvudpanelen ändras utefter den sida man navigerat till.
Längst upp till höger i headern finns det en knappgrupp, där knappar till vyer för användarinställningar, laginställningar och utloggning är placerade.
I sidomenyn till vänster är programmets huvudsakliga funktion samlad och de knappar som finns där för används för att ta sig vidare till dessa vyer. Navigerar användaren till ”Your Inventroy” ser den alla föremål som den har tillgång till. Under ”All Items” finns alla föremål som tillhör organisationen. I ”Reservation requests” ligger de bokningar och bokningsförfrågningar en användare har tillgång till att hantera.



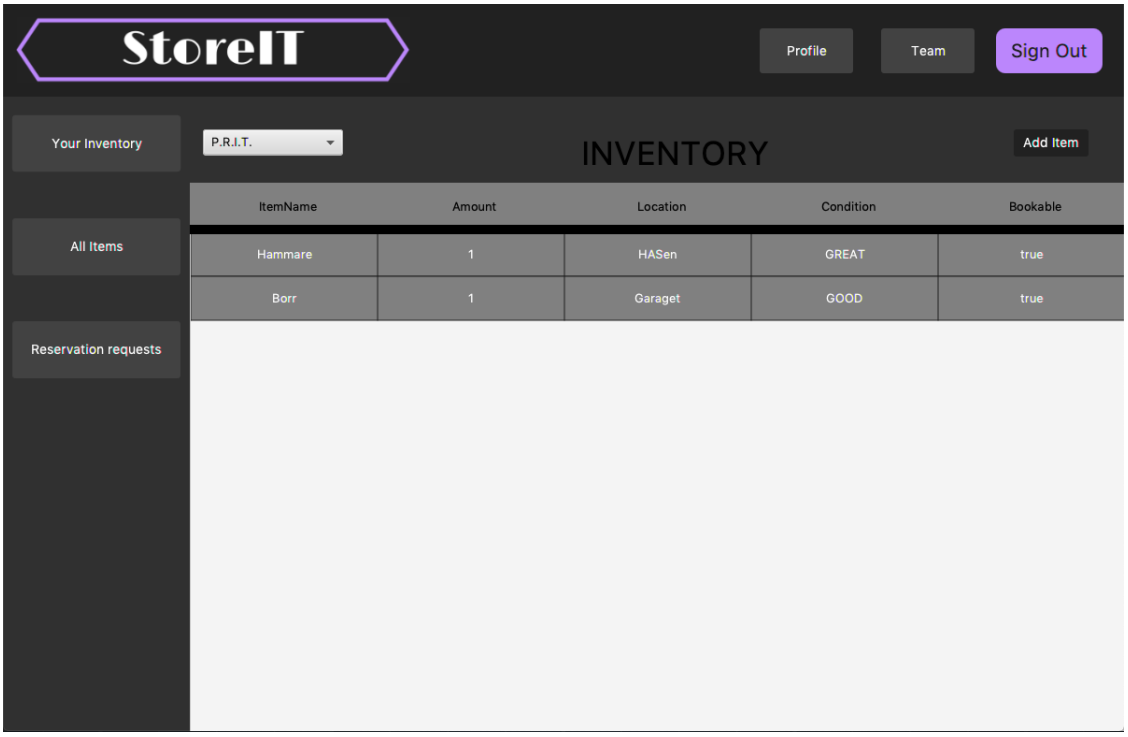
Figur 3: En vy för ett lag

Då användaren klickar på knappen ”Team” tas den vidare till en teamPage (se figur 3). Här ser användaren det valda lagets namn samt dess ”Terms and Conditions”. Genom att klicka på ”Edit team” kan användaren även navigera sig till ett redigeringsläge för det valda laget där man kan redigera information och även lägga till och ta bort medlemmar ur ett lag.



Figur 4: En vy för en användare.

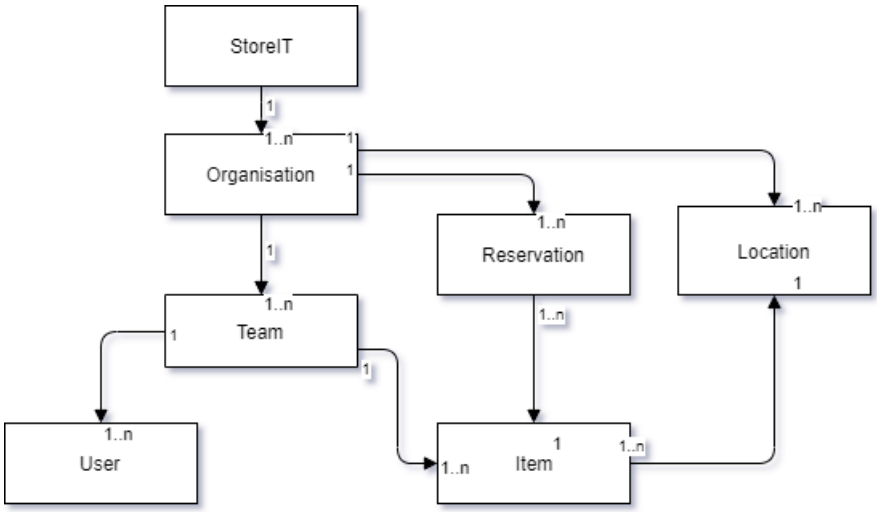
Då användaren klickar på knappen ”Profile” ta den vidare till en userPage (se figur 4. Här kan användaren se sin information om namn, kontaktuppgift, beskrivning, vilken organisation den tillhör och lösernord. Alla dessa uppgifter förutom organisation går att ändra om användaren navigerar sig vidare till ”Edit profile”.



Figur 5: En användares inventarie.

Då användaren väljer att naviger till ”Your Inventory” kan den få se alla de items i det inventarie som tillhör det lag som är valt (se figur 5). Användaren ser alla items i en lista, men kan även klicka på ett item för att få se en detaljvy som är redigerbar. Användaren kan även lägga till fler items i inventariet för att utöka det genom att klicka på ”Add Item” och fylla i den data som behövs.

3 Domänmodell



Figur 6: Domänmodell

3.1 Klassers ansvarsområde

StoreIT – Vår ingång i modellen som startar upp och håller i stora delar av modellen. Har funktionalitet som att få olika items, reservationer teams och användare som den delegerar till de delar av modellen vars ansvar det är. Känner även till nuvarande användare, team och organisation. Det är här data laddas in vid start och sparas vid avslut.

Organisation – Den högsta graden av tillhörighet. Programmet kan innehålla flera organisationer, där varje organisation bara är intresserad av innehållet i sin organisation, och inte andra organisationer. En organisation består av flera teams, och av flera users. Organisation har koll på vilka users som hör till vilka teams, där en user kan vara del av flera teams och vice versa.
Organisationen kommer åt alla reservationer som skapas i programmet
Team – Ett team har information om vilka item som ägs av teamet och vilka medlemmer som är del av det. Team har inte mycket funktionalitet annat än att lägga till och ta bort ovanstående.

User – User representerar en användare. Här finns information om användaren så som beskrivning och kontaktuppgifter men även användarenamn och lösenord.

Location – Representaion av en fysisk plats, där det finns plats för förvaring. Denna används för att visa var föremål finns, och innehåller inte mycket mer än ett namn och en beskrivning.

Reservation – En viktig del i applikationen. En reservation skapas för precis ett föremål, under ett visst tidsintervall.
Håller information om vad det är som har bokats, mellan vilka tider samt vem det är som har bokat.

Item – Ett föremål som går att låna ut. Har information såsom plats, skick och en bild på själva föremålet.

System design document for StoreIT

Pär Aronsson William Albertsson
Jonathan Eksberg Hugo Stegrell Carl Lindh

25 oktober 2019

Innehåll

1 Introduktion

Denna text kommer gå in på djupet av vad programmet gör och även förklara tekniska detaljer som datahantering, systemarkitektur, och design. Applikationen strävar efter att följa ett objektorienterat upplägg med SOLID principerna som riktlinje. Struktur och kod är därför anpassad till olika designmönster som exempelvis MVC och factory pattern.
Applikationen heter *StoreIT* och är menat som ett inventarie- och bokningsystem där användare lätt kan skapa varor och organisera dem genom applikationen. Det är bäst tillämpat för organisationer som har många separata lag som behöver kunna låna produkter av varandra. Exempel på en målgrupp är föreningar som har ett eget inventarie men är i behov av att låna saker av andra föreningar.

1.1 Definitioner och förkortningar

I kod och modeller används engelska, men i denna rapport används i första hand svenska. Ett exempel på detta kan vara item och föremål. Föremål används i första hand, men när det skrivs om klassen Item används då det.

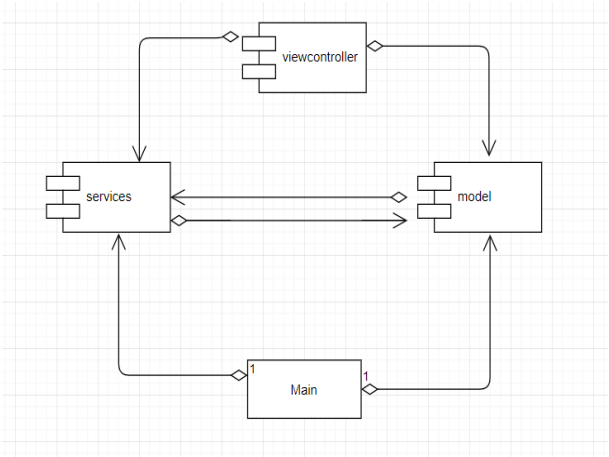
- **SOLID -** En akronym för fem olika objektorienterade principer vars syfte är att strukturera upp programmet.
- **JSON -** Ett format som sparar och hämtar data på ett visst vis åt vårt program.
- **MVC -** Ett design-mönster som ligger på ett programs övre strukturnivå.
- **JavaFX -** En mjukvaruplatform där det går att designa ett grafiskt gränssnitt i java.
- **Junit -** Ett ramverk för tester.
- **Joda-Time -** Ett bibliotek för hantering av datum och tid

2 Systemarkitektur

2.1 Programmets flow

När applikationen startar så laddar den in sparad data från en JSON-fil. Datan sparas i modellen som håller i all data under körning. Det första en användare ser är en inloggningsskärm. Där kan användaren registrera en ny användare eller logga in med en tidigare registrerad användare. Efter inloggning finns möjlighet att titta på sitt eget teams eller alla items, se alla ingående och utgående reservationer samt göra ändringar i sitt konto eller sitt team. När programmet avslutas sparas allt genom att organisationsdata från modellen skrivs in i JSON-filen `src/main/resources/json/organisationDB.json`.

3 Systemdesign



Figur 1: StoreIT's kopplingar på hög nivå

Applikationen är uppdelad i de 3 följande moduler:

- Model - Här hanteras och sparas all data under körning. Saker som logik och föremål samlas här.
- Viewcontroller - Representationen som användaren kommer att se och kunna interigera med. Vy-klasserna uppdaterar sig efter vad modellen har för data samt så uppdateras modellen när en användare interagerar med programmet genom kontroll-klasserna.
- Services - Agerar som extra modell där funktionalitet som ej hör hemma i modellen finns. Här finns JSONhandler klassen som hanterar sparandet och hämtandet av data.

Applikationen har även en huvudklass:

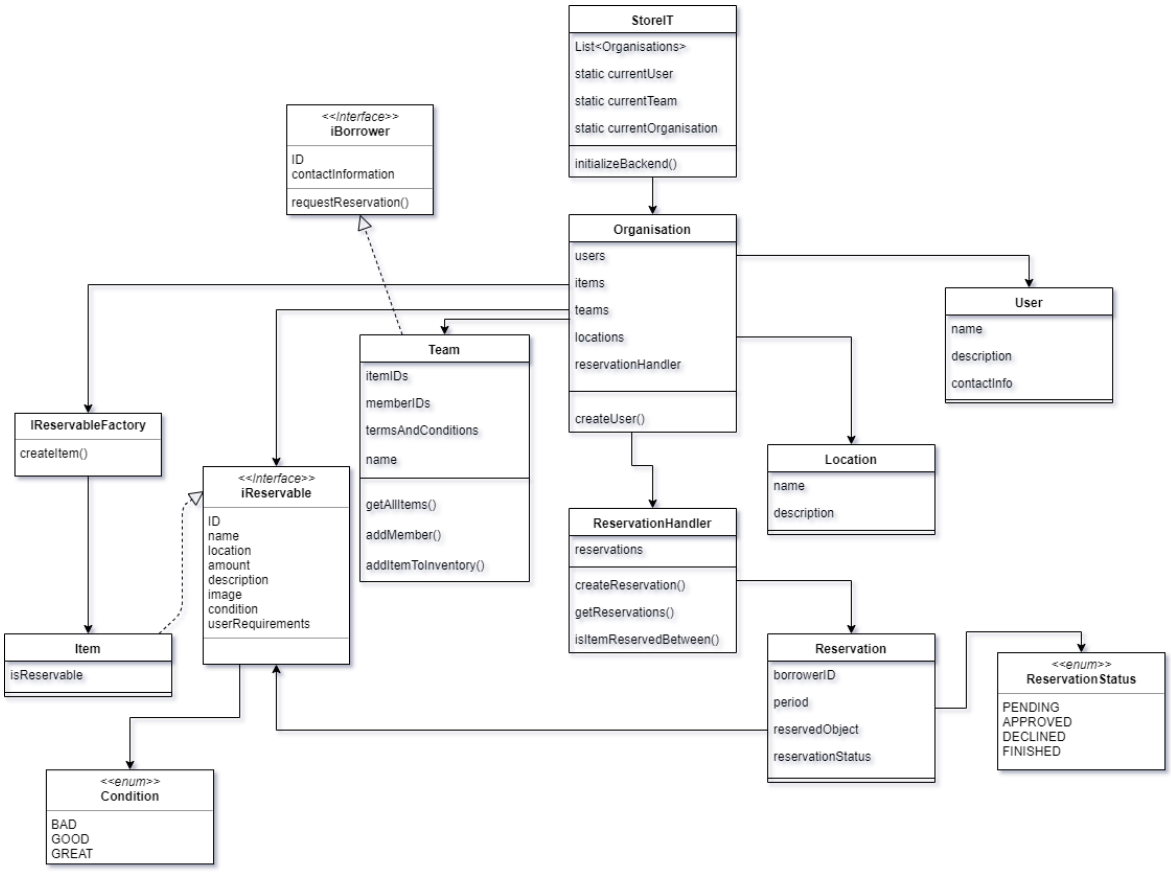
- Main - Huvudklassen som används för att starta programmet. Det är ingen modul men den ligger överst i hierarkin tillsammans med dem andra modulerna.

Se figur 1 nedan hur applikationens moduler är kopplade.

Viewcontroller är den modul som hämtar data för att ladda up vyn som användaren ser. Den har även hand om den data som manipuleras av användaren under körning i kontrollerna och ändrar modellen utefter vad användaren gör. Den har beroende av *modellen* då det är där föremål och metodik finns. Ändringar som görs av användare skickas från *viewcontroller* till modellen. Kontrollerna i *viewcontroller* använder sig även av *services*.
Designmönster som går att hitta på den övre strukturnivån är MVC mönstret. Detta görs genom att separera applikationen till tre olika moduler och ej låta modellen vara beroende av de andra modulerna och låta modellen uppdateras efter kontrollernas ändringar. View och Controller är sammanslagna i programmet för att lättare

kunna separera applikationens funktionalitet och delar. Exempelvis är filer som har med inloggning att göra i samma undermapp i Viewcontroller modulen för att lätt kunna hitta relaterande filer.

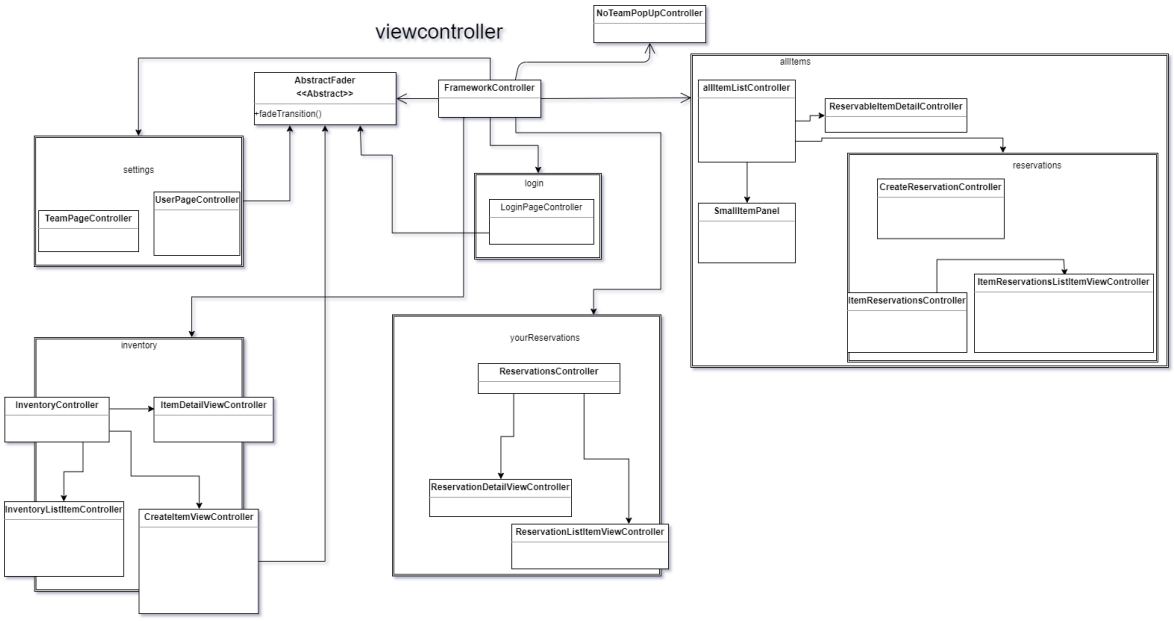
3.1 Modell



Figur 2: StoreIT's designmodell

Skillnaderna mellan vår domän och designmodell är att designmodellen är mer detaljerad och förklarar tydligare de relationer som finns i domänmodellen. Som syns i figur 2 är det organisationen som håller i teams, users och items. Detta är för att minska beroendena mellan de tre klasserna och att istället låta organisation hantera beroendet.
För att skapa lösare beroenden använder vi oss av Dependency inversion principle till viss grad. Där Item klassen implementerar IReservable och Team implementerar IBorrower. Detta möjliggör utökning av det som går att låna ut samt vem det är som får skapa reservation.

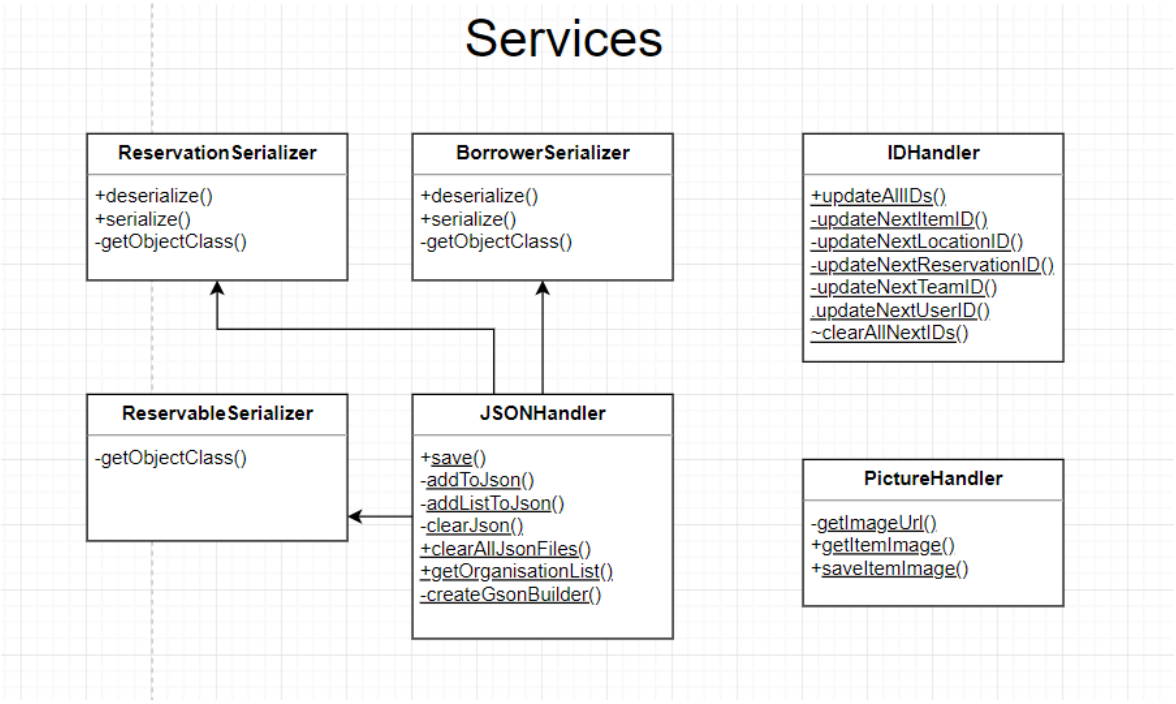
3.2 View-Controller



Figur 3: StoreIT's Views & Controllers

Vi valde att ha vy och kontroll i samma modul eftersom det blir bättre organiserat på så sätt. I javaFX är det vanligt att en .fxml fil kopplas till en separat kontroller, därför bestämde vi oss för att dela in vy och kontroller filer beroende på den funktionalitet de har.

3.3 Services



Figur 4: StoreIT's Services' UML

Services är beroende av modellen främst för att JSON-paketet ligger där och behöver data från modellen för att spara data vid avslutad körning.

3.4 Design Patterns

Nedan följer de designmönster som använts i applikationen:

- **MVC** mönstret hittas i det översta lagret i programmet. Hur MVC implementeras hittas i punkt 3 i Systemarkitektur.
- **Factory pattern** är ett mönster som minskar beroenden då ett item skapas genom att skriva till ett interface istället för att prata med item-klassen. Detta sker när en användare ska skapa ett item i sitt inventarie. Istället för att organisationen skapar ett item så kallar den på IReservableFactory som istället skapar ett item.
- **Adapter** används genom serializer klasserna i `src/main/java/storeit/services`, där vi serialiserar exempelvis IReservable till JsonElement samt deserialiserar från JsonElement tillbaka till IReservable.
- **Module** gör så att applikationen har ett mindre komplex upplägg. Det möjliggör även för att metoder i bland annat team och ReservationViewDetailController kan ha package-private för att undvika onödig exponering.

4 Hantering av kvarvarande data

4.1 JSONHandler

Vårt program sparar data med JSON-filer med hjälp av GSON(1) genom klassen JSONHandler i Services modulen. Vi använder dessa formaten eftersom de är lätt–implementerade och relativt lätta att tolka, läsa igenom och lära sig. JSON-filer sparas ned i `src/main/resources/json` mappen i slutet av varje körning och hämtas därifrån vid initiering av programmet. JSONHandler innehåller ett antal metoder genom vilka man kan skriva till och läsa från den JSON fil som är deklarerad.
För att allt detta ska fungera behövs dock särskilda så kallade serializers och deserializers för de typer som GSON inte kan hantera automatiskt. Exempelvis kan inte GSON automatiskt identifiera vilken typ av item den ska skapa när man använder sig utav interfaces. GSON kastar då ett undantag om GSON inte kan hitta en argumentslös konstruktor i interfacet, en självklarhet då interfaces inte kan innehålla konstruktorer.

```
java.lang.RuntimeException: Unable to invoke no-args constructor for interface
 storagesystem.model.IReservable. Registering an InstanceCreator with Gson for this
  type may fix this problem.

    at com.google.gson.internal.ConstructorConstructor$14.construct
(ConstructorConstructor.java:228)
    at com.google.gson.internal.bind.ReflectiveTypeAdapterFactory$Adapter.read
(ReflectiveTypeAdapterFactory.java:212)
    at com.google.gson.internal.bind.TypeAdapterRuntimeTypeWrapper.read
(TypeAdapterRuntimeTypeWrapper.java:41)
    at com.google.gson.internal.bind.CollectionTypeAdapterFactory$Adapter.read
(CollectionTypeAdapterFactory.java:82)
    at com.google.gson.internal.bind.CollectionTypeAdapterFactory$Adapter.read
(CollectionTypeAdapterFactory.java:61)
    at com.google.gson.internal.bind.ReflectiveTypeAdapterFactory$1.read
```

Figur 5: Undantag utan en särskild adapter för IReservable

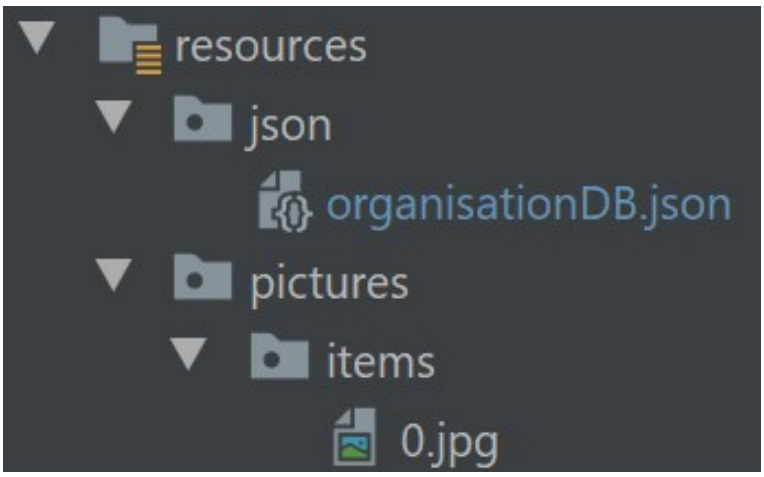
4.2 Organisationer

Informationen som sparas av JSONHandler är en lista av alla Organisations och deras data. Organisation innehåller listor med Items, Teams, Users och Locations samt en ReservationHandler, som i sin tur innehåller en lista med Reservations. I och med detta behöver inte Items, Teams, Users etcetera sparas ner då all denna information redan finns i Organisation. När man hämtar informationen från JSON-filen via JSONHandler returnerar den en lista med Organisation-objekt.

4.3 Bilder

Samtliga bilder som används sparas i `src/main/resources/pictures` mappen. Allmäna bilder som används till applikationen finns direkt i den tidigarenämnda mappen.

Bilder som är specifika till items finns i egna mappar så som `src/main/resources/pictures/items`. Bilderna som tillhör alla Items sparas genom klassen PictureHandler och dess metoder. Varje Item-bild kopplas till respektive Item genom att nämnas till dess itemID.



Figur 6: Resources mappens struktur

5 Kvalitet

5.1 Användarsäkerhet

När man startar programmet möts man direkt av en loginskärm. Har man inget användarkonto kan man där registrera sig. Användaren sparas då till den organisation som är vald i programmet.

När någon försöker logga in så jämför modellen användarnamnet och lösenordet mot alla som finns i den valda organisationen. Finns ingen användare med det lösenordet så nekas inloggning. Vi har valt att inte jämföra användarnamn skiftläge–skänsliga, detta då många är vana vid att inte behöva tänka på det vid inlogg. Användarnamnet sparas dock exakt så som man skriver in det när man registrerar sig eller ändrar på sin användare.

5.2 Tester

Vi har använt JUnit som ramverk när vi har skapat tester vilket har möjliggjort olika varianter av tester. De tester Vi har skrivit har varit på majoriteten metoder i modellen med fokus främst på publika metoder. Det vi valt att inte testa är get-metorder, set-metoder och kontrollerklasser.
Den väg vi gått för att testa kontrollers är att vi själva har gått igenom programmet och testat så att funktionalitet stämmer överens med våra förväntningar. Detta eftersom det är svårt att testa funktionalitet som endast uppstår under körning av programmet.
Testerna går att hitta under: `src/test/java/storeit`.

5.3 Analysverktyg

Förutom de tips man får direkt i IntelliJ så har vi har använt oss utav två analysverktyg. Dessa främst under slutet av projektet för att se till att vi hade tillräckligt med material för att analysen skulle vara värd tiden.

5.3.1 STAN – Structure Analysis for Java

STAN har hjälpt till att visa vilka beroenden men har mellan paket och inom paket, hur många beroenden som finns. Med hjälp av STAN är det enkelt att jämföra så att man följer den designmodell som man har skapat.

5.3.2 PMD – Programming Mistake Detector

PMD används genom Maven-kommandot ”maven compile site”. Den generar en sida (se figur) där vanliga misstag såsom oanvända variabler, tomma catch-block och liknande rapporteras.

5.4 Kända problem

Ingen direkt säkerhet för lösenord. De sparas just nu i plaintext i JSON filen. Detta då tanken är att ifall man vill använda programmet ”på riktigt” i framtiden så får man koppla in sitt egna användarsystem alternativt utveckla ett nytt.(2)

Reservationer blir aldrig klara utan är kvar för evigt. Egentligen ska de sättas till ”FINISHED” då tiden man bokat tagit slut, men detta görs inte.

Just nu finns bara ett visst antal lag, det finns ingen möjlighet att under körning skapa nya lag. Detta är en funktionalitet som borde finnas med, och en knapp i sidomenyn i ramverket som leder till en ny vy för att lösa detta, borde läggas till.

Det går att acceptera flera reservationer med samma item och tid, vilket inte borde gå att göra. Istället borde alla överlappande reservationer avslås när en av dem accepteras.

Just nu går det att ta bort sig själv från ett lag. Detta är en funktionalitet som borde finnas, men inte så som den fungerar just nu. Man borde bland annat få respons på att man håller på att ta bort sig själv och vad som gäller då.

IBorrower är ett interface som aldrig används, vilket den borde göra.

Det kan ske problem med ett exception som inte är kollat, nämnligen ”NoSuchElementException”, vilket verkar som dålig design.

Programmet använder sig flitigt av olika ID:n istället för att använda sig av referensvariabler. Detta kan vara något som förstör poängen med objektorienterad design. Till exempel finns det i en reservation ett item’s ID istället för själva föremålet.

6 External tools

Följande bibliotek har vi importet genom att skapa dependencies i maven:

- JUnit (3)
- joda-time (4)
- gson (1)
- gson-jodatime-serialisers (5)

Dessa har alla valts med omsorg för att de har mycket funktionalitet som krävs, och de är alla välunderhållna.
JUnit är en självklarhet och en välaccepterad standard för att skriva tester till sitt program.
Joda-time är vårat val av hantering av tid. De bibliotek som redan finns i Java har alla problem, och joda-time är rekommenderat av många och uppdateras regelbundet. Utöver det är det något enklare att använda.
GSON är vårt val av bibliotek för att hantera json filer. Det är googles bibliotek, och har flera fördelar. Det är lättimplementerat, lätt att lära sig och ger läsbar kod. Det är även ett av de oftare uppdaterade biblioteken.
Gson-jodattime-serialisers var ett nödvändigt beroende för att kunna omvandla objekt från joda-time till objekt Gson kan tolka och spara i json.

7 Referenser

[1] "Gson." https://github.com/google/gson.

[2] L. Gupta, "Solid principles in java." https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/.

[3] "Junit5." https://junit.org/junit5/.

[4] "Joda-time." https://www.joda.org/joda-time/.

[5] "gson-jodatime-serialisers." https://github.com/gkopff/gson-jodatime-serialisers.

Peer Review of bYMe

Pär Aronsson William Albertsson
Jonathan Eksberg Hugo Stegrell Carl Lindh

25 oktober 2019

Domänmodell

- "Bild" och hela raden längst nere känns som de inte är relevanta i domänmodellen, utan är en del av andra objekt. Detta blir tydligt eftersom de inte har några klasser i designmodellen eller implementationen
- Vad är "Som köpare/Säljare" uppe till vänster? Inte helt tydligt, och pilen hjälper inte
- Användare kan ha 0 bilder i applikationen, men inte i domänmodellen.

Designmodell

- Omöjligt att följa vissa pilar som överlappar

Saknar Javadoc helt, så kan knappt kommentera dokumentationen. Blir svårt att följa vissa metoder och klassers ansvar/syften. Finns några kommentarer i koden men de är single-line, de hjälper en att följa med bättre i metoder men de borde stå som javadoc istället.

Vad är det för lista till vänster på startskärmen? Sökrutan söker när man skriver, borde söka efter man har skrivit. Sökrutan söker också "ett tecken för sent".

Det var några år sedan Googles JSON simple uppdaterades, kan va värt att ha koll på ifall några problem skulle dyka upp! Finns en annan version som hålls uppdaterad här.

Ett test misslyckas. Detta borde kollas innan något pushas till master (förutsatt att det var master vi fick) för att säkerställa att allt där är fungerande och att tidigare kod är helt fungerande.

generateRandomAdId() kollar aldrig efter om ID:t redan finns, vilket leder till att olika Ads kan ha samma ID. Den genererar dessutom bara 4 siffror, alltså hade programmet garanterat haft två Ads med samma ID om man hade fler än 1 000 ads.

generateRandomAdID() metoden måste inte användas för just en Ad. Funktionen skulle istället kunna heta generateFourRandomNumbers() och sedan bara returnera det. Känns även som returtypen borde vara int och inte String för att man inte ska göra några antaganden om hur den ska användas.

"Byme" klassen sköter mycket funktionalitet. Både för exempelvis ads och inloggning. Detta borde brytas upp i flera klasser för att följa Single Responsibility Principle. Namnet på klassen har dessutom ingenting med de sakerna att göra.

Borde printa i programmet när t.ex. ett konto med det namnet redan finns, inte i terminalen som det görs i nuläget.

Vi behöver inte instanser av Search, utan den kan vara statisk. Allt den gör är att söka bland ads, kanske kan vara en del av Ad till och med? Om bYMe delas upp som tidigare föreslagits kanske det finns en ny klass som naturligt hanterar Ads och sökning bland dessa.

Ni har valt att slå ihop vy och kontroller vilket är rimligt med tanke på användandet av javafx, men alla fxml.filer borde läggas i nåt annat än /main/java där de ligger nu. Modellen verkar bra separerat från view-controller.

profile_pics och adPictures kan läggas i en mapp, och ges konsekvent namngivning.

AccountHandler.parseAccount(..) skriver ut i terminalen när något går fel, bör använda relevant felhantering. Sker på flertal ställen i programmet.

Formatering är enkelt fixat och gör det lättare att läsa, är inte helt rätt överallt. Går att högerklicka på src mappen och sen "Reformat code" så fixar IntelliJ det.

Det är ganska få tester, men eftersom de testar större delen av modellen är det inget stort problem.

I bYMe finns en Hashmap<String, Account> där strängen är usernamet som är samma som i account. Vi förstår inte riktigt varför en hashmap behövs och det känns som duplicering av data är onödig.

Det går att skriva bokstäver i pris för en Ad men sedan går det inte att spara. Någon respons borde ges till användaren.