Final Report

Johan Almroth, Albin Otterhäll, David Persson,
Gustav Pihlquist, Morgan Thowsen

2019-10-25
Version 0.1

Contents

List of Figures

1 System design document

1.1 Introduction

Det här dokumentet beskriver projektets arkitektur, fil- och databashantering, externa beroenden, tester, kända fel och säkerhet.

Vår applikation siktar på att ersätta kontanthanteringen till en kiosk som tillhör studenterna från det Datavetenskapliga programmet på Göteborgs Universitet. I applikationen finns istället en virtuell valuta, som kan användas för att köpa produkter i en kiosk, t.ex. snacks, dricka, frukt etc. Det är viktigt för oss att applikationen är utökningsbar då det kan finnas många potentiella användningsområden i framtiden. Till exempel reklam, kommande evenemang och försäljning av kurslitteratur.

1.1.1 Definitions, acronyms, and abbreviations

- **Immutable** - Ett objekt vars tillstånd inte kan ändras efter att det skapats.
- **Mutate-by-copy** - Istället för att ändra ett Immutable objekt så skapas en kopia med andra värden.
- **Fragment** - En dynamiskt föränderlig del av en aktivitet
- **Activity** - En aktivitet där t ex, knappar, text med mera kan visas. Det kan vara ett aggregat för flera Fragments, se ovan.
- **Debug** - Felsökning
- **Bug** - Allvarlig fel i programmet som påverkar prestandan avsevärt och/eller orsakar krascher.
- **Mock** - Som i Mock User, skenanvändare. En temporär immitation under utvecklingsfasen.
- **Code smell / illaluktande kod** - Mindre bra implementerad kod som gör det den ska men följer inte goda designprinciper.

1.2 System architecture

Arkitekturen i vårt projekt består av en tolkad MVC samt module-pattern. Vår tolkning är att Model är helt separerad från resterande delar av applikationen, controller motsvaras av fragment, activities med mera som gör anrop på modellen. View motsvaras av XML-filer under paketet `res/layout/` och `res/menu/`.

Modulepattern är använt för att strukturera och tydliggöra beroenden mellan moduler.

När applikationen startas kommer användaren till inloggningsskärmen. Förutsatt att användaren skriver in korrekt användernamn och lösenord startas huvudaktiviteten. I huvudaktiviteten initieras modellen, databas och alla fragment. Modellen startas utan databas men blir tilldelad en lista av produkter från databasen efter initieringen. Denna kommunikation sker via ett interface.

Grundtanken är att modellen skall vara helt oberoende. Databaspaketet har ett beroende på modellen. Controllerpaketet har beroende på stora delar av modellen.

Då vår tolkning för Android är att XML-layout filer samt XML-menu filer motsvarar view-komponenter har de alltså inga beroenden på modellen.

Ansvarsområden:

- Dialogpaketets enda ansvarsområde är att presentera en dialog där skapandet har ett så litet interface som möjligt. Utan externa beroenden bortsett från Android-Dialog.
- Model - logik för applikationen. Får inte vara beroende på något utomstående.
- Databas - paketet har beroende på modellen men inget annat. Paketet är strukturerat så att den faktiska databasimplementationen (mock för tillfället) inte har något beroende på modellen utan endast kommunicerar via ett interface som i sin tur delegerar konstruktionen av objekt till en deserialiserare och lämnar resultatet till modellen. Även den sista överlämningen sker i form av ett interface.
- AuthenticationController ansvarsområde är att koppla ihop delarna authentication-paketet i modellen med vyn till användaren. AuthenticationController-paketet har även en koppling till huvudaktiviteten då den startar huvudskärmen vid korrekt angivna inloggningsuppgifter.
- MainController ansvarar för hela huvudvyn med alla dynamiska fragment som byts ut när användaren klickar sig runt. Den ansvarar även för initialiseringen av modellen samt att lägga in de produkter i modellen som skall visas.

För visualisering se figurer i **List of Figures**:

- För modell UML, se figur 14
- För Controller UML, se figur 15
- För Database UML, se figur 16
- För Domänmodell, se figur 13

1.3 System design

1.3.1 Design Patterns

Vi har implementerat följande designmönster i vårt projekt:

- **Immutable Pattern**
  Produkt och User är immutable. I fallet då vi vill ändra en användares credits använder vi istället mutate-by-copy. Detta för att enkelt kunna felsöka koden.

- **Factory Pattern**
  Vi använder oss av en så kallad Factory när vi vill skapa nya produkter, Dialoger mm. Detta för att kunna helt dölja den interna implementationen bakom ett interface där vi kan begränsa möjliga metodanrop.

  I fallet med Dialog så får vi även fler möjligheter vid skapandet då vi har möjlighet att ändra saker utanför konstruktorn vid skapandet av dialogen.

- **Module Pattern**
  Projektet är uppdelat i tydliga moduler, där alla klasser som tillhör modellen ligger under paketet "Model" och alla fragments/activities ligger under "Controller". Under modellmappen ligger t.ex. även Cart, Product och User i egna paket med tillhörande klasser. Med module pattern tycker vi inte bara det blir visuellt tydligare, det blir också mer intuitivt och tydliggöra eventuella externa beroenden från en modul.

- **Strategy Pattern**
  Strategy Pattern tillämpas i sorteringsmetoden. Det finns fyra olika sätt att sortera på - namn i stigande ordning, namn i fallande ordning, pris i stigande ordning och pris i fallande ordning. Modellen tillhandahåller ett antalet "strategier" men det möjliggör att vid ett senare tillfälle skulle kunna sortera på attribut som inte redan finns idag, t ex klimatpåverkan.

- **Singleton Pattern** Används i t ex DatabaseHelper för att försäkra oss om att det endast finns en instans av databasen. Detta då det inte finns någon anledning till att ha flera instanser.

1.4 Persistent data management

För tillfället tillhandahåller applikationen själv en ”mock” databas med produkter. Denna går inte att skriva till. Även de bilderna som visas i applikationen är idag hårdkodade i projektet.

Databaspaketet består huvudsakligen av 3 entiteter, en Deserializer, en Databas-handler samt en faktiskt databasimplementation. All kommunikation till Databasen sker genom Databas-handler. Denna klass tar emot en Json-string från den konkreta databasimplementationen och använder sig av en deserializer som utgör kopplingen till objektet som skall konstrueras.

Tanken är att deserializern enkelt skall kunna bytas om en annan typ av objekt skall deserialiseras. Även den konkreta implementationen av databasen skall enkelt kunna bytas ut då kommunikationen sker genom ett interface.

Våra ambitioner om denna produktdatabas är följande:

- Uppdatera/ändra i produktlistan och lagersaldot
- Ta emot användares rekommendationer på nya inköp till kiosken

Det skall tilläggas att vi räknar med att behöva en separat databas för hantering av användarregister och deras krediter.

1.5 Quality

1.5.1 Testing

Vi har försökt i största mån att testa metoder enskilt utan yttre påverkan med hjälp av **JUnit** . För att just enskilt testa dem används ett externt bibliotek, **Mockito**. Med detta bibliotek kan vi hårdkoda vad som skall returneras från externa objekt och metodanrop. Ett exempel på detta är att vi vet vad en produkt skall returnera när den frågas efter pris. För att inte bli beroende på denna funktion när vi testar produktsortering så har vi alltså använt mockito för att utesluta denna yttre påverkan.

Vi har i största möjliga mån försökt testa metoder i modellen med flera tester per metod.

Projektet använder Travis Continous Integeration som hittas på följande länk.
`https://travis-ci.org/Ackaman/KoskenKiosken-DIT212/builds`

1.5.2 Known issues

Bugs

- Man kan köpa 0 varor för 0 kronor.
- Man får en varning att lösenord behöver vara mindre än 5 tecken trots att det är mer.
- Den användaren som loggar in har ingen koppling till den användaren vars information presenteras i AccountFragment.

Delvis implementerat / arbete pågår

- Modellen använder fortfarande en falsk användare istället för en faktiskt användare.
- Aktivitetsfältet längst upp på telefonen är grönt trots att vårt färgschema i övrigt är blått.
- Tangentbordet är fortfarande synligt efter att man lämnat "Store" fliken medans sök-inmatningen var aktiv.

Code smell

- Onödigt beroende av IProduct i DatabaseHandler. DatabaseHandler erhåller en deserializer vilken beror på IProduct. Följden blir att DatabaseHandler egentligen inte behöver känna till vilket objekt som skapats. Att ordna detta visade sig ta för lång tid och lämnades därmed i nuvarande skick.
- Många onödiga beroende i Fragmenten. Fragmenten får en refernes till modellen istället för ett anpassat interface.
- En Cart-Factory har inte implementerats ännu vilket skapar onödiga beroende till modellen. Följden blir att modellen alltså är beroende av båda Cart och ICart.
- Modellen bör inte hålla i en lista med produkter som utgör inventarie. Detta bör vara ett separat paket som likt Cart inte är beroende på produktpaketet. Med detta följer att sortering och filtreringsmetoder bör flyttas till denna nya klass.
- Köp, bör inte behandlas av User. Detta är en temporär lösning. Även implementationen i Modell bör vara en delegering till extern payment-handler likt hur IDatabase relationen till modellen är.
- Hårdkodad användarkonto för autentisering. Variabeln med referensen till objektet är publik.
- Flera av användarklasserna som används för autentiseringssystemet kan vid en refaktorisering ersättas med redan existerande klasser.
- Metoden `onCreate` i `LoginActivity` klassen är väldigt stor.
- Autentiseringssystemet använder inte samma kodstil som övriga delar av applikationen. Till exempel används parametersierade funktioner i flera delar.
- Klassen `Result` kan refaktoriseras för att peka på exceptions och relevanta objekt.

1.5.3 STAN

Vi har analyserat strukturen av vår kod med verktyget STAN. Verktyget hittade inga cirkulära beroenden, se figur 1

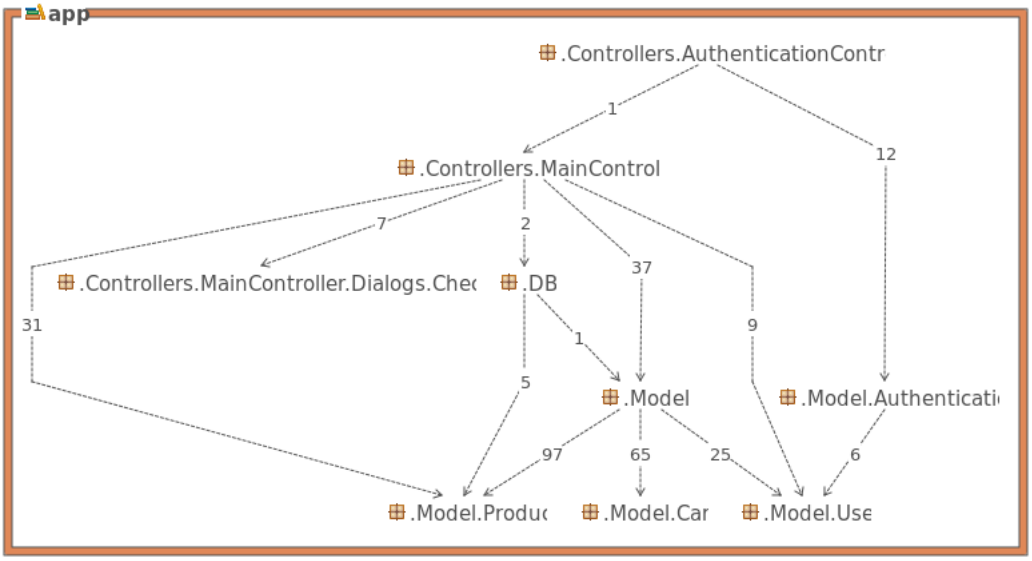

Figure 1: Stan Analysis

1.5.4 PMD

Vi hänvisar till filen *pmdoutput.html*.

1.5.5 Access control and security

Autentiseringssystemet ansvarar för att användare ska kunna identifiera sig för att kunna få tillgång till användarens tillgångar. I nuvarande form är systemet mycket osäkert då den ända användarens användarnamn och lösenord finns sparat i klartext i applikationen. Det ända som behövs göras är att reverse engineera binären för att få tillgång till inloggningsuppgifterna.

Nu kan man logga in med användarnamnet `firstuser` och lösenordet `password`. Om man skriver fel inloggningsuppgifter får man ett felmeddelande. Med rätt inloggningsuppgifter kommer man vidare till affären.

2 Requirements and Analysis Document

2.1 Introduction

Vår applikation siktar på att ersätta kontanthanteringen till en kiosk som tillhör studenterna från det Datavetenskapliga programmet på Göteborgs Universitet. I applikationen finns istället en virtuell valuta, som då kan användas till att betala för utbudet i kiosken, t.ex. snacks, dricka, frukt etc. Det är viktigt för oss att applikationen är utbyggbar då det finns många potentiella användningsområden. Till exempel reklam, marknadsplats för kurslitteratur, kommande evenemang mm..
De som är ansvariga över kiosken kan genom att använda denna appen undvika dyra betalningsmetoder, samt se användbar statistik såsom transaktionshistorik. Idén bygger på att en eller flera administratörer ska kunna mata in ett definierat antal av varje produkt i en databas(lagersaldo). När transaktioner utförs uppdateras lagersaldot automatisk. Till skillnad mot en kontant kassa kan med hjälp av appen även spåra eventuellt svinn och stölder.

Appen är i nuläget inte planerad att släppas på Google Playstore och kommer istället distribueras till studenterna inom föreningen.

2.1.1 Definitions, acronyms, and abbreviations

- master - mästargrenen ”master branch” på github.
- dev - utvecklingsgrenen ”dev branch” på github.
- fragment - del av en större vy. Dessa kan bytas mellan utan att byta vy.
- vy - det som visas på skärmen.
- feature - funktion
- interface - funktionsgränssnitt
- activity - Ett koncept i Android ekosystemet för en sida med vyer. Dessa bestämmer när och vad som ritar upp på skärmen. En av grunderna som applikationer för Android använder.

2.2 Requirements

2.2.1 Completed User Stories

User Story 1

Story Identifier: 1
Story Name: Ändra antalet produkter i kundvagn.

Description

Som en användare, vill jag kunna ändra antal produkter i kundvagnen innan check-out så att jag kan rätta felvalda produkter.

Confirmation

Functional

- koppla knapparna till antalet av produkten i varukorgen
- en knapptryckning motsvarar 1 inkrementering/dekrementering

Non-functional

- implementera upp knapp för att inkrementera antalet av varan.
- implementera ned knapp för att dekrementera antalet av varan.
- Om användaren dekrementerar antalet till 0 skall produkten försvinna.

User Story 4

Story Identifier: 4
Story Name: Söka bland produkter i affären.

Description

Som användare, vill jag kunna söka bland produkter för att snabbt hitta vad jag letar efter.

Confirmation

Functional

- Expandera en textruta där du skriver in produktnamnet du söker
- sökfältet skall inte förkomma i vyer utanför storeview.
- Vyn ska ändras av sökningen
- Vyn skall filtreras enligt sökning.
- När du tar bort söksträngen skall alla produkter visas igen
- Endast produkter som matchar strängen skall visas

Non-functional

- Implementera en sökknapp i storeview.
- Visa sökträffarna i samma vy, fast filtrera bort/göm produkterna som inte matchar sökresultatet.

User Story 5

Story Identifier: 5
Story Name: Hello world.

Description

Som en ägare vill jag se en startskärm ”Hello world” för att se en första prototyp så jag vet att utvecklingen är igång.

Confirmation

Functional

- skall kompilera och vara en körbar app.
- TravisCI skall säga ifrån när tester inte passar
- TravisCI skall ge grönt ljus när alla tester passar

Non-functional

- inga warnings skall förekomma i IDEn.
- TravisCI skall köras vid pull-request push.

User Story 6

Story Identifier: 6
Story Name: Lista av produkter i kiosk.

Description

Som en användare vill jag kunna se vilka produkter som finns i kiosken, så jag får en överblick över vad som finns.

Confirmation

Functional

- Listan visar alla produkter som finns i kiosken, om så bara en hårdkodad.
- Listan skall gå att scrolla om den är för stor.
- produkterna i listan skall inte bli mindre för att alla skall få plats.

Non-functional

- Bör framgå att detta är en huvudvy. (då det är en kiosk).
- Det ska vara tydligt vilken information som hör till vilken produkt.

User Story 7

Story Identifier: 7
Story Name: Produktrepresentation i flödet enl. android standard.

Description

Som en användare vill jag kunna se en representation av en produkt i form av ett ”card”

Confirmation

Functional

- namnet skall representeras.
- priset skall representeras.

Non-functional

- Samtliga fält Måste synas tydligt.
- fälten skall framgå tydligt vad de avser.

User Story 8

Story Identifier: 8
Story Name: Klicka på produkt för mer information.

Description

Som en användare vill jag kunna titta på en produkt för att få mer information om den så jag kan få mer information.

Confirmation

Functional

- Det ska gå att trycka på produkten för att öppna en ny expanderad vy för att få yttligare information.
- det skall gå att gå tillbaka från denna vy.

Non-functional

- Fyll vyn med relevant information.
- Det skall finnas produktbeskrivning.
- Det skall finnas produktnamn.
- Det skall finnas pris på produkten.

User Story 10

Story Identifier: 10
Story Name: Profilsida.

Description

Som en användare vill jag kunna se min profilsida med information om credits och användaruppgifter, för att få en överblick om hur mitt konto ser ut.

Confirmation

Functional

- Aktuellt användarnamn, kreditbalans visas på profilsidan.

Non-functional

- Det skall finnas en knapp för att ta sig till kundvagnen
- ovan knapp skall inte ligga i en undermeny. (lätt åtkomlig)

User Story 11

Story Identifier: 11
Story Name: Betalning.

Description

Som en användare vill jag kunna betala för valda produkter i varukorgen, så att jag kan genomföra köpet av produkterna.

Confirmation

Functional

- Dra credits från användaren om köpet genomfördes
- Credits skall aldrig dras om köpet inte genomförs
- Köpet skall inte genomföras om användaren inte har tillräckligt med credits
- Vi betalning dras motsvarande credits från inloggad användare.

Non-functional

- det skall finnas en köpknapp.
- Skapa en visuell bekräftelse om köpet genomfördes.
- Skicka ett felmeddelande om köpet inte kunde genomföras.

User Story 12

Story Identifier: 12
Story Name: Varor i varukorgen.

Description

Som en användare vill jag kunna se vilka varor jag har i min varukorg, så att jag ser vad jag har valt.

Confirmation

Functional

- När inga items valt ska varukorgen vara tom.
- Om appen kraschar eller att man tvångsavslutar den så skall varukorgen tömmas

Non-functional

- Varukorgen ska finnas tillgänglig i UI:t.
- Varukorgen ska lista alla produkter som lagts till under sessionen.

User Story 14

Story Identifier: 14
Story Name: Lägga till produkter i varukorgen.

Description

Som en användare vill jag kunna lägga en produkt i kundvagnen, för att kunna lista produkterna jag skall köpa och inte begränsas till en vara åt gången.

Confirmation

Functional

- När man trycker på ikonen läggs produkten i kundvagnen.
- Lägg till produkt i listan för kundvagn när man trycker på ”+” eller motsvarande.

Non-functional

- Det skall framgå tydligt på produkten hur den läggs i varukorgen.
- Det skall vara lätt att hitta kundenvagnen. (bör inte ligga i någon undermeny)
- Det finns en knapp på produktens kort med en ikon av en kundvagn.

User Story 15

Story Identifier: 15
Story Name: Sortera produkter i affären.

Description

Som användare, vill jag kunna sortera produkter efter pris, namn etc. För att t ex snabbt kunna hitta en billig dricka.

Confirmation

Functional

- När sorteringsknappen har använts så ska listan sorteras efter vad användaren valde att sortera (Pris, A-Z, etc..)
- När knappen för att ta bort filtreringen används så ska listan återgå till standardläget.

Non-functional

- det skall framgå av kontexten att sorteringen hör till varorna.
- valet av sortering bör ske på samma skärm som varorna presenteras.
- Samtliga fält Måste synas.

User Story 16

Story Identifier: 16
Story Name: Kundvagnsbubblan

Description

Som en användare, vill jag se redan på kundvagnsknappen hur många produkter jag valt så att jag kan hålla räkningen.

Confirmation

Functional

- Bubblan ska uppdateras live efter hur många varor som befinner sig i varukorgen
- När kundvagnen inte har några produkter skall ingen bubbla visas

Non-functional

- Implementera en bubbla på kundvagnsknappen
- det bör vara tydligt att bubblan avser antalet produkter i just kundvagnen.
- Bubblan bör alltid vara synlig från alla huvudvyer om det finns produkter i kundvagnen.

User Story 24

Story Identifier: 24
Story Name: Produktrekommendationer

Description

Som användare vill jag kunna rekommendera produkter att köpa in så att jag kan vara med och påverka framtida inköp av produkter.

Confirmation

Functional

- textfält för rekommendation skall ha en begränsning på antalet tecken.

Non-functional

- Användaren skall få bekräftelse på att rekommendationen har skickats
- Produktrekommendationsvyn skall vara lätt att hitta.

User Story 36

Story Identifier: 36
Story Name: Inloggningssystem

Description

Som en användare vill jag kunna logga in i appen, för att få tillgång till funktionerna när man är inloggad.

Confirmation

Functional

- Skapa standard login activity
- Starta login activityn vid uppstart
- Fråga efter användarnamn vid inloggning
- Skickas till shoppen efter inloggning
- Lägg till lösenord till användarkonton
- Kontrollera att det finns ett användarkonto med det angivna lösenordet
- Om korrekt inloggningsuppgifter är angivna: visa shoppen för användaren
  Om inkorrekt inloggningsuppgifter är angivna: visa felmeddelande till användaren

Non-functional

- När man startar appen visas ett inloggningsformulär med fält för användarnamn och lösenord

2.2.2 Non-completed User Stories

User Story 2

Story Identifier: 2
Story Name: Ta bort valda varor från varukorgen

Description

Som en användare vill jag kunna ta bort valda varor från varukorgen, för att jag inte längre vill köpa produkten.

Confirmation

Functional

- vid knapptryck skall produkten försvinna från varukorgen.
- När knappen trycks på skall produkten oavsett antal försvinna från varukorgen.

Non-functional

- det skall finnas en ”papperskorgsknapp” som är knuten till respektive produkt.
- Knappen skall vara väl synlig.
- Knappen skall vara knuten till en specifik produkt.

User Story 3

Story Identifier: 3
Story Name: Se lagersaldo för produkter

Description

Som en användare vill jag kunna se lagersaldo, så att jag vet om varan finns i kiosken.

Confirmation

Functional

- Skicka en request till en databas för att få det aktuella lagersaldot
- Läs in svaret som vi får från databasen

Non-functional

- Visa det aktuella lagersaldot i produktkortet från databasen
- Lagersaldot för en produkt i appen ska överensstämma med antalet som finns fysiskt i kiosken.

User Story 13

Story Identifier: 13
Story Name: Skapa konto

Description

Som en användare, vill jag kunna skapa ett nytt användarekonto för att kunna handla i shoppen

Confirmation

Functional

- det ska finnas en autentiseringsfunktion som godkänner dina inloggning-suppgifter
- Angivna uppgifter skall sparas mot databasen.
- Godkända användarnamn får bara lov att vara engelska bokstäver.
- Godkända lösenord innehåller endast engelska bokstäver, siffror och tecken.
- det skall inte gå att skapa en användare som redan finns.
- det skall inte gå att skapa en användare med ”för få” tecken i namnet.

Non-functional

- Lägg till en ”Registrera ny användare”-knapp
- Lägg till en textruta för användarnamn
- Lägg till en textruta för lösenord
- Lägg till en ”Logga in” knapp
- Vid start av appen i utloggat läge ska det finnas en knapp med texten ”Registrera nytt konto”
- När man trycker på ”Registrera nytt konto” ska ett formulär visas med textfält för användarnamn och lösenord.
- När användaren har fyllt i registreringsformuläret och tryckt på ”Skapa konto” ska man hamna på inloggningssidan.

2.2.3 Definition of Done

Innan varje userstory får anses vara klar skall samtliga föregående tester köras. Detta görs även automatiskt via **Travis CI** men uppmuntras även att göras individuellt innan. För de metoder och ändringar som gjorts i modellen skall samtliga testas mha. **JUnit**. För de androidklasser och vyer som ändras och/el. skapas har vi dessvärre inga ordentliga tester. Var person ansvarar för att de testas och hittas ett problem så lyfts detta vid mötet och delegeras som uppgift att ordna. Det skall tilläggas att alla nya metoder och klasser oavsett plats skall vara välkommenterade för att anses färdiga.

Var user story som påbörjas skapas i en ny branch och påverkar därmed inte det övriga arbetet. När user storyns acceptance criteras är uppfyllda och ovan stycke är uppfyllt så skall den via pull request in i dev. Under detta steg körs Travis CI ytterligare en gång för att säkerställa att alla tidigare tester fortfarande håller.

Varje user story påbörjas skapas i en ny s.k. branch för den valda user storyn. När branchen har uppnåt de efterfrågade features och acceptance criterias för den aktiva user storyn testar vi metoderna med JUnit innan den mergeas till dev-branchen. Innan mergen är godkänd behöver den gå igenom ett andra test, Travis CI, som simulerar en androidenhet på en enskild dator som vidare bekräftar att koden fungerar.

Efter varje iteration/sprint (vanligtvis en vecka lång) merge:ar vi från dev till master. Dev kan innehålla mindre buggar, men inte allvarliga nog för att applikationen skall krascha. Vi tillåter inga kända buggar i Master. Mergeprocessen diskuteras och inventeras av alla i gruppen för att försäkra att inga större buggar hamnar i master.

En pull request till master kan inte accepteras av samma person som skapade förfrågan.

En allvarlig bug är något som orsaker följande

- Programmet kompilerar inte.
- Modellen fungerar inte som avsett och resulterar i en krasch.
- Specialfall som resulterar i ett oväntat eller oönskat beteende under exekvering.

- Oväntat undantag så som att pausa applikationen och rotera från porträttläge till landskapsläge ändrar eller korrupterar modellen.
- Android-specifika inställningar så som upplösnings skalning påverkar funktionalitet/beteende.

2.2.4 User Interface

Vid uppstart av applikationen möts användaren av en inloggningsskärm. För att få tillgång till övriga funktioner i applikationen krävs det att användaren loggar in med sina uppgifter, se figur 2.

Skriver man in felaktiga tecken i användarnamnet får man ett felmeddelande "Ogiltigt användarnamn", se figur 4. Likaså får man ett felmeddelande vid felaktigt inmatat lösenord, "Lösenord måste vara >5 tecken långt", se figur 5.

Vid misslyckat inloggningsförösk får användaren ett felmeddelande "Inloggning misslyckades" och hen kan göra fler försök att logga in, se figur 6.

Navigering mellan sektioner i applikationen görs med hjälp av flikar i botten. Markeringen av den aktuella sektionen säger till användaren vart han eller hon befinner sig i applikationen. Kontext specifika kontroller, ie. kontroller som varierar beroende på sektionen, är inte inkluderade i verktygsfältet. Dessa nämnda kontroller vistas i toppen av applikationen som visat i "store", se figur 8. Detta är inte bara intuitivt för en Android användare men följer riktlinjerna för Android plattformen.

De i ovan kontext nämna funktionerna är sortering av produkter samt sökfunktion. Filterfunktionen presenterar användaren med en dialogruta som frågar vilken typ av sortering som efterfrågas, se figur 7. Dessa dialogrutor används för att användaren inte skall förlora känslan för i vilken kontext sortering skall ske. En motsats till detta vore alltså att användaren sköter detta via en separat vy, vilket inte kan anses optimalt. Denna typ av dialogruta förekommer även när användaren önskar köpa produkterna i varukorgenm se figur 8.

Feedback till användaren vid operationer sker med hjälp av så kallade toasts, det vill säga. korta notiser. Exempel på när sådana dyker upp är t ex när användaren loggar in, köper varor eller inte har tillräckligt med credits för att genomföra ett köp, se nedre delen av figur 6

Navigationen i applikationen sker enligt figur 12, där målet med applikationen var att ha så platt struktur som möjligt. Alla vyer är tillgängliga genom högst två

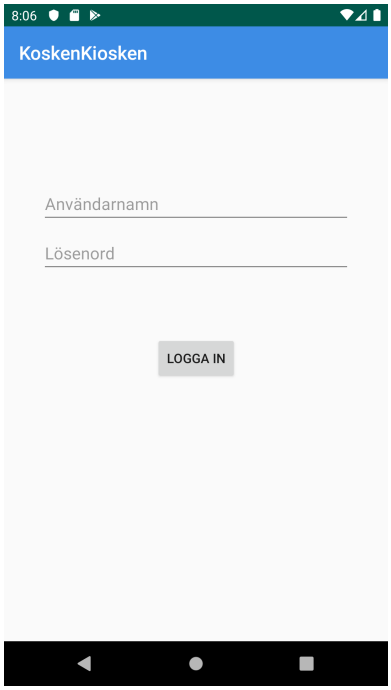

Figure 2: Login-Screen without info

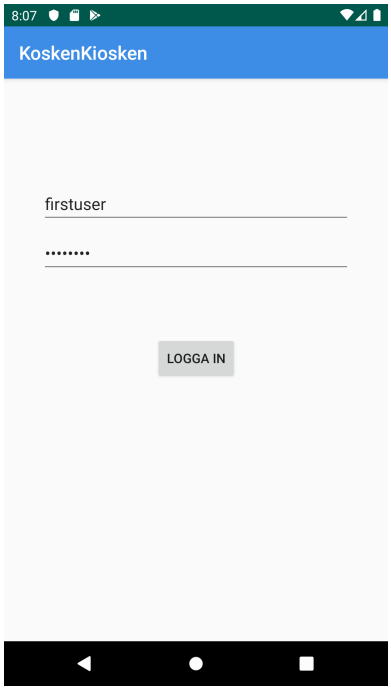

Figure 3: Login-Screen with info

knapptryckningar.

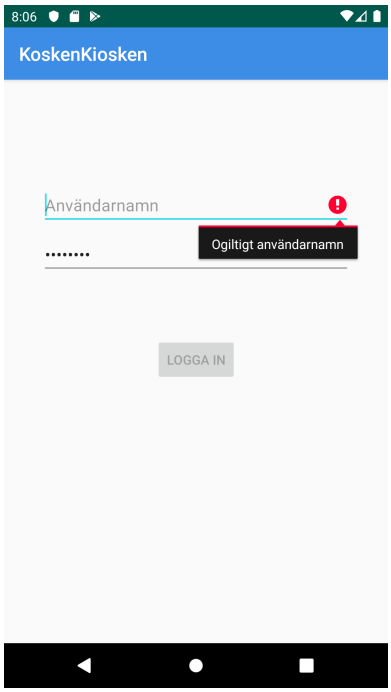

Figure 4: Login-Screen wrong username

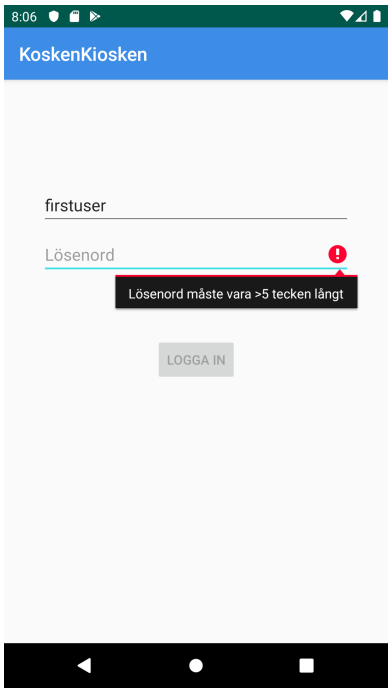

Figure 5: Login-Screen wrong password

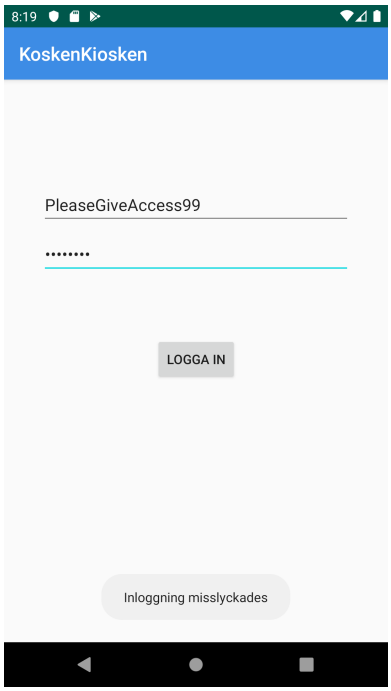

Figure 6: Login failed

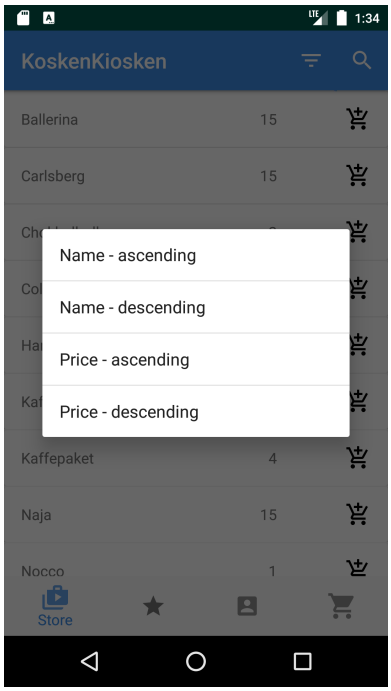

Figure 7: Store sorting

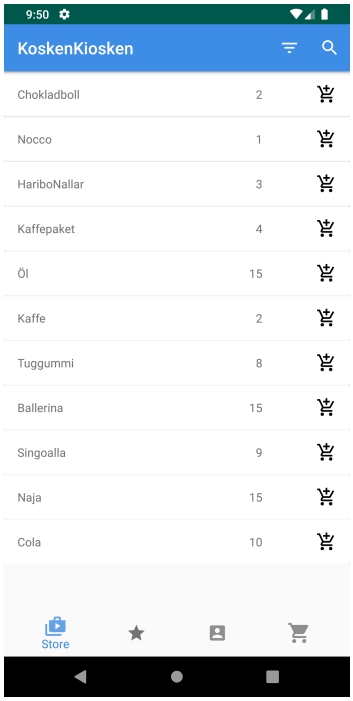

Figure 8: Shop-Fragment

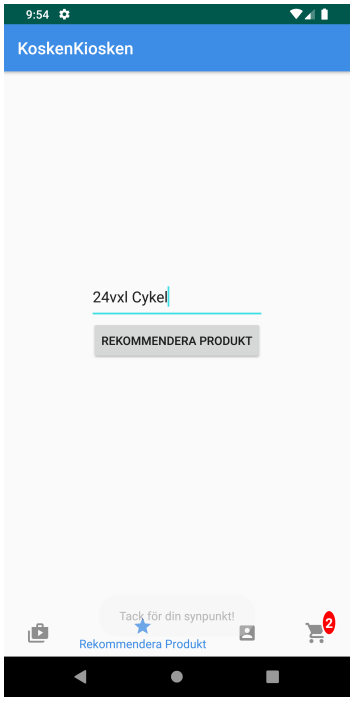

Figure 9: Recommendation-Fragment

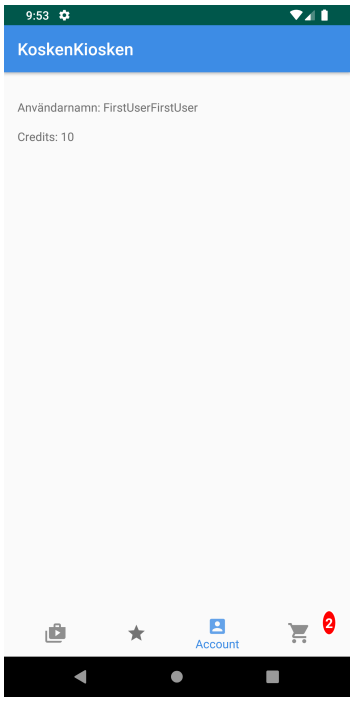

Figure 10: User-Fragment

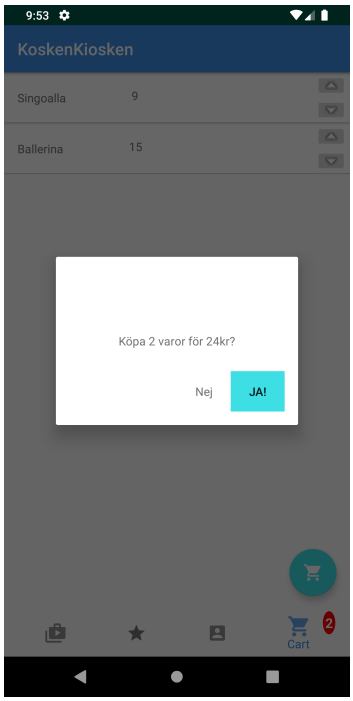

Figure 11: Cart-Fragment with checkout pop up

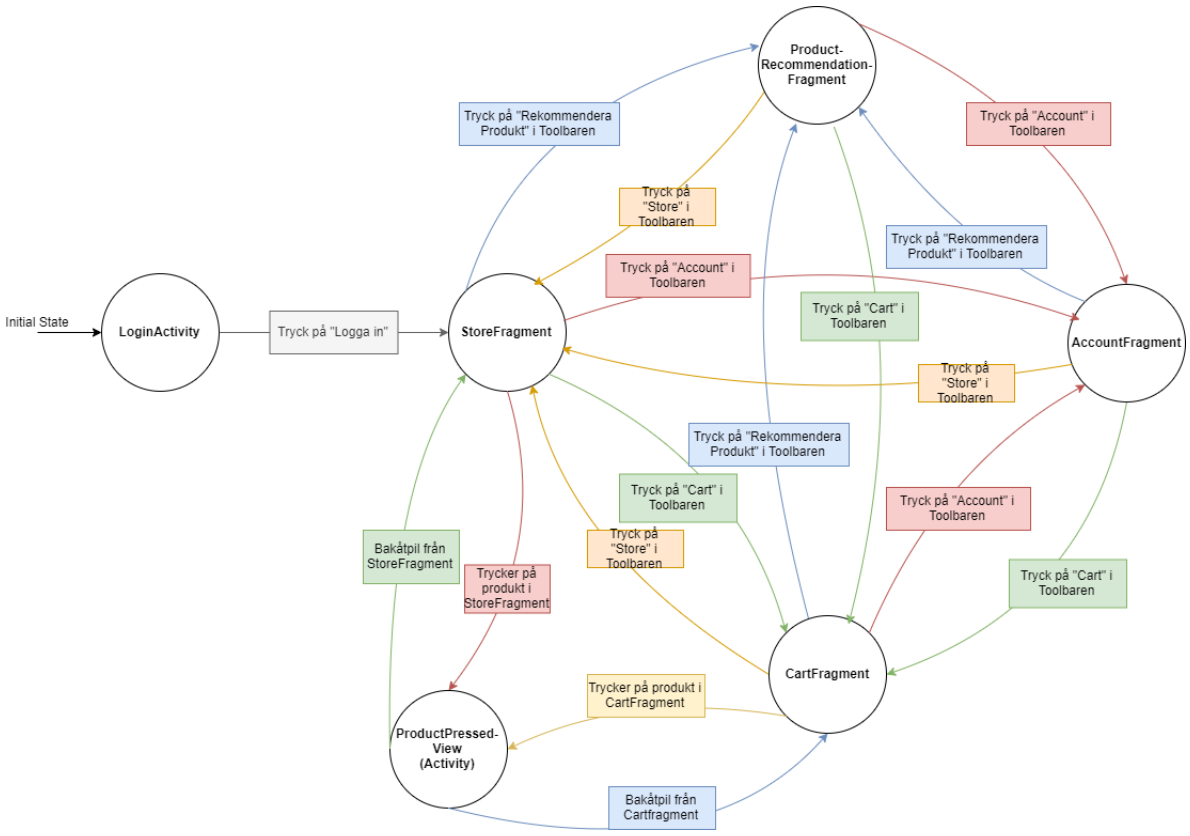

Figure 12: Navigation flowchart

2.3 Domain model

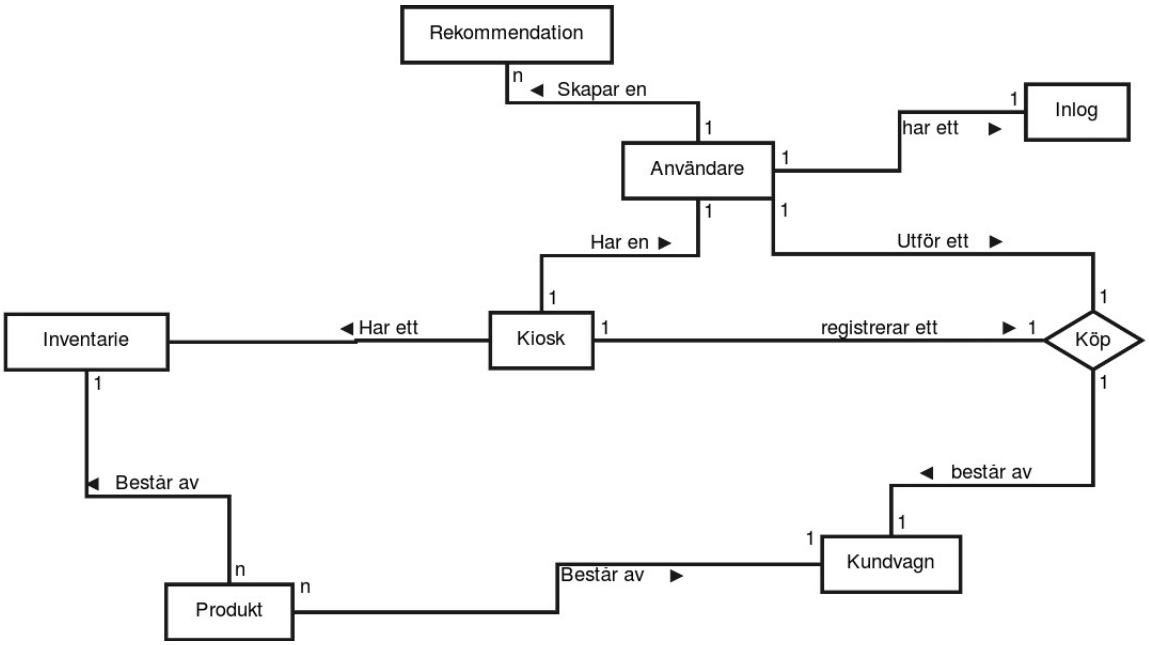

Figure 13: High level overview of *KoskenKiosken*

2.3.1 Class responsibilities

Kiosk

Entiteten Kiosk representerar själva kärnan av applikationen, och fungerar som ett domänaggregat.

Inventarie

Inventariet är hela utbudet av produkter i kiosken. Detta representeras av en lista med n-antal produkter, och ansvarar för att visa användarna vad som finns att köpa och vad det kostar.

Rekommendation

En användare kan rekommendera en produkt att köpa in i kiosken. Det finns ingen gräns på hur många rekommendationer en användare kan skicka in.

Användare

Användare ansvarar för att hålla koll på antalet credits som är kopplat till användaren, och vilka varor som har lagts i den personliga varukorgen. En användare kan också rekommendera in produkter att köpa till kiosken.

Inlogg

Inloggningen ansvarar för att en användare som loggar in skall autentiseras på rätt sätt.

Köp

Köp är en relation mellan en användare och kundvagnen. Köpet genomförs eller misslyckas beroende på ifall en användare har råd med produkterna i varukorgen.

Kundvagn

Kundvagnens ansvar är att hålla i en lista av n-antal produkter, som är tillagda från inventariet. En användare skall kunna köpa det som finns listat i kundvagnen.

Produkt

Produkt representerar en fysik produkt som finns att köpa i kiosken. Dessa skall kunna läggas till successivt i kiosken, beroende på hur utbudet ser ut eller ändras över tid.

3 Peer Review

Objektorienterat Programmeringsprojekt, Peer Review till Haqerbois

Johan Almroth, David Persson, Albin Otterhäll,
Morgan Thowsen, Gustav Pihlquist,

Grupp KlassFörakt

18th October 2019

3.1 Do the design and implementation follow design principles?

Single Responsibility Principle Koden följer *Single Responsibility Principle* väl.

Open-Closed Principle Vi ifrågasätter bland annat hårdkodningen av strängar samt colors. Läs avsnitt **1**.

Liskov Substitution Principle Vi ser inga arv bortsett från de från Android.

Interface Segregation Principle Interfaces är små och kompakta, dock inte många. Förlagsvis hade de delar i modellen som exponeras kunna exponeras mha. interfaces.

Dependency Inversion Principle I fallet med IDeckRepository ser vi ett bra exempel på *DIP*. Då projektet är tämligen stort har vi inte möjlighet att studera mer djupgående gällande brott.

3.1.1 Are design patterns used?

I flera delar av kodbasen finns det design patterns som är halvt implementerade. En klass står det dokumenterat i koden att man har implementerat *immutable pattern*, men man har inte gjort fälten i klassen `final`. Andra klasser har factory metoder implementerade direkt i de konkreta klasserna, dessa bör flyttas till extern klass.

Observer Pattern används som ett färdigt bibliotek, RXJava. Vi vill tillägga att vi gärna ser betydligt större användning av design patterns men det är möjligt att flera inte är utskrivna i *SDD* och vi inte haft möjlighet att pga. tidsramen hitta dem.

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

Projektet verkar vara i stort behov av en refaktorisering. Som det ser ut nu är det väldigt många klasser och kodrader som fortfarande inte används. Till exempel så finns det 4 klasser under modulen *Archive* men när man navigerar till fragmentet i appen är det endast en tom sida. Likadant gäller för *Share-*, *Send-* och *Tools-*fragmenten. Vad vi försöker säga är att vi har tillsynes svårt att motivera ca 50 klasser med något annat än att det måste finnas många lösa trådar. Vi ifrågasätter huruvida gruppen har arbetat interativt.

Implementering av ytterliggare interfaces är att rekommendera. I detta utkast finns endast 8 interfaces på ca: 50 klasser.

Modularisering har använts bra i stora delar av projektet bortsett från modellen. Näst intill samtliga klasser i modellen ligger i samma paket. Förslagsvis hade module-pattern använts här för att tydliggöra dependencies. Vid implementation av detta hittas även dåliga beroenden lätt.

Modellaggregatet går inte instansiera utan IDeckRepository. Med andra ord finns ett underliggande dependency på att något externt tillförs. En bättre implementation av detta vore att ha en metod likt "setDeckReposity" men att modellen funkar och startar utan detta.

Det finns även många hårdkodade strängar både i modell samt androidklasser. För strängar i modellen borde de komma från extern källa så som *IDeckRepository*. För fragment med mer bör de komma från en XML. Detta gör att det blir enkelt att åtgärda typos över hela projektet samt i framtiden öppnar upp för ev. tilläggsspråk. Till detta hör även colors. De hårdkodade färgerna i modellen är svår motiverade sett till Open-Closed-Principle.

3.3 Is the code well tested?

När testerna körs med Coverage så ser vi att stora delar av modellen är testad. Med det sagt så är testerna mycket få och det kan bara förklaras med att det helt enkelt testas flera metoder vid varje test. Vi tror det hade blivit mer intuitivt att börja med att dedikera en testfil per klass och först därefter testa samverkan mellan klasser.
Ett exempel på detta är ArchiveLibraryTest.java . Varken Archive eller Library testas enskilt.

3.4 Is the code documented?

I modellmappen så är koden generellt sett bra kommenterad men det finns vissa ställen där koden inte är särskilt kommenterad. Archive är ett exempel där det inte finns mycket kommentarer att följa. Vissa kommentarer säger inte så mycket, till exempel i FlashCardApp på rad 11. "The library" säger inte mycket till folk som inte är insatta i projektet sedan innan. Lite mer förtydligande hade inte skadat. Kommenterandet genom projektet är blandat, ibland används vanliga kommentarer där det bör varar javaDoc. Det är dock bra att det finns kommentarer i komplexa metoder.

3.5 Are proper names used?

I projektet är namngivningen bra överlag. Namnen på variabler och metoder beskriver bra vad metodens/variabelns syfte är och vad den ska användas till. Det är hjälper mycket att namnen är tydliga, det blir mycket lättare för folk som inte är insatta i projektet att följa strukturen. Observera att Tag redan finns som klass i Android och ni överskuggar denna. Det kan vara en idé att byta denna då det även förekommer som ett keyword i de XML-filer som används av android.

3.6 Does the project use a consistent coding style?

Från vad vi ser så tycker vi att koden i projektet ser konsekvent ut. Namngivning för metoder och variabler och sättet det är implementerade på följer samma mönster genom projektet. Från vad vi ser så sticker inget ut.

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

Koden följer designmönstret MVVM istället för MVC, som är anpassat efter Androidutveckling. Detta följer gruppen bra, det finns specifika ViewModels till varje fragment. Modellen ligger separerad i en egen modul, och är därav isolerad från vyerna.

Själva koden är inte särskilt svårförståelig, men det blir rörigt att veta vad som används och inte, eftersom det finns mycket kod och klasser i projektet som ännu inte används.

3.8 Does the code use proper abstractions?

Flera av de konkreta klasserna i modellen implementerar inte något interface, vilket omöjliggör beroenden på abstraktioner istället för konkreta klasser.

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

Kodbasen är uppdelad i många paket, moduler och klasser. Det ger en tydlig modulär struktur vilket också underlättar för att fortsätta utveckla koden. Vi upplever att vissa klasser skulle kunna implementeras i ett senare skede, då de i

nulgäet agerar ”placeholders”. Då koden är under utveckling faller sig detta dock naturligt, så länge det implementeras på riktigt innan appen släpps.

Överlag följer koden goda design-principer. Vi ser inga tydliga onödiga beroende och MVVM (androids motsvarighet till MVC) följs väl.

4 References

- gson - Google verktyg för att hantera json. Json-strings använd som intern kommunikation i databasen
- Mockito - Mock-verktyg för öka precisionen av tester.
- espresso - tester för Androidkompontenter. **Används inte**
- runner - Testverktyg tillhandahållet av platformen.
- junit - Testverktyg för att skapa tester.
- lifecycle - Används av android för fragments och activites ”lifecycles”.
- annotations - Används för att anmärka i koden om något får eller inte får lov att vara t ex null. ”@nonnull”. Dessa blir s.k. warnings som kan kollas statiskt.
- design - Google designpaket f. knappar mm.
- material - Google designpaket f. knappar mm.
- recyclerview - Används som en prestandaeffektiv lista att scrolla i. Rekommenderas och underhålls av Google.
- legacy-support - Kompabilitetslager för android.
- appcompat. - Compabilitetslager för android.
- constraintlayout - Androidunderhåller XML layout verktyg.
- travis - Continious Integration Tests. Används för löpande tester vid push, merge och pullrequest.
- git - Version Control System.

UML Diagarams

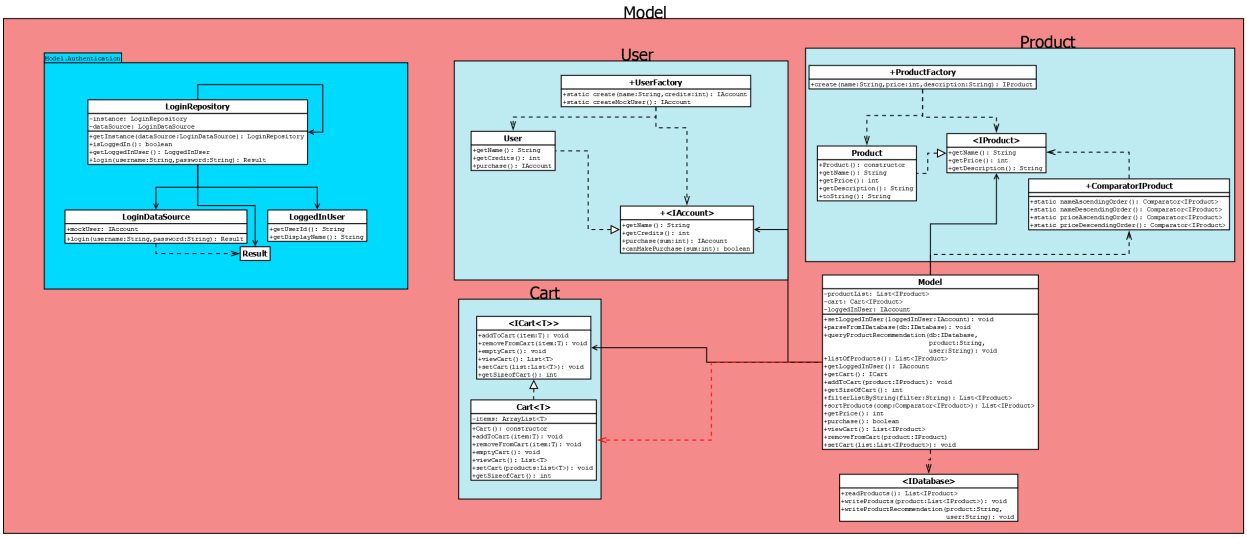

Figure 14: Model Package

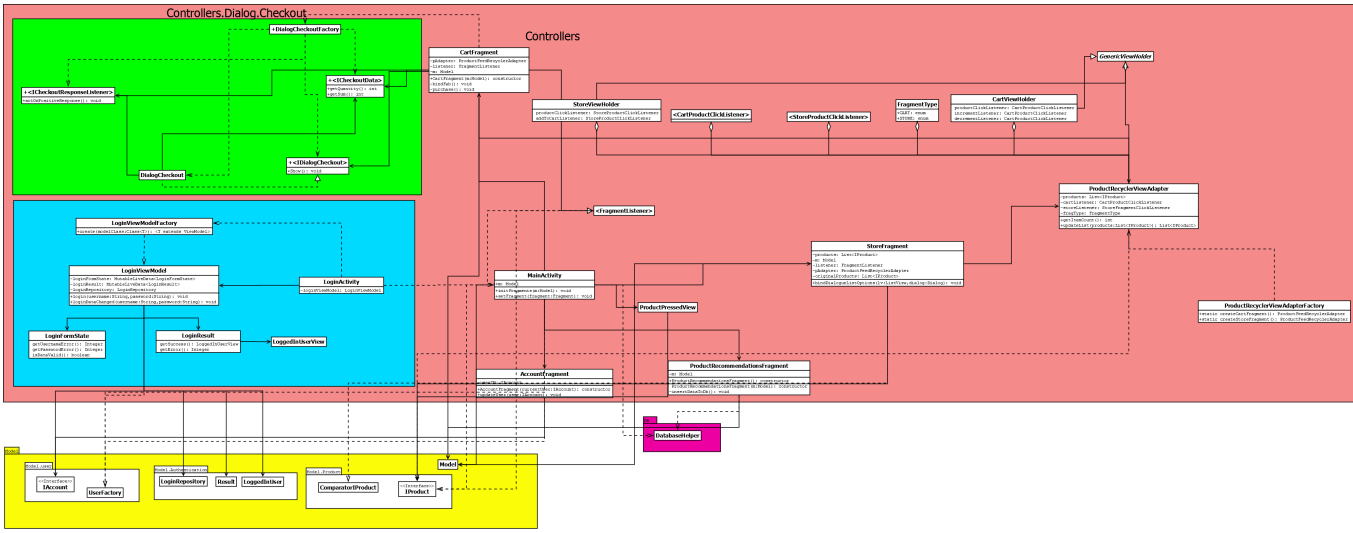

Figure 15: Controllers Package

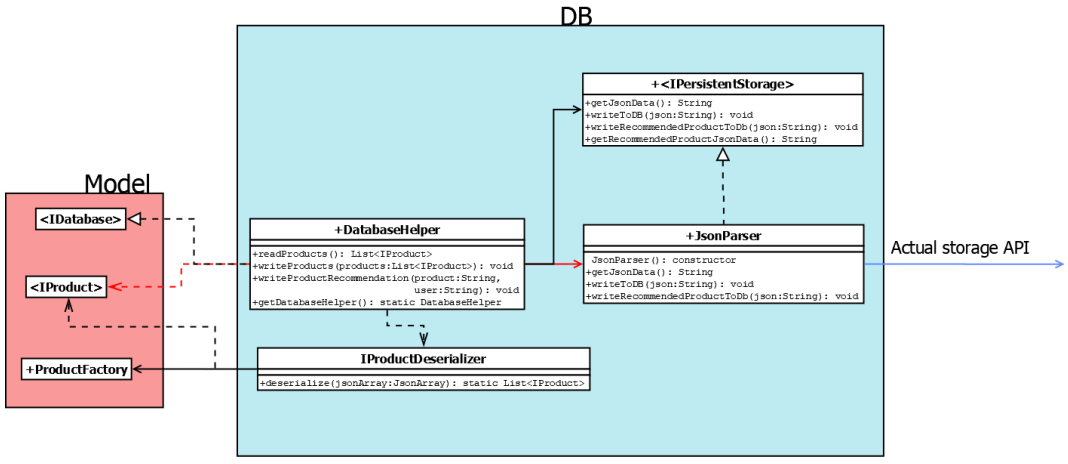

Figure 16: Database Package