En kvittoskannare

Implementering av optisk teckenigenkänning för Android-enheter

Kandidatarbete inom Data- och Informationsteknik

Josefina Andreasson
Mattias Fridén
Cecilia Geijer
Johan Gustavsson
Johannes Jansson
Anton Karlsson

Chalmers tekniska högskola
Göteborgs universitet
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, Juni 2013

A receipt scanner
Implementation of optical character recognition for Android devices

Josefina Andreasson,
Mattias Fridén,
Cecilia Geijer,
Johan Gustavsson,
Johannes Jansson,
Anton Karlsson.

Examiner: Arne Linde and Sven Arne Andreasson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Förord

Rapporten är ett kandidatarbete som utfördes under våren 2013 vid Institutionen för Data- och Informationsteknik, Chalmers Tekniska Högskola. I rapporten beskrivs utvecklingen av en applikation för att fotografera kvitton och med hjälp av optisk teckenigenkänning lagra informationen från dem på en smart mobiltelefon.

Projektgruppen önskar tacka Sven-Arne Andreasson för den tid han lagt på att handleda projektet. Projektgruppen önskar även tacka Magnus Gustafsson och Anna-Lena Fredriksson på Avdelningen för Fackspråk och Kommunikation på Chalmers för den vägledning de gett under arbetet med rapporten.

Abstract

In this thesis the development of an application for Android, used to facilitate the managing of receipt for students, is described. Many students are struggling to keep their expenses and receipts in order. Using optical character recognition, receipts can be read with the built-in camera of a mobile phone and saved in the application for ease of access and simplicity.

A specification of requirements was produced by the group in which the different requirements were divided into several levels of priority. A survey amongst students on Chalmers was used to amend the specifications, as well as confirming the already formulated requirements.

Furthermore, the thesis describes how the different parts of the application were implemented and the agile approach used in the development process. As a result of the work a functional alpha version of the application is now available for further development by the group or by a third party developer.

Keywords: Android, OCR, Receipts, Agile development

Sammanfattning

I denna rapport beskrivs utvecklingen av en Android-applikation vars syfte är att underlätta kvittohanteringen för studenter. Många studenter har svårt att hålla ordning på sina utgifter och sina kvitton. Med hjälp av optisk teckenigenkänning kan kvitton läsas av genom en mobiltelefons inbyggda kamera för att sedan sparas lättillgängligt och lättöverskådligt i applikationen.

Inom projektgruppen togs en kravspecifikation fram i vilken de olika kraven sorterades i olika prioritetsnivåer. En enkätundersökning bland studenter på Chalmers användes för att korrigera krav-specifikationen men bekräftade även i stora drag de redan satta kraven för applikationen.

Vidare beskriver rapporten hur de olika delarna av applikationen implementerades samt det agila arbetssätt som gruppen använde för utvecklingen. Som resultat av arbetet finns nu en fungerande alfaversion av applikationen för vidareutveckling av gruppen eller tredje part.

Nyckelord: Android, OCR, Kvitton, Agil utveckling

Ordlista

Action bar	Den menyrad som löper längst upp i applikationens fönster och innehåller information och valmöjligheter.
Agil utveckling	Ett samlingsnamn för olika arbetssätt med iterativ målsättning och arbetsflöde.
Android	Ett linuxbaserat operativsystem för främst mobila enheter.
API	Ett gränssnitt för kommunikation mot existerande mjukvara. Kommer från engelskans *Application Programming Interface*.
Activity	Ett element i Android som styr användargränssnittet.
Binärisering	Teknik som används för att reducera antalet färger i en bild till två, ofta svart och vit.
ER-diagram	En grafisk representation av hur en viss databas ska vara uppbyggd. ER står för *Entity-relationship*, vilka betonar att så kallade entiteter och relationer är de viktigaste beståndsdelarna i ett ER-diagram.
Google Play	Androids distribueringscenter för applikationer.
iOS	Ett operativsystem från Apple för deras mobila enheter.
Java	Ett objektorienterat programmeringsspråk. Används bland annat till utveckling av applikationer till Android.
Kvitto	Den fysiska pappersversionen av ett kvitto från en affär.
Kvittoavbild	Kortet som tagits på kvittot.
Kvittodata	Applikationens information om ett kvitto.
OCR	Optisk teckenigenkänning, den automatiserade och maskinbaserade översättningen av text från ett analogt till ett digitalt medium. Förkortningen kommer från engelskans *Optical Character Recognition*.
Overflow	Ett begrepp som används i relation till de av Androids menyer som inte har ett reserverat utrymme på skärmen. En overflow-meny är nära ihopkopplad med action bar. Uttrycket betyder på svenska *överflöd*.
Plattform	Ett existerande ramverk av mjuk- och/eller hårdvara som en applikation kan köras på, till exempel en Android-enhet.
Ramverk	Ett färdigt, oftast externt, program vars funktionalitet kan användas av andra program.
Regex	Används för att matcha strängar mot ett visst mönster. Kommer från engelskans *Regular expression* och betyder på svenska *reguljära uttryck*.
Scrum	Ett agilt utvecklingssätt fokuserat på iterativ utveckling och användarberättelser.
SDK	Används för att utveckla mot en specifik plattform. Förkortningen kommer från engelskans *Software Development Kit*.
Wrapper	En kapsling av existerande funktionalitet för användning genom ett gränssnitt.
XML	Ett märkspråk som används för att strukturera data. Används till exempel i Android för att definiera hur vyer ska se ut. Förkortningen kommer från engelskans *Extensible Markup Language*.

Innehållsförteckning

1. Inledning

En budget är något som alla bör ha, men många studenter anser att det är krångligt och tidskrävande. Man skapar en budget, samlar på sig kvitton, och kanske matar in dem i datorn. Tyvärr kommer man oftast inte så långt eftersom tiden och orken inte räcker till. Det vore bra om det fanns ett system som möjliggjorde snabb digitalisering av kvitton, samtidigt som systemet gav en enkel överblick över användarens ekonomiska situation. Frågor som uppkommer är *Vad är ett smidigt sätt att lägga in kvitton? Hur kan man visualisera budget och kvitton på ett bra sätt i mobilen? Vad behöver användare av en sådan mjukvara för funktionalitet?*.

Problemet med lagring av kvitton är intressant ur en teknisk synvinkel då överföring av data från bild till text fortfarande inte är trivialt. Optisk teckenigenkänning, förkortat OCR efter engelskans *Optical Character Recognition*, är ett verktyg som kan användas för att hitta text i bilder, och det är något som det fortfarande bedrivs forskning kring (Menard 2008). Det vore därför intressant att undersöka forskningen inom optisk teckenigenkänning för att hitta ett bra sätt att avläsa och behandla kvitton.

1.1 Syfte

Projektets syfte är i första hand att skapa en fungerande prototyp av en mobil applikation som kan användas till att digitalisera, lagra och presentera information om kvitton på ett smidigt sätt. Målgruppen för applikationen är studenter vid högskola eller universitet. Ett delsyfte är att analysera målgruppens behov och försöka uppfylla dem i ett antal mjukvarukrav. Ett annat är att undersöka olika OCR-tekniker för att utforska om dessa kan möjligöra processen med att digitalisera kvitton eller om det är möjligt att utveckla en egen OCR-teknik.

1.2 Avgränsningar

På grund av den ökade komplexiteten kommer ej projektgruppen att utveckla eller sätta sig in i OCR för handskrivna kvitton. Projektet kommer inte heller att behandla mottagning av elektroniska kvitton direkt från affären, då detta bygger på annan teknik och är för omfattande för detta projekt. Fokus ligger på översikt av ekonomin för privatpersoner snarare än att sträva efter att kunna använda de lagrade kvittona i exempelvis bokförings- eller reklamationssyfte.

1.3 Rapportens upplägg

I kapitel 2 av rapporten beskrivs processen för att fastställa applikationens kravspecifikation, och kapitel 3 består av en domänanalys över en applikation för kvittoskanning. I kapitel 4 sker en omvärldsanalys där ramverk, områden och arbetssätt som är relevanta för projektets genomförande granskas, och utifrån detta fattas ett antal beslut i kapitel 5. Beslutkapitlet paras ihop med relevanta delar av omvärldsanalysen för att skapa en tydlig uppdelning mellan beslutsfattande och inhämtad information. I kapitel 6 redogörs problem, tankegångar och intressanta lösningar som uppkom under implementationsfasen. Rapporten avslutas med en redovisning av resultatet för kvittoskannaren i kapitel 7, följt av en diskussion av utveckling och tillvägagångssätt i kapitel 8, och slutligen arbetets slutsats i kapitel 9.

2. Kravspecifikation för applikationen

Det uppskattas att mellan 40 och 60 procent av alla fel som uppstår under utvecklingen av mjukvara orsakas av dåligt eller felaktigt ställda krav (Leffingwell 1997, Wiegers 1999), och det innebär en kostnadsbesparing att fokusera på kravställning för att sedan helt undvika dyra och tröttsamma problem. Det finns ett antal olika metoder för att försäkra sig om att specifikationerna och kraven möter marknadens efterfrågan. Ulf Eriksson har i boken *Kravhantering för IT-system* skrivit: "*Baserat på systemets intressenter och de mål som ställts upp för systemet kan olika typer av krav samlas in. Det behövs flera olika insamlingstekniker för att samla in en så komplett bild av kraven som möjligt*" (2008, s. 66). Detta betyder att projektgruppen bör titta på, och överväga, flera olika typer av undersökningar.

En effektiv insamlingsform när det gäller kravinsamling är marknadsundersökningar i form av enkäter, vilket är ett av de mest effektiva sätten för att komplettera de redan framtagna kraven och specifikationerna (Davis 2005). En alternativ insamlingsform skulle kunna vara intervjuer, med möjlighet att ställa följdfrågor och djupare frågor (Sharp, Rogers & Preece 2009). Att mer tid tillbringas på varje person i undersökningen kan ses som en nackdel då det krävs mer tid för att få samma statistiska underlag.

Eftersom det är möjligt att få tillfredsställande svar på relevanta frågor för projektet via en enkät, och då det fanns tidspress, valdes intervjun bort som metod för den inledande kravundersökningen. Projektgruppen beslutade istället att intervjufrågor kommer att nyttjas vid användartester som sker i ett senare skede, eftersom det är viktigt att ha möjligheten att ställa följdfrågor och komma närmare testanvändaren och därmed specifika problem med applikationen.

2.1 Utformning av kravundersökning

Med en enkät är det möjligt att ställa specifika frågor med svarsalternativ i kombination med öppna frågor. Docent Claes Herlitz konstaterar att "*all erfarenhet pekar på att svarsfrekvensen i enkätundersökningar har samband med enkätformulärets utformning, omfattning och innehåll. Det finns visst skäl att anta att människor i dag är trötta på alla enkäter som skickas ut till dem och därför är mer selektiva med att besvara dem*" (2011). Utifrån detta bör enkäten bestå av färre öppna och fler stängda frågor med ett antal utvalda svarsalternativ. Detta gör enkäten kort och bör i större utsträckning garantera mer tillförlitliga svar och en större respons, och undviker risken av bortfall på grund av ovan nämnda anledningar.

Som en följd av denna information bestämde gruppen att genomföra en marknadsundersökning i form av en enkät. Den fungerar som en fältstudie vars mål är att få en ungefärlig uppfattning om ifall de preliminära kraven är tillräckliga eller om de behöver omvärderas i prioritetslistan för utvecklingen av funktioner i olika iterationer (se *2.3 Funktionella krav*). Den används även för att kunna ge eventuella användare möjligheten att komma med förslag på mjukvarufunktioner. Frågorna till enkäten som gruppen konstruerade finns i appendix A.

2.2 Analys av kravundersökning

Enkäten gick ut till 100 studenter vid Chalmers Tekniska Högskola. Deltagarna representerar applikationens målgrupp väl då de är studenter, och deras tekniska kunskaper skulle kunna resultera i användbara kommentarer. Enkäten besvarades av 31 personer, vilket ger en svarsfrekvens på 31 %. Det är viktigt att notera att 31 svar är få sett till det totala antalet studenter på Chalmers.

Enkäten visade att 68 % (21 st.) av de eventuella användarna prioriterade enkel användbarhet högre än bland annat prestanda, säkerhet och visuellt tilltalande utseende, se figur 2.1. Säkerhet ansåg 16 % (5 st.) vara viktigast medan 13 % (4 st.) ansåg det vara prestanda, och endast 3 % (1 st.) ansåg att det viktigaste i en budget-/privatekonomiapplikation är utseendet. Den kompletta enkäten med summering av alla svar kan hittas i appendix A.

Figur 2.1: Fördelning av svar på frågan "Vad skulle du tycka var viktigast om du använde en mobilapplikation inom privatekonomi?".

Dessa resultat gav riktlinjer för vad projektet bör fokusera mer på. De riktlinjer som framkom ur enkäten indikerar att användbarhet bör prioriteras högt, men även att det är viktigt att lägga fokus på bra prestanda då det är många som prioriterar detta. Även säkerheten ska prioriteras, vilket görs genom att utveckla vissa säkerhetsfunktioner som exempelvis lösenordsskydd och kryptering av känslig data som lösenord och nätverkstrafik. På grund av användbarheten kommer designen vara en viktig punkt. Designen bör testas och utvärderas under projektets gång genom användbarhetstester mellan de olika iterationerna (Sharp, Rogers & Preece 2009). Prestanda kommer vara ett icke-funktionellt krav, och kommer att appliceras när gruppen implementerar kod innefattande bland annat responstid och databeräkningar.

Nästan alla preliminära funktionella krav uppskattades i av de svarande, med undantag av mjukvarufunktionen att kunna dela ut kvitton till sociala nätverk. Denna funktion prioriterades därför om från prioritetsnivå 2 till en senare iteration, nämligen prioritetsnivå 4. Flera användare hade även bra förslag på funktioner applikationen kunde ha. En sammanställning av förslag på funktioner från enkätsvaren kan ses i appendix A. Lämpliga funktioner som användarna ville se prioriterades också in i de olika prioritetsgrupperna för de funktionella kraven.

2.3 Funktionella krav

Funktionella krav speglar de funktioner som ska implementeras i applikationen. För att visa vilka krav som är viktiga att implementera tidigt har de delats upp i olika prioritetsnivåer. Notera att inom de olika nivåerna är kraven listade utan inbördes ordning.

2.3.1 Första prioritet

De krav som är första prioritet är de som är nödvändiga för applikationens grundfunktionalitet.

- Fotografera ett kvitto med enhetens inbyggda kamera
- Manuell inmatning av kvittoinformation
- Analysera text från kort
    - Spara informationen från den analyserade bilden i en databas
- Presentera den tolkade texten från kvittot med hjälp av datum, summa, produkter och kategori

Ovan finns de krav som krävs för att kunna föra in ett kvitto i applikationen, både genom att ta kort på det och genom att manuellt fylla i fält.

2.3.2 Andra prioritet

Prioritetsnivå två inkluderar funktionalitet som inte är grundläggande men viktig för att applikationen ska vara användbar och lätt att använda.

- Visa grafer som rör inlagda kvitton samt budgetplan, till exempel uppdelade enligt tid eller område
- Säker hantering av data
    - Lösenordsskyddad åtkomst under uppstart (inställningsbart)
    - Kryptering av nätverkstrafik
- Säkerhetsinställningar (se *Säker hantering av data* ovan)
- Exportera kvitton till datorn i flera format, exempelvis csv
- Importera redan existerande bilder på kvitton till applikationen
- Hantera kvittodata
    - Ta bort kvittodata
    - Ändra kvittodata
- Manuell inmatning av inkomster och utgifter
- Färgkodning av kategorier

Här inkluderas till exempel redigering av kvitton efter att de har blivit tillagda och visning av information i grafer. Detta gör det lättare för användaren att visualisera och hantera sin data, vilket enligt oss höjer kvaliteten på applikationen.

2.3.3 Tredje prioritet

I prioritetsnivå tre finns funktionalitet för att göra applikationen säkrare och snabbare.

- Säkerhetskopiering: importera/exportera all programdata till exempelvis datorn
- Automatisk igenkänning av specifika delar av ett kvitto (pris, produkt, med mera)
- Användning av en extern server för att hantera beräkningar, lagring och säkerhetskopiering
    - Användning av en lokal SQL-databas på telefonen som mellanlagring emellan extern databaskoppling.
    - Sökning bland existerande kvitton
- Inställningar, till exempel:
    - Spara bildfilen permanent eller tillfälligt
    - Ta bort sparade bilder

De flesta av ovanstående krav behandlas i bakgrunden av applikationen och användaren kommer inte direkt påverkas av dem. På grund av deras låga synlighet har dessa optimeringar lagts som tredje prioritet.

2.3.4 Fjärde prioritet

De krav som skulle kunna vara intressanta tillägg, men som inte tillför någon viktig funktionalitet har placerats i fjärde prioritet.

- Delning till sociala nätverk såsom Facebook
- Stöd för andra språk
- Implementera Bankdroids API (Google 2013a) för att kunna se kontoförändringar
- Uppstartsmeddelande baserat på statistik från existerande kvitton

Dessa funktioner är till stor det fokuserade på att vidga gränserna för projektet, till exempel genom koppling till sociala nätverk, men är inte tillräckligt viktiga för att implementeras tidigare.

2.4 Icke-funktionella krav

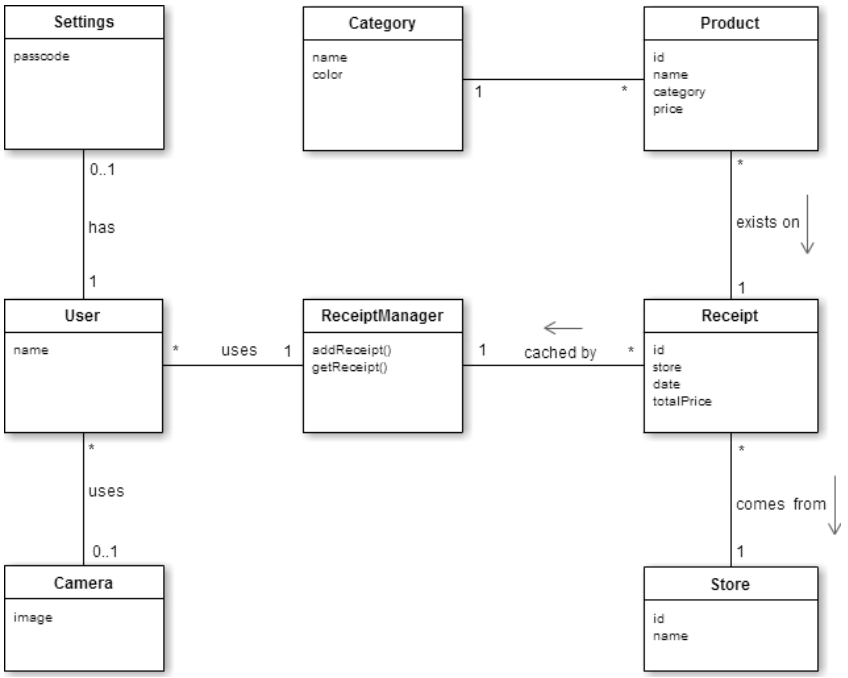
Icke-funktionella krav används för att bedöma hela systemet snarare är specifika funktioner. Detta projekt utgår från de nedan listade kraven.

- Plattform - Stöd för enheter med operativsystemet Android.
- Användbarhet - Ett enkelt användargränssnitt som alla kan använda. Det ska vara lätt för användaren att förstå vad nästa steg är.
- Pålitlighet - Applikationen beter sig på ett förutsägbart sätt, och ser till exempel till att ingen data oväntat försvinner.
- Installation och distribuering - Applikationen kommer att behöva installeras till telefonen innan den kan användas. Det enklaste sättet att distribuera den på är via Google Play.
- Prestanda - Applikationen ska vara snabb och ge rimlig responstid mellan system och användare.

Överlag är målet att tillverka en applikation till Android som användaren tycker om att använda, och de icke-funktionella kraven är specificerade utifrån det.

3. Domänanalys

En domänmodell är en modell som beskriver projektets nyckelkoncept och problemdomän. Man kan i figur 3.1 se vilka nyckeldelar som kommer behövas i projektet och deras relation till varandra. Modellen ska kunna appliceras i ett valfritt objektorienterat system.



Figur 3.1: Kvittoskannarens domänmodell. Modellen visar applikationens olika nyckelkoncept och deras relationer. Pilarna visar i vilken riktning relationerna gäller, till exempel utläses relationen mellan Product och Receipt som *Product exist on Receipt*.

I detta projekt är de olika beståndsdelarna från modellen synliga i diagrammet. Koncepten förklaras närmare nedan.

- ReceiptManager: Hanterar och lagrar kvitton från olika användare.
- Receipt: Representation av kvittot. Har många produkter och innehåller relevant information, som till exempel vilken affär som har besökts.
- Product: Representerar en produkt som det kan finnas många av på ett kvitto.
- User: Varje användare kan ha inställningar kopplade till sig. Alla användare har tillgång till en ReceiptManager, och kan använda sig av bilder som fångas ur en kamera.
- Camera: Verktyg som används av användare för att ta kort.
- Settings: Personliga inställningar kopplat till en användare.
- Category: En kategori som kan beskriva produkter. Kan till exempel vara “Mat” eller “Kläder”.

Koncepten ovan utgör en grundstomme för projektet och tillför en gemensam bild och uppfattning av systemet som ska implementeras. Detta gör att det sedan är enkelt att bygga på modellen och därmed också systemet.

4. Omvärldsanalys

En lämplig metod för att höja kvaliteten på ett projekt är att göra en omvärldsanalys, även kallad förstudie, av områden som är relaterade till projektet. Förstudien innefattar bland annat analys av olika delar som kan ingå i slutprodukten, exempelvis plattform, mjukvara, ramverk och programspråk. Omvärldsanalysen utforskar även hur gruppen bör strukturera arbetet genom att undersöka olika arbetsmetoder inom mjukvaruutveckling. Följaktligen innehåller förstudien även en analytisk genomgång av redan existerande tjänster, algoritmer och andra lösningar i olika tekniska områden som kan bidra till att uppnå syftet och målet.

4.1 Existerande tjänster

Innan ett projekt inleds är det ofta intressant att undersöka om det finns liknande tjänster tillgängliga på marknaden, och eftersom kvittokulturen ser lite olika ut i olika länder har endast svenska applikationer undersökts. Projektgruppen fann tre olika produkter som är snarlika den produkt som projektet kommer att producera: Kvittar, SparaKvittot och Recibo. Dessa tjänster marknadsförs som ett sätt att få in kvitton i mobiltelefonen och slippa papperskopian, och de delar målgruppen företag. SparaKvittot använder sig endast av kvitton som från början är elektroniska, medan Kvittar och Recibo även erbjuder möjligheten att fotografera kvittot. Ingen av dem använder sig av OCR för läsa av informationen på kvittot, och applikationerna är inriktade på bokföring och förvaring i molnet (Kvittar 2013; Sparakvittot 2012; Recibo 2013).

Eftersom de existerande applikationerna har en annan målgrupp och syfte än vad som planerats för projektets applikation drar gruppen slutsatsen att det just nu inte verkar finnas en likvärdig produkt på marknaden. Detta gör projektet unikt och värt att genomföra eftersom det inte finns en bra kvittohanterare för studenter.

4.2 Analys av plattform

Det var redan från början bestämt att Android skulle användas som plattform till applikationen. Det valdes ändå att utforska möjliga alternativ och utvärdera valet av Android.

I dagens tekniska samhälle finns en rad olika plattformar att utveckla till. Projektet har som fokus att rikta in sig på plattformar som kan användas på portabla enheter, främst mobiltelefoner, då syftet är att hitta ett enkelt sätt att digitalisera kvitton oberoende av var användaren befinner sig. Följaktligen bör processen utföras så att användare kan vara i exempelvis affären och där ha möjligheten att digitalisera sitt kvitto. Frågan är vilken mobilplattform som passar utvecklargruppen bäst, både när det gäller att uppnå ett bra resultat och att tillmötesgå användarnas behov.

En undersökning från Gartner Incorporation baserad på fjärde kvartalet av 2012 visar att operativsystemet Android, som ägs av Google Incorporation, har 69,7% marknadsandelar av mobilmarknaden. Jämförelsevis har iOS, som ägs av Apple Incorporation, enligt statistiken 20,9% av marknaden (Rivera & Van der Meulen 2013). Det framkommer alltså tydligt att iOS och Android är två stora operativsystem som är populära bland mobilanvändare, och är därför intressanta att utveckla till.

4.2.1 Android

Android är licenserad under Apache 2.0 (Android 2013a) vilket innebär att vem som helst får modifiera och distribuera operativsystemet. Android används därför av olika tillverkare vars modeller har olika skärmupplösningar och hårdvarukombinationer, vilket medför att det aldrig går att garantera att programmet uppför sig likadant på alla enheter.

Android finns i ett antal olika versioner som har tillgång till olika funktioner, och det är det viktigt att betänka vilken version en uppbyggnad av en Android-applikation ska använda. Mängden användare varierar mellan versionerna, men version 2.2 och framåt täcker 97,6% av Android-marknaden (Android 2013b).

Plattformen är öppen, och källkoden är fritt tillgänglig via Androids hemsida (Android 2013c). Öppenheten möjliggör att utvecklare kan inspektera källkoden för att undersöka hur olika saker är implementerade, vilket leder till att det är lätt att lära sig Androids kodstruktur.

Jämfört med utveckling till iOS (se *4.2.2 iOS*) finns det inte lika många restriktioner på vilken utvecklingsmiljö som får användas för utveckling av Android-applikationer; det fungerar till exempel att utveckla i Linux, Max OS X (10.5.8 eller högre) och Windows (XP eller högre) (Android 2013d). Själva programmeringen till Android sker i Java, med ett ramverk som anpassats till mobilprogrammering. Vissa av språkets delar har blivit ersatta, till exempel har uppbyggnaden av det grafiska användargränssnittet ersatts av ett XML-baserat system (Goadrich & Rogers 2012).

4.2.2 iOS

För att kunna utveckla till Apples produkter är det nödvändigt att använda Apples produkter, iOS-utvecklingen måste nämligen ske med hjälp av en dator med Mac OS X (Sadun 2011). Detta kan ses som en nackdel eftersom utvecklare inte är fria till att välja den miljö som passar dem bäst. Fördelen med det är att man lättare kan få stöd av andra utvecklare eftersom alla utvecklar på samma plattform.

Programmering i iOS sker med hjälp av programmeringsspråket Objective-C. En skillnad mot exempelvis Java är att programmeraren har mer kontroll över minneshanteringen, vilket kan leda till effektivare programvara men lägger även mer ansvar på programmeraren (Zarra 2005). Utöver minneshanteringen har iOS flera olika funktionella stöd, exempelvis för databashantering genom SQLite (Goadrich & Rogers 2012) och för 2D- och 3D-grafik med grafikbiblioteket OpenGL ES 1.1/2.0 (Sadun 2005). När det gäller implementering använder utvecklare under iOS ofta en utvecklingsmiljö kallad Xcode, där uppbyggnaden av de grafiska delarna sker med hjälp av en inbyggd modul kallad Interface Builder. Denna modul är integrerad med Xcode från version 4 och framåt (Sadun 2005).

4.2.3 Webbaserad plattform

Webbaserade plattformar är, som namnet antyder, plattformar tillgängliga via Internet och de är därför oberoende av vilket operativsystem som används. Istället baseras upplevelsen på en webbläsare. Detta betyder att applikationens kod inte behöver anpassas efter olika enheter. Även utseendet på applikationen kan hållas konsekvent över olika operativsystem. Undvikandet av kodduplicering leder till att det blir lättare för utvecklaren att underhålla applikationen, men om utvecklaren vill stödja flera olika webbläsare kan problem uppstå, liknande dem knutna till anpassningen till flera olika operativsystem (Charland & Leroux 2011).

Prestandamässigt är webbaserade plattformar något långsammare, men det är en skillnad som blir mindre och mindre. Svårigheten med en webbaserad applikation ligger istället i att åtkomsten till enhetens inbyggda funktioner är begränsad (Charland & Leroux 2011). Eftersom applikationer med en webbplattform inte distribueras via till exempel Google Play blir det även svårare för användarna att hitta dem.

4.3 Ramverk

Mjukvaruprojekt kan behöva använda sig av flera beståndsdelar, vilket styrks av Riehle i boken *Framework Design - A Role Modeling Approach*, där han skriver att delarna kan existera som olika ramverk i projektet. Existerande kod samt design kan då återanvändas för att få projektet att framskrida snabbare och med ökad produktivitet (2000).

4.3.1 Android SDK

För att kunna utveckla och testa mjukvara till Android behövs Android Software Development Kit (SDK). Verktyget Android SDK Manager används för att hämta SDK-plattformar, API för den version som hämtats, källkod, kodexempel samt emulatorer. För att kunna testa applikationer under olika omständigheter finns det tillgång till en emulator som simulerar hård- och mjukvara (Android 2013d).

4.3.2 Grafer

Av kravundersökningen och domänanalysen att döma kommer delar av applikationen innefatta grafer för att visa data från kvitton. Därför är det relevant att analysera olika befintliga ramverk för att kunna se vad som finns att tillgå.

Ett bibliotek med öppen källkod för att bygga och visualisera olika sorters grafer i Android är aChartEngine (4ViewSoft Company 2012). Att använda sig av aChartEngine är enkelt då det finns bra dokumentation och exempel att utgå ifrån. Graferna är estetiskt tilltalande och det går lätt att anpassa färg, layout och text. Biblioteket stödjer många typer av grafer, exempelvis stapel-, linje- och tårtbitsdiagram. Detta gör det möjligt att visualisera data på många olika sätt beroende på behov.

Ett annat ramverk är ChartDroid, ett externt program som applikationer kan använda sig av (ChartDroid 2010). För att kunna använda ChartDroid måste användaren först installera ChartDroid. Vid utveckling med ChartDroid skapar utvecklaren en `Content Provider` (Android 2013e) som är ett dataset av information som grafen ska bygga på. Detta dataset skickas sedan via intent till ChartDroid som skapar grafen.

4.3.3 Testning

Att testa mjukvara utgör en viktig del i utvecklingsprocessen, då det är möjligt att motverka både allvarliga och mindre allvarliga fel och brister i programvaran. Att ha bra tester ger också en säkerhet, eftersom man vid förändringar direkt kan se om programmet fortfarande fungerar som det är tänkt. För att kunna göra detta på ett smidigt sätt finns det olika ramverk till förfogande.

För att underlätta testning i Android finns JUnit, ett ramverk som används för att testa Java-kod (Gamma 2009). Med ramverket är det möjligt att skriva automatiserade testfall som kan köras flera gånger under projektet. Detta är smidigt vid uppdatering av kod för att verifiera att inga defekter uppstått.

Varje test körs ett flertal gånger med olika tillstånd och parametrar för att täcka alla möjliga fall. Resultatet från den programdel som skall verifieras jämförs med förväntat värde. Det svåra med att testa programkod generellt sett är att skriva bra testfall som täcker alla eller tillräckligt många relevanta tillstånd.

Under utvecklingen i Android förkommer mycket programmering av gränssnitt och många funktioner är byggda på respons från användaren, vilket inte går att testa på ett bra sätt med enhetstester. För att enkelt kunna testa programdelar som hör ihop med gränssnitt finns ramverket Robotium som skall göra testprocessen enkel för "blackbox testing", vilket betyder att man utför test utan närmare kännedom om systemet (Robotium 2012). Med Robotium skrivs automatiserade tester på samma sätt som i JUnit, men användarens interaktion med applikationen och dess gränssnitt automatiseras genom kod som stegvis utför användarens operationer tills gränssnittet är i ett utvärderingsbart läge.

4.4 Optisk teckenigenkänning (OCR)

Att smidigt överföra kvittotext till digitaliserad kvittodata är en central uppgift i projektet, och därför uppkom funktionella krav på att kunna använda telefonens kamera för att underlätta överföringsprocessen. Detta skulle innefatta bildbearbetning, textigenkänning och mönstermatchning för att kunna extrahera texten från kvitton. I dagens tekniska samhälle används OCR på många olika plattformar, till exempel hemsidor, servrar, datorprogram och mobiltelefoner.

4.4.1 OCRs historia

Enligt en arbetsrapport från Stanford Universitet som är skriven av Bhaskar, Lavassar och Green (2010) är det första patentet som behandlar OCR från Tyskland år 1929, tillhörande tysken Gustav Tauscheck. Detta patent ska handla om en maskin som berör OCR-teknik. Tauscheck har dessutom en liknande eller modifierad maskin patenterad i USA från år 1935 (Tauschek 1935).

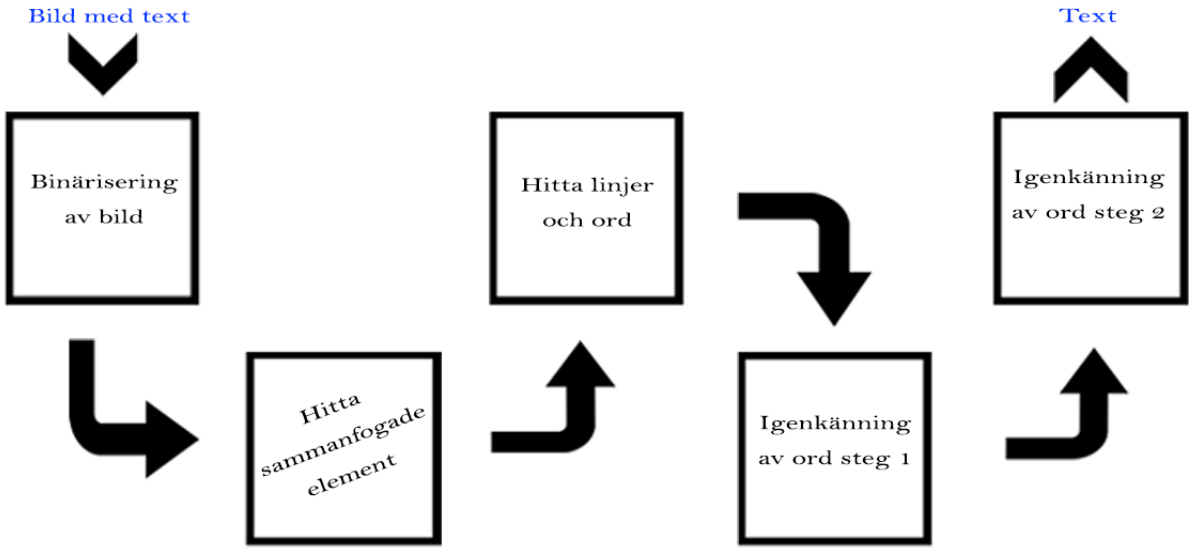
I boken *Character Recognition Systems - A Guide for Students and Practioners* skriven av Cheriet et. al. (2007) kan man läsa att OCR uppkom under 1950-talet, då forskare började med att extrahera siffror och text ur bilder. Till en början utfördes textigenkänning via mekaniska och optiska verktyg och senare med hjälp av fotoceller. Utvecklingen fick sitt genombrott tack vare teknikens utveckling under 1960- och 1970-talet. Under dessa årtionden kom OCR att återfinnas på flera olika institutioner såsom sjukhus, banker och postkontor.

4.4.2 Tesseract

Mellan 1984 och 1994 arbetade en utvecklingsgrupp på Hewlett-Packards laboratorium ledd av Ray Smith (Google 2013b) med att ta fram en OCR-motor (Google 2012). Målet var att ta fram en tilläggsfunktion som Hewlett-Packard kunde använda till de skrivbordsskannrar som utvecklades (Smith 2007a), men funktionen kom aldrig att användas i en HP-produkt (Smith 2007b).

Programvaran som Ray Smith och utvecklargruppen utformade fick senare namnet Tesseract och gjordes år 2005 tillgänglig för allmänheten i och med att den släpptes som ett projekt med öppen källkod under en Apache Licens 2.0 (Smith 2007a). I dagsläget stödjer Google projektet (Bhaskar, Lavassar & Green 2010) som finns tillgänglig via Tesseracts hemsida (Google 2013b). Källkoden är skrivet i programmeringsspråket C/C++ och kan därmed exempelvis integreras i ett projekt genom att referera till den kompilerade koden (Patel, Patel & Patel 2012).

Som synes ovan går det att applicera Tesseract på flera olika plattformar. Det har även gjorts flera implementeringar i form av wrappers i olika språk. Dessa är program, bibliotek eller klasser som har som uppgift att endast anropa underliggande och motsvarande funktionalitet, exempelvis C/C++-koden till Tesseract. Detta underlättar valet av plattform, eftersom språket kan anpassas till plattformen eller vad utvecklare föredrar. Tesseract kan alltså integreras direkt till en mobilplattform eller till en server som utför teckenigenkänning.



Figur 4.1: Illustration av Tesseracts uppbyggnad för extrahering av text ur en bild (inspirerad av Patel, Patel & Patel 2012, s. 51).

För att extrahera texten ur en bild tillämpar Tesseract flera stegvisa procedurer (Smith 2007a) vilket illustreras i figur 4.1. Det första steget som Tesseract utför är en binärisering, det vill säga transformering av bildens färger till endast svart och vit. Detta sker genom en metod kallad Otsu, enligt en arbetsrapport skriven av Bhaskar, Lavassar & Green (2010) från Stanford University. Nästa steg i proceduren handlar om att hitta potentiella bokstäver och siffror; ett steg som blir enkelt med en binäriserad bild eftersom bilden endast har två färger. Möjliga bokstäver kan enkelt markeras ut efter analys av pixlarnas färger (Smith 2007b; Bhaskar, Lavassar och Green 2010). I tredje steget försöker man hitta hela textlinjer med hjälp av de markerade delarna från tidigare steg.

När alla förberedande procedurer färdigställts återstår att känna igen de markerade bokstäverna och orden. Detta utförs genom små olika steg, där ett steg handlar om att jämföra redan existerande bilder på kända bokstäver som är inlagda på förhand (Smith 2007a).

4.4.3 ABBYY

ABBYY ett företag som utvecklar olika tekniska produkter. De jobbar till exempel med teknisk hantering av dokument i olika slag. ABBYY är bland annat kända för sin OCR-mjukvara FineReader, som är en kommersiell produkt (ABBYY 2013a).

Ett verktyg som ABBYY har tagit fram är en optisk teckenigenkänningsmotor för mobiler. Motorn stödjer Android från version 2.2 och uppåt. För att få tillgång till motorn kan en provversion hämtas (ABBYY 2013b). Utöver denna mobila motor har ABBYY även en molntjänst som går att använda med tillgängligt API. Denna tjänst är dock inte fri att använda kommersiellt (OCR SDK 2013).

4.4.4 Optimering av optisk teckenigenkänning

Det finns projekt som försöker optimera textigenkänningsprocessen genom att återanvända olika existerande bibliotek ihop med optimeringar. Ett projekt som är uppbyggt på detta sätt är OCRopus som är skrivet i programmeringsspråket Python. I projektet utförs textigenkänning på dokument, och koden är öppen under en Apache Licence 2.0. Det går att hämta OCRopus via dess projekthemsida (OCRopus 2013a).

OCRopus använder Tesseract och MLP-baserad (på engelska *multilayer perception*) igenkänning av bokstäver. MLP-baserad identifiering bygger på att man gör en graf bestående av de bokstäver en figur kan tänkas föreställa, kombinerat med avläsningen av bokstäver (Breuel 2008).

Det finns saker som projektet inte stödjer i nuläget, till exempel bilder som innehåller handskrivna texter. Det är även svårt att hantera bilder tagna med kamera eftersom projektet förlitar sig på högupplösta bilder från en skannerenhet som bör ha en upplösning mellan 300-600 dpi. Vidare stödjer projektet endast engelska texter (OCRopus 2013b).

4.5 Bildbehandling

I *4.4.2 Tesseract* nämns att Tesseract använder sig utav Otsus metod för binärisering. För att kunna förbättra förbehandlingsprocessen har gruppen valt att undersöka alternativ till Otsu. Enligt Mo och Mathews är lokal binärisering, vilket innebär att gränsvärdet det bestäms lokalt runt en pixel, effektiv vid binärisering av kort på kvitton (1998), och lokal binärisering står därför i fokus. Det finns många metoder för utförande av lokal binärisering, och i Singhs rapport nämns Sauvola och Niblack som de två som ger bäst resultat (2011). Därför har dessa undersökts närmare. För att tydligare se skillnad mellan de olika algoritmerna används figur 4.2 som utgångspunkt för visning av de olika metodernas resultat.

Figur 4.2: Exempelbild för jämförelse av olika binäriseringsmetoder.

4.5.1 Otsus metod för binärisering

Otsus metod för binärisering använder sig av ett globalt gränsvärde för att bestämma pixlarnas nya färgvärde. Detta gränsvärde kan antingen bestämmas statiskt eller räknas ut efter färgvärdena i originalbilden (Otsu 1979). En fördel med denna metod är att den är effektiv eftersom gränsvärdet antingen redan finns eller bara behöver räknas ut en gång. Endast en genomlöpning av pixlarna måste göras eftersom de är oberoende av varandra efter att gränsvärdet har beräknas. Inte heller behöver en

kopia göras av originalbilden då pixlarnas färg kan ändras direkt i originalbilden. Otsus metod är därför resurssnål både i beräkningskraft och i minnesåtgång.

En nackdel är svårigheten med att binärisera bilder med varierande bakgrundsfärg. Då gränsvärdet är globalt tas ingen hänsyn till skillnader i lokala områden i bilden, istället bestäms en pixels nya färg enbart utifrån dess gamla färg och förhållandet till gränsvärdet. Detta resulterar i en bild med en tydlig gräns mellan vad datorn tolkar som svart och vitt. I figur 4.3 är det lätt att se gränsen mellan vad som anses vara bakgrund och vad den anses vara förgrund.

Figur 4.3: Resultat efter applicering av Otsus metod för global binärisering på figur 4.2.

4.5.2 Niblacks metod för binärisering

Niblacks metod för binärisering använder sig av medelvärde och standardavvikelse för att bestämma varje pixels slutgiltiga värde. Medelvärdet och standardavvikelsen beräknas utifrån ett fönster med bestämd storlek runt varje pixel, vilket möjliggör ett lokalt bedömande av varje pixel (Sauvola & Pietikainen 2000). Enkelt förklarat undersöker algoritmen en liten del av bilden, och avgör därifrån om skillnaden mellan pixlarnas färgvärden är tillräckligt stor för att de ska räknas som olika. I figur 4.4 är det tydligt att resultatet inte längre är lika påverkat av den gradienta bakgrunden som i Otsus metod för binärisering. Detta är tack vare den lokala bedömningen av varje pixel.

Ekvation 4.1 visar Niblacks metod för att beräkna gränsvärdet för en pixel. Ekvationen innehåller en konstant med beteckningen *k* som reglerar hur stor påverkan standardavvikelsen har på gränsvärdet.

Figur 4.4: Resultat efter applicering av Niblacks metod för lokal binärisering på figur 4.2.

$$T(x,y) = \mu(x,y) + k * \sigma(x,y)$$

Ekvation 4.1: Niblacks formel för gränsvärde vid lokal binärisering (Sauvola & Pietikainen 2000).

4.5.3 Sauvolas metod för binärisering

Sauvolas metod för binärisering har sitt ursprung i Niblacks metod, och använder därför också ett lokalt gränsvärde. Skillnaden mellan Sauvola och Niblack är, förutom resultatet, formeln som används för att bestämma gränsvärde efter att medelvärde och standardavvikelse är uträknade (se ekvation 4.2). Förutom konstanten *k*, som används i Niblacks metod, finns även konstanten *R* som beskriver standardavvikelsens maximala värde och som för RGB-värden mellan 0 och 255 har värdet 128 (Sauvola & Pietikainen 2000). Resultatet från Sauvolas metod för binärisering kan ses i figur 4.5.

Figur 4.5: Resultat efter applicering av Sauvolas metod för lokal binärisering på figur 4.2.

$$T(x,y) = \mu(x,y) * \left(1 + k\left(\frac{\sigma(x,y)}{R} - 1\right)\right)$$

Ekvation 4.2: Sauvolas formel för gränsvärde vid lokal binärisering (Sauvola & Pietikainen 2000).

4.6 Agil utveckling

Agil utveckling är ett samlingsnamn för ett flertal olika utvecklingsmetoder, bland annat Scrum och eXtreme Programming. Det de olika metoderna har gemensamt är att de lyder under ett manifest som definierar riktlinjer för hur utvecklingen skall ske. Riktlinjerna föreslår till exempel att individer, interaktion, fungerande mjukvara, och att reagera på förändringar ska värderas högre än att till exempel följa en uppgjord plan eller lägga fokus på omfattande dokumentation (Pham & Pham 2012; Björkholm & Brattberg 2010).

Det som ofta brukas utpekas som motsatsen till agila processer är vattenfallsmodellen, som är en så kallad förutseende modell. Den utgår från att stora mängder tid läggs på kravställning i början av projektet, vilket leder till förlust av både tid och pengar då det visat sig att kraven i projekt förändras i minst 25 % av fallen, och då 50 % av funktionaliteten som efterfrågas i början inte används. Ett mjukvaruprojekt brukar anses vara adaptivt och iterativt snarare än förutseende, vilket är varför agila processer ofta passar bättre än vattenfallsmodellen (Whitaker 2009).

Fördelen med agila metoder är många. Bland annat har det visat sig att både flexibiliteten och effektiviteten på arbetsplatsen höjs. Användningen av iterationer, vilket är cykler i utvecklingen, betyder också att risktagandet minimeras. Fokus läggs på att prioritera den viktigaste funktionaliteten, hålla hög kvalitet på det producerade, och att vara öppna och transparenta. För att kunna uppnå dessa mål krävs det effektiv kommunikation, och i agila metoder framhålls öppna konversationer över skrivna dokument. Detta innebär att utvecklarna måste umgås så gott som dagligen för att kommunikationen inte skall bryta ned, och för att undvika ett antal vanliga källor till slöseri av tid, såsom onödiga funktioner, onödigt arbete och frustration (Björkholm & Brattberg 2010; Whitaker 2009).

Scrum är en agil metod centrerad kring sprintar och korta dagliga möten. Scrum beskrivs av vissa som en metod där det är viktigt att anpassa projektet allteftersom man lär sig mer. Scrums arbetsflöde cirklar kring en funktionalitetslista, användarberättelser, och sprintar på en till fyra veckor. Några av fördelarna med att använda denna arbetsprocess är att tiden utnyttjas på ett effektivare sätt, och att riskelementet i projektet minimeras på grund av inspektions- och anpassningsmomenten i varje sprint (Björkholm & Brattberg 2010; Pham & Pham 2012; Whitaker 2010).

4.7 Databashantering

Domänanalysen (se *3.0 Domänanalys*) uttrycker att systemet kommer att behöva lagra kvittodata, vilket kan ske med olika slags databastyper. Den databastyp som finns tillgänglig vid utveckling till Android är SQLite, en databashanterare som ligger lokalt på användarens enhet. SQLite är integrerat i Android och är därmed lätt för utvecklaren att använda. Alternativet är att projektet använder sig av en extern server med valfritt databashanteringssystem installerat.

4.7.1 Intern databashantering

SQLite är en databashanterare som är inbyggd i Android och iOS. En fördel med att använda SQLite, jämfört med en extern databas, är att ingen internetuppkoppling krävs. Varje användare har lagring och hantering av data på sin egen enhet, och därmed sker ingen nätverkskommunikation mellan telefon och server, vilket leder till att det går snabbare att ladda och visa kvittodata från databasen. Detta är en stor skillnad mot att använda en extern databashanterare, där alla användare kopplar upp sig mot samma system som sedan skickar tillbaka data (Android 2013f).

Ett problem med detta koncept av datahantering är att om någonting skulle gå fel, som exempelvis en telefonkrasch, förlorar man också den data som är lagrad i databasen. En extern lösning skulle i samma situation behålla innehållet i databasen oavsett vad som händer med telefonen, vilket är fördelaktigt.

4.7.2 Extern databashantering

En fördel med att använda en extern databas är att det finns större möjlighet att välja databashanteringssystem efter kunskap och behov. Olika databashanteringssystem fungerar på olika sätt och har olika funktioner implementerade. En funktion som skulle vara till användning för projektet är säkerhetskopiering av data, eftersom ett externt databashanteringssystem skulle upprätthålla all data på samma ställe, oberoende av en telefonkrasch eller dylikt.

4.8 Designstandarder för Android

Android är noga med att hålla en enhetlig design i sina applikationer, och råder utvecklare att följa deras standarder. På deras hemsida finns bland annat guider, färgvärden, mått och ikoner att använda sig av i designprocessen (Android 2013g). Dock har mycket hänt sedan Android 2.2, och riktlinjerna på hemsidan stämmer inte alltid överens med designtillgångarna i API-nivå 8 som projektet bygger på (se *5.2 Beslut om användande av plattform*).

4.8.1 Action bar

I Android 3.0 introducerades det nya konceptet action bar som har kommit att bli ett av de viktigaste designinslagen för Android-applikationer (Android 2013g). Androids exempel på en action bar finns att se i figur 4.6, som illustrerar den övergripande strukturen på menyraden. En action bar underlättar navigationen i applikationen genom att ersätta den meny som tidigare fanns tillgänglig via enhetens menyknapp. Nyare enheter har inte alltid en menyknapp, och för att täcka en större del av marknaden är det därför viktigt att implementera en action bar i sin applikation (Android 2013h).

Figur 4.6: Menyraden action bar (Android 2013g). Illustrerar grunddesignen av en action bar. Notera overflow-ikonen längst till höger.

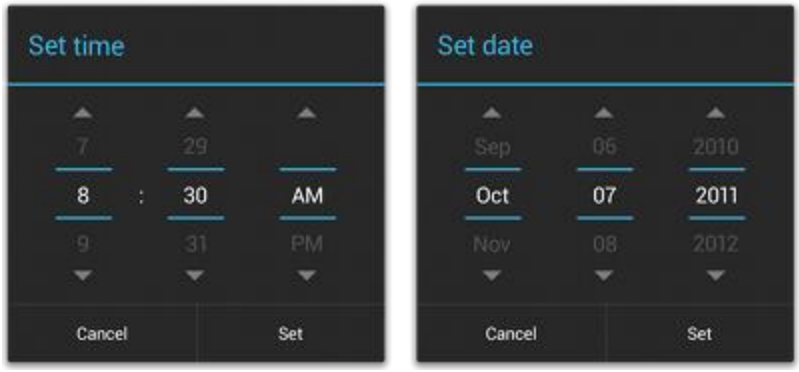
Med det nya konceptet action bar introducerades även en “uppknapp” som påminner något om den redan existerande bakåtknappen hos enheten. Skillnaden är att uppknappen navigerar uppåt i vyernas hierarki, medan bakåtknappen navigerar bakåt i historiken. I flera fall ger båda knapparna samma effekt (Android 2013g).

Action overflow är vad Android kallar den meny som ibland går att finna längst till höger i action bar genom overflow-ikonen. Ikonen föreställer tre vertikala punkter och dyker upp endast om det finns en action overflow att visa. I action overflow listas alla handlingar som inte får plats, eller som inte är viktiga nog, att representeras av ikoner i action bar (Android 2013g).

Innan Android 3.0 (API-nivå 11) fanns inte action bar, och istället använde man sig av en meny som dök upp då användaren tryckte på telefonens menyknapp. Vill man ändå implementera en action bar i en applikation byggd för tidigare Android-versioner finns ett fåtal alternativ. Android själva föreslår att en egen layout som liknar action bar ska skapas (Android 2013h). Ett annat alternativ är att implementera tredjepartsbiblioteket ActionBarSherlock som användare själva har utvecklat. Biblioteket är tänkt att använda en riktig action bar när mobiltelefonen stödjer det, och annars ersätta den med egna layouter. För att använda biblioteket måste projektet byggas på API-nivå 11. Ska applikationen stödja äldre Android-versioner måste utvecklarna därför vara extra noga med att alla metoder som används även finns i motsvarande äldre API-nivåer (ActionBarSherlock 2012).

4.8.2 Pickers

Pickers är Androids samlingsnamn för de redan uppbyggda dialogrutor som ger möjlighet att välja tid eller datum genom tre fält som går att räkna upp och ner, se figur 4.7. Android rekommenderar att man kallar på dessa pickers genom klassen `DialogFragment`, som inte introducerades förrän i API-nivå 11. Jobbar man med en tidigare API-nivå hänvisar de till klassen `DialogFragment` i sitt supportbibliotek för bakåtkompabilitet som ska stödja versioner från 1.6 och uppåt (Android 2013h). Användandet av `DialogFragment` är förstås bara en rekommendation och dialogrutor för val av tid- och datum infördes redan i API-nivå 1 (Android 2013i).



Figur 4.7: En `TimePicker` och `DatePicker`, som de kan se ut i Android 3.0 och uppåt (Android 2013h).

5. Projektdesign

Utifrån efterforskningen genomförd i kapitel *4 Omvärldsanalys* har ett antal beslut tagits för applikationens och projektets utformning. Dessa beslut redovisas och förklaras i följande avsnitt.

5.1 Arbetssätt

Enligt argument framförda i *4.6 Agil utveckling* kommer projektet genomföras enligt agila metoder, med nyckelelement ur Scrum. Fokus kommer ligga på sprintar och kontinuerlig testning. Dock utgår planeringen från "att göra"-kort snarare än användarberättelser. Detta på grund av den breda variationen på typ av arbete som ska utföras, som till exempel utforskning, skrivande och implementering. För att hålla alla "att göra"-kort under uppsikt används ett hjälpmedel kallat Trello (Trello 2013). Under varje sprint kommer ett antal funktioner stå i fokus, och målet är att de ska slutföras innan sprintens slut.

Detta projekt kommer genomföras av sex individer med vitt skilda scheman och arbetsbelastning, och det är därför inte alltid realistiskt att mötas varje dag enligt Scrums principer. Detta innebär en risk för nedbrytning i kommunikation, vilket kommer undvikas med hjälp av frekvent informell kommunikation av olika slag. Gruppen kommer även träffas ett antal gånger i veckan, minst ett av dessa möten är avsett för organisering, avslut och uppstart av en sprint, medan de andra är avsatt tid för att jobba tillsammans. På dessa så kallade arbetsmöten ska hela projektgruppen delta minst en gång i veckan, men möten där endast vissa medlemmar deltar kommer också inträffa. Självklart är målet att maximera tiden tillbringad tillsammans, men projektet genomförs med hjälp av agila processer blandat med individuellt arbete och distanskommunikation.

5.1.1 Versionshantering

För att stödja parallell utveckling och versionshantering har verktyget Git valt att användas. Git erbjuder bland annat tillgång till historik av filer samt spårning av författare och gjorda ändringar (Git 2013). En privat Git-server tillhörande en medlem ur projektgruppen användes för att lagra filerna.

5.2 Användande av plattform

Utifrån analysen av de olika plattformarna bekräftas valet av Android med stöd för version 2.2 och framåt (API-nivå 8) som plattform. Utifrån argumenten presenterade i *4.2.3 Webbaserad plattform* väljs nämligen webbaserade plattformar bort då de inte passar projektets krav väl, och eftersom en kvittoskannare följaktligen kräver tillgång till telefonens kamera. Statistiken i *4.2 Analys av möjliga plattformar* talar för att Android används på fler enheter än iOS. Det är dock svårt att säga att Android är populärast i projektets målgrupp studenter. Dessutom har ingen i projektgruppen tidigare erfarenhet eller hårdvara för utveckling av iOS-produkter, och iOS är därför ett oattraktivt val av plattform.

Android version 2.2 väljs för att nå ett så brett urval användare som möjligt, version 2.2 stöds nämligen av 97,6% av alla Android-enheter på dagens marknad (Android 2013b). Då Android finns i flera olika typer av enheter som mobiler, datorer och surfplattor skulle en anpassning av applikationens gränssnitt till alla dessa enheter komplicera designen. Utifrån denna ökade komplexitet togs beslutet att avgränsa utvecklingen av applikationen till enbart mobiltelefoner, eftersom mobilplattformen även är den mest relevanta för en applikation som bygger på ett frekvent användande av en kamera. I beslutet togs även hänsyn till att surfplattor har kamera, men att dessa är för klumpiga för att kunna användas i

exempelvis affären. Gruppen tror nämligen att det är vid en sådan plats som användaren kommer känna mest motivation till att använda applikationen.

5.3 Val av ramverk

Android har ett öppet förhållningssätt till utvecklare då Android SDK tillhandahåller bra verktyg för att enkelt och snabbt komma igång med utveckling av applikationer. Den kod som skrivits med hjälp av Android API ska också testas. Testing är ytterst viktigt och JUnit är ett väldigt bra verktyg som kommer att användas flitigt under projektets gång.

Ett annat testramverk som analyserades i omvärldsanalysen (se *4.3.4 Testing*) var Robotium, vilket är ett verktyg för att automatisera och simulera en användare. Detta gör Robotium till ett bra alternativ för att testa gränssnittet och kunna upptäcka brister, men också för att säkerställa att grafiska element, som exempelvis textsträngar, innehåller korrekt data. Under projektets gång kommer Robotium att användas för att testa gränssnitt.

En del av detta projekt kommer innefatta grafer, då ett mål är att kunna visa data på ett grafiskt och lättöverskådligt sätt. Biblioteket aChartEngine är enligt förstudien (se *4.1 Ramverk*) en bra lösning om man väljer att använda sig av ett bibliotek, eftersom viss anpassningsmöjlighet över grafernas utseende finns. Vid ett eventuellt användande av aChartEngine skulle det bli komplicerat att anpassa biblioteket till applikationens grafiska formgivning. Dessutom skulle det bli svårare att göra smarta tillägg till graferna. Därför har gruppen beslutat att försöka implementera ett eget anpassat grafbibliotek istället för att använda aChartEngine.

5.4 Användning av optisk teckenigenkänning

Det beslutades utifrån *4.4 Optisk teckenigenkänning* att undersöka Tesseract som en möjlig teknik för teckenigenkänning eftersom Google stödjer projektet, men också eftersom det har utvecklats under en lång tid. Andra fördelar med Tesseract är att det används i flera olika projekt och att det finns stöd i form av wrappers och dokumentation tillgängligt.

ABBYYs lösning för OCR valdes bort då den är en kommersiell produkt och ansågs ha bristfällande dokumentation och möjligheter till kodgranskning. Detta innebär att deras teknik blir svår att utforska och eventuellt förbättra, samt att den blir svår att anpassa till projektets applikation.

5.4.1 Tesseract på mobil plattform

Enligt ovan nämnda argument kommer Tesseract att integreras i applikationen. Gruppen vill även testa att applicera Tesseract direkt under den mobila plattformen för att undersöka hur väl resultatet blir och hur lång tid det tar att beräkna resultatet under mobilhårdvaran. Utifrån resultaten kan det bli aktuellt att testa en serverlösning för att effektivisera beräkningen.

5.5 Metod för bildbehandling

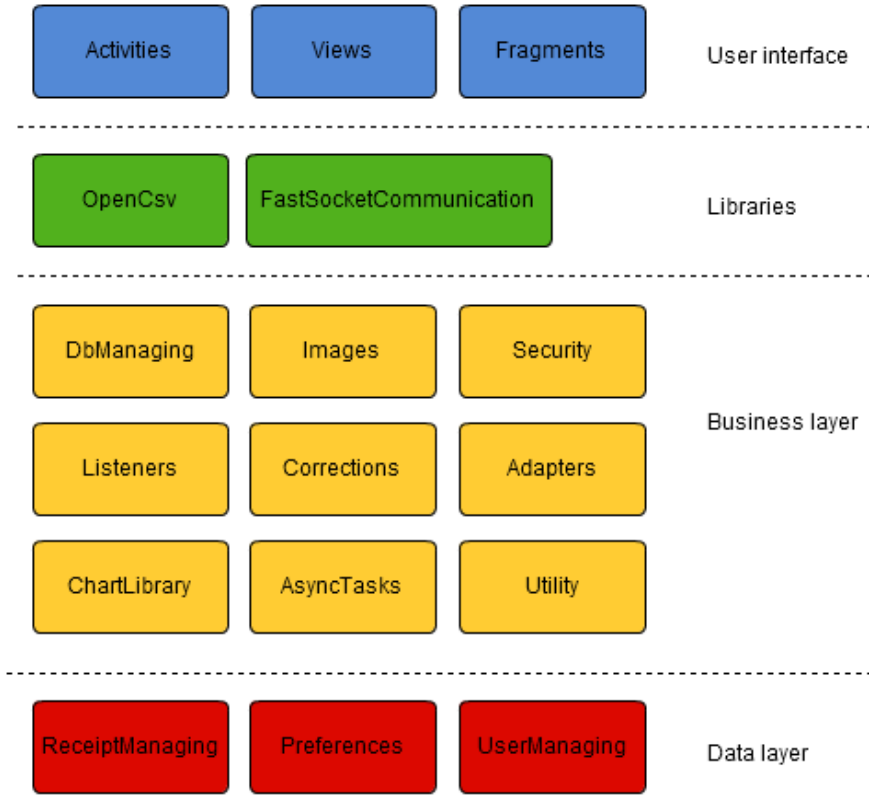
Att använda Otsus metod för binärisering kommer resultera i dåliga resultat då bilderna som hanteras innehåller gradienta övergångar från vitt till svart i bildens kanter (se *4.5 Bildbehandling*). Svarta hörn uppenbaras då metoden inte tar hänsyn till dessa övergångar, vilket leder till ett stort antal fall då resultatet innehåller fler svarta pixlar än nödvändigt. Detta innebär mer data som den optiska

teckenigenkänningen behöver analysera och leder till långa beräkningstider. För att slippa lägga ner tid på att optimera en metod som inte ger bra resultat valdes metoden bort.

Medan Niblacks metod gav bra resultat i de områden av bilden som innehöll text, innehöll de vita områdena mycket brus. På grund av detta valdes Niblacks metod bort.

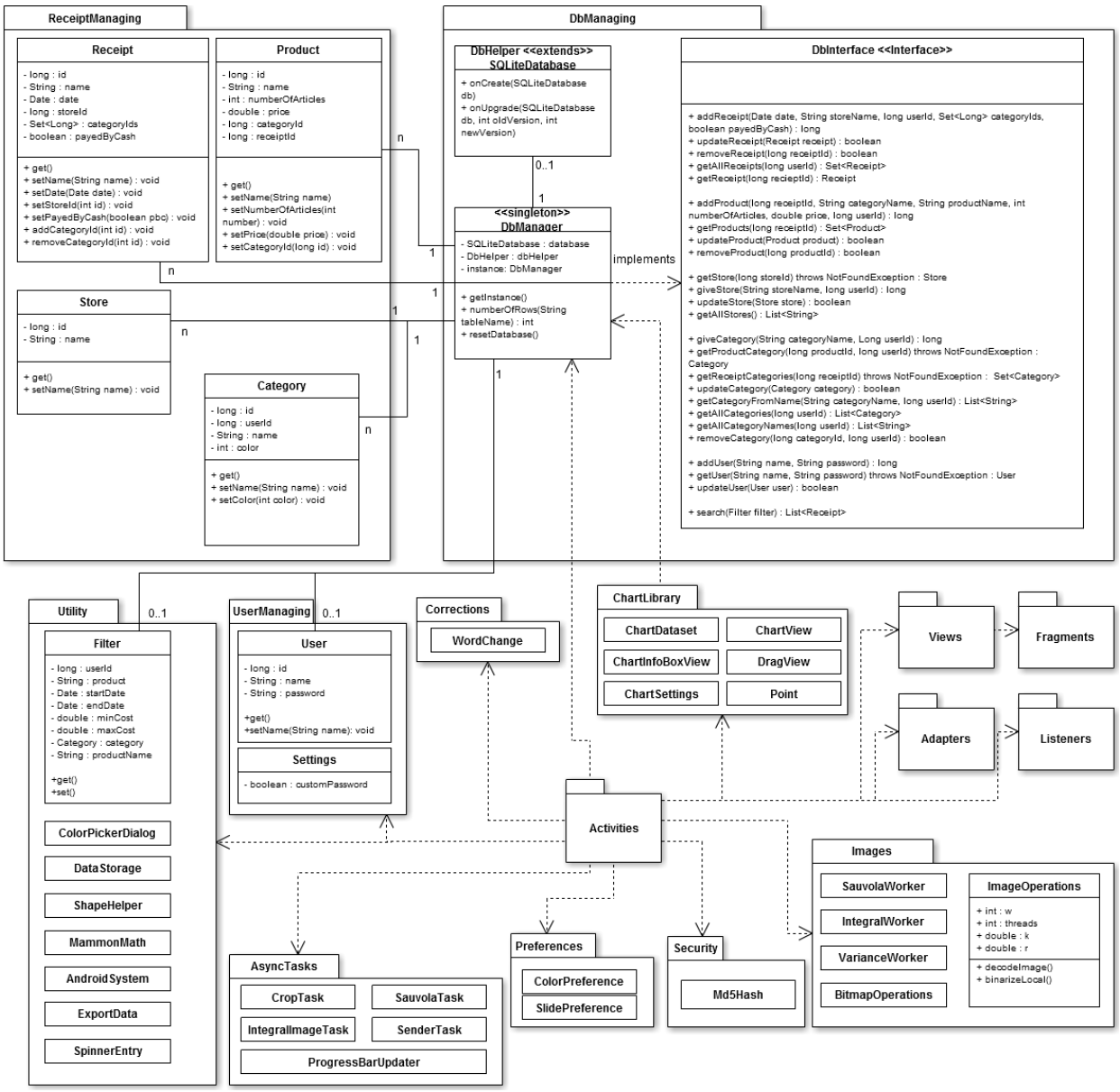
Sauvolas metod för binärisering är den metod som generellt verkar ge bäst resultat vid användandet av bilder med varierande bakgrund, vilket kan ses i avsnitt *4.5 Bildbehandling*. Algoritmen tar dock alldeles för lång tid för att kunna användas i en applikation på en mobiltelefon, därför är det intressant att optimera denna metod och anpassa den till mobila enheter.

5.6 Systemets arkitektur



Figur 5.1: Applikationens arkitektur uppdelad i fyra grundläggande funktionalitetslager.

I figur 5.1 finns applikationens övergripande struktur, representerat i form av olika lager av funktionalitet. Användargränssnittet är mest abstrakt, och visas därför längst upp, medan den konkreta datahanteringen finns längst ner. Innehållet reflekterar till stor del klassdiagrammet som finns i figur 5.2, då de olika lagren innehåller paket därifrån. Användargränssnittet är representerat med hjälp av `Activities` då detta är vad Android-applikationer använder sig av för att kontrollera gränssnitt.



Figur 5.2: Det slutgiltiga diagrammet över klassuppbyggnaden i applikationen. Linjer mellan de olika paketen och klasserna illustrerar deras beroende av varandra i applikationen.

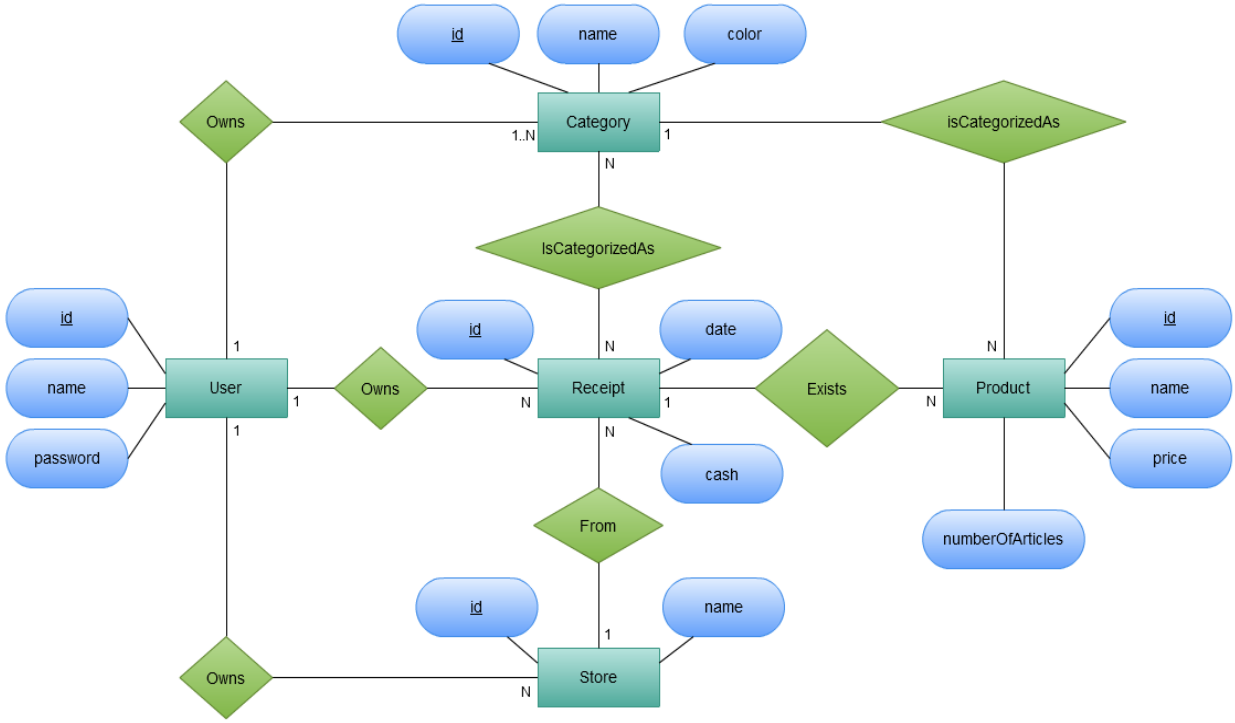
I figur 5.2 ses programmets klassdiagram. Applikationen är uppbyggd på ett modulärt sätt, då detta minskar beroendet mellan klasser och förenklar framtida expansion. Klasstrukturen har påverkats av hur Android-applikationer är uppbyggda, vilket kan ses på relationerna mellan paketet `Activities` och de andra paketen i diagrammet, men även av hur databasen är utformad (se *5.7 Databasutformning*). Den mest centrala klassen är `DbManager`, då den sköter all kommunikation mot databasen.

5.7 Utformning av databas

Databasen är en central punkt i applikationen då projektet bygger på lagring och hämtning av kvittodata (se *3 Domänmodell*). Det är därför viktigt att planera väl hur databasen ska utformas, samt hur och var den ska byggas. Utifrån analysen av olika koncept av databashantering (se *4.7 Databashantering*) kommer databasen byggas internt i Android med hjälp av SQLite. Detta innebär nämligen att det är enkelt att bygga upp koden för databasen och att användaren inte behöver internetuppkoppling för att nå sina kvitton.

För att inte låsa applikationen till en intern lösning för att upprätthålla databasen har uppbyggnaden av databasen anpassats till att kunna hantera flera användare med tillhörande kvitton. Då varje enhet teoretiskt sett endast har en användare finns endast en applikationsanvändare som har tillgång till sin egen interna databas. Därmed är möjligheten att ha flera användare överflödig vid implementation av intern databas. Denna uppbyggnad motiveras dock av att det ska vara möjligt att med lätthet kunna extrahera databasen till en extern server i framtiden.

5.7.1 ER-diagram



Figur 5.3: ER-diagram över databasen kopplad till applikationen

I figur 5.3 finns applikationens ER-diagram, en visualisering av databasens struktur. De olika enheterna i ER-diagrammet förklaras nedan.

- User: En användare som äger flera kategorier, kvitton, och affärer. Anges av ett användarnamn och ett lösenord.
- Receipt: Varje kvitto har flera olika kategoristämplar där varje kategori tillhör en användare. Följaktligen har ett kvitto en affär som ägs av en användare. Slutligen har ett kvitto flera

produkter kopplade till sig, samt information om när kvittot skapades och om kunden betalat kontant.

- Product: En produkt tillhör en kategori. Varje produkt innehåller även information om namnet på produkten, och hur många artiklar av produkten som existerar på kvittot som produkten tillhör. Där ingår också information om priset på produkten.
- Category: En produkt kan ha en kategori, medan ett kvitto kan ha flera.
- Store: En affär finns på varje kvitto och tillhör en viss användare.

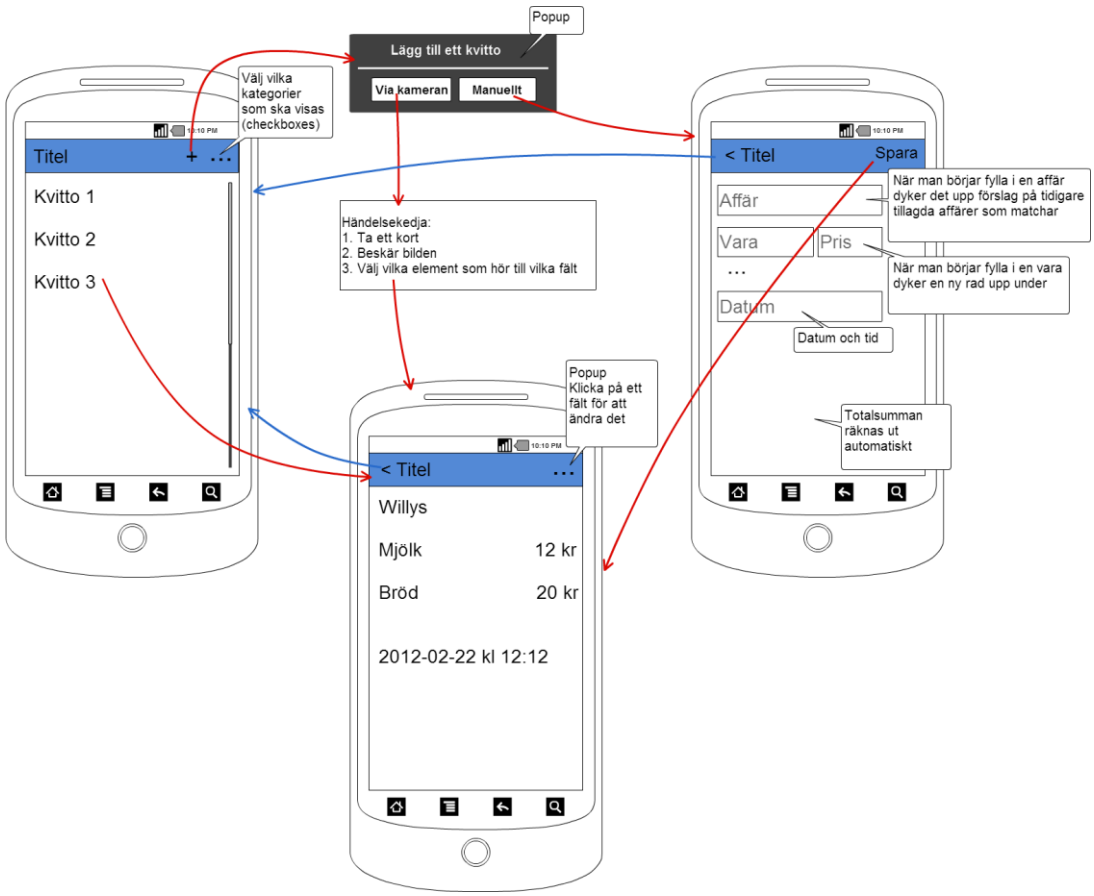
En användare är en central punkt ur både applikations- och databassynvinkel, eftersom varje användare kommer ha en privat uppsättning kvitton med tillhörande produkter. Följaktligen har även varje användare en egen kategori- och affärsuppsättning att tillgå. Denna konstruktion är till för att förhindra att olika användare använder samma enhetsdata från databasen, eftersom det skulle skapa problem om en användare exempelvis vill ändra stavningen på en affär och därmed påverkar alla kvitton som refererar till den affären, även för alla andra användare. Utöver detta krävs både en användare och en kategori för att hämta ett kategoriobjekt, eftersom en kategori ihop med en användare är unik i databasen.

5.7.2 Val av referensnycklar

Det beslutades att entiteterna `Category` och `Store` skulle använda ett id som referensnyckel istället för kategorinamn respektive affärsnamn (se figur 5.3). Beslutet grundar sig i att flera olika användare ska kunna ha samma affärer och kategorier tillagda i databasen utan att de delar samma uppsättning. Även referenserna från ett kvitto gynnas av detta då det hade krävts en uppdatering av referenserna på alla relevanta kvitton om affärsnamn respektive kategorinamn hade varit nycklar. Med beslutet att använda ett id som referensnyckel krävs endast en uppdatering i tabellen för affär eller kategori, vilket innebär en förbättring av prestanda för databasen.

5.8 Användargränssnitt

Användargränssnittet är viktigt då det utgör användarens möjlighet att påverka vad som händer och dess inblick i applikationen. Här är det grafiska användargränssnittet vad användaren kommer att se och interagera med. Det utgör en särskilt stor del när det gäller mjukvara till en enhet med touchskärm, då man nästan uteslutande interagerar direkt med gränssnittet. I figur 5.5 ses en tidig skiss över det planerade användargränssnittet med viss funktionalitet.



Figur 5.4: En tidig skiss av applikationens användargränssnitt där de tre huvudvyerna och förflyttningen mellan dem visas.

Androids riktlinjer gällande gränssnittsdesign (Android 2013g) ska till så stor del som möjligt följas. Där finns bland annat information om hur navigering ska utformas och hur och när man ska ge bekräftelser till användaren. Till exempel ska tilläggandet av ett nytt kvitto bekräftas, och vid borttagande av ett kvitto ska en dialogruta som kräver interaktion från användaren visas.

Ett av projektets mål är att gränssnittet ska vara intuitivt och självförklarande då applikationen vill hållas så enkel som möjligt, och för att användaren ska slippa en genomgång i hur applikationen ska användas. Om applikationen har ett förvirrande gränssnitt är det möjligt att användaren drar sig för att använda den, då det finns en risk att användaren upplever stressframkallande moment i applikationen.

Moraveji och Soesanto (2012) beskriver i en artikel hur man kan utforma gränssnittet för att undvika de stressframkallande momenten. Bland annat skriver de att man ska ge feedback vid användarinteraktion, vilket även är i linje med Androids riktlinjer. En annan viktig punkt är att användaren lätt kan känna sig stressad eller irriterad om ingen återkoppling ges då det är nödvändigt att vänta på att en viss process blir slutförd. I projektets applikation kan ett tillämpbart fall vara när användaren väntar på att OCR-algoritmerna körs på bilden av ett kvitto. För att undvika irritation bör man då ha en förloppsindikator som visar på hur stor del som är klar och hur mycket som återstår.

5.8.1 Designval baserade i den inledande enkäten

Vid analys av enkäten (se *2.2 Analys av kravundersökning*) observerades att det viktigaste för de svarande var att applikationen ska vara smidig att använda. Med detta som riktlinje valdes en enkel och intuitiv design av gränssnittet där den aktivitet som ska utföras, till exempel inläggning av ett nytt kvitto, ska gå att genomföra med minimal ansträngning från användarens sida.

5.8.2 Design av action bar

Menyraden action bar kommer att implementeras i applikationen för att stödja nyare enheter (se *4.8.1 Action bar*). Då applikationen bygger på API-nivå 8, där inget stöd för action bar finns, kommer menyraden att ersättas med efterliknande layouter. Biblioteket ActionBarSherlock kommer inte att användas då det skulle vara omständigt att bygga projektet på API-nivå 11, och därmed behöva se till att alla metoder som används inte tillkommit senare än i API-nivå 8.

5.8.3 Tids- och datumväljare

Androids färdiga tids- och datumväljare kommer inte att användas då tidsväljaren endast stödjer val av timmar, minuter och AM/PM, men inte sekunder (Android 2013i). Istället kommer en ny tidsväljare på formen timmar-minuter-sekunder att byggas från grunden. Datumväljaren skulle kunna återanvändas då alla nödvändiga fält finns. Dock visas den på formen månad-dag-år, där månaden står utskriven med de första tre bokstäverna ur dess engelska namn (se figur 4.4). För att slippa språkdifferenser och för att bättre efterlikna datumets form på kvittot, kommer även datumväljaren att byggas från grunden på formen år-månad-dag, där månaden skrivs ut med siffror.

6. Genomförande

Detta kapitel beskriver till stor del den del av projektet då implementeringen skedde. Aktiviteter som utfördes i samband med detta, till exempel testning och användartester inkluderas också i kapitlet. Även problem, lösningar och resultat av intresse inkluderas, med fokus på vad som gjort detta projekt unikt.

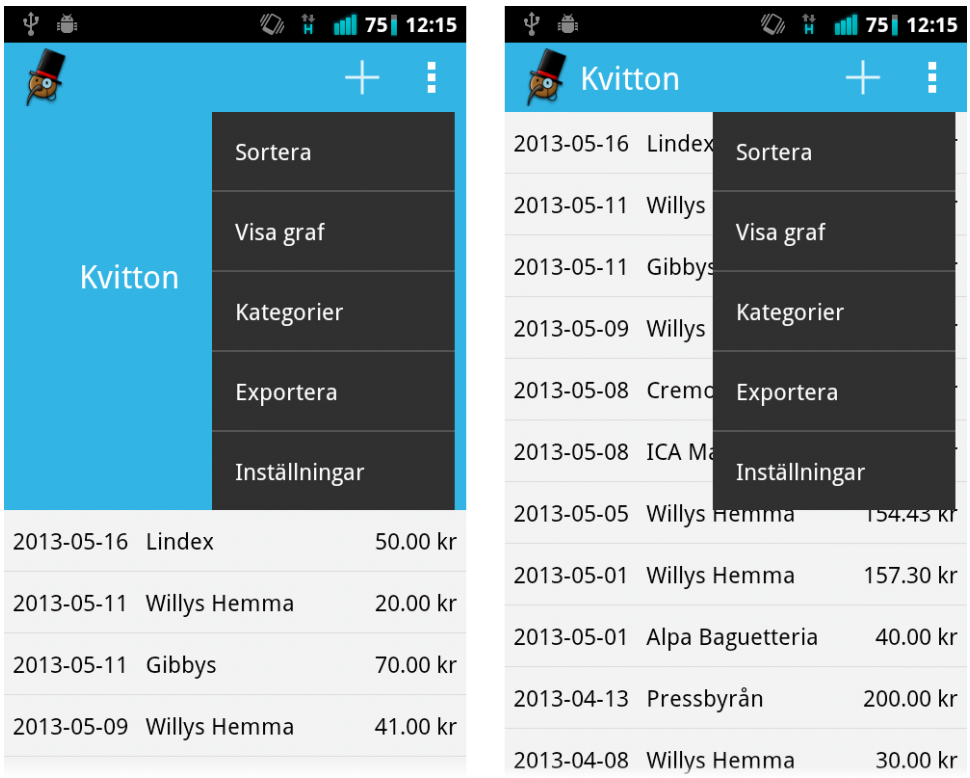
6.1 Användargränssnitt

Den största utmaningen i utformningen och implementationen av användargränssnittet var att anpassa det till Android-versioner både före och efter version 3.0, eftersom denna introducerade många förändringar (se *4.8 Designstandarder för Android*). Gränssnittet är utformat för att passa nyare Android-versioner, samtidigt som det är byggt för att stödja äldre. Många annars självklara delar är därför byggda från grunden.

6.1.1 Implementation av action bar

Menyraden action bar implementerades genom layouter som skulle efterlikna dess utseende. Till en början lades en unik action bar in i varje aktivitets layoutfil, då texten och ikonerna i fältet skiljer sig mellan aktiviteterna. Detta gjorde dock alla layoutfiler onödigt långa, och ett beslut togs att extrahera action bar till en egen layoutfil som därefter inkluderades i alla aktiviteters layoutfiler. För att de olika ikonerna i action bar fortfarande skulle finnas tillgängliga för klasserna lades alla ikoner in i action bars layoutfil i ett gömt tillstånd. Därefter lades några rader kod till i varje aktivitet för att ta fram de ikoner som skulle användas i just den vyn, och för att ändra texten beroende på aktivitet.

Menyn action overflow (se *4.8.1 Action bar*) ingick inte i layoutfilen för action bar utan implementerades istället som en separat listvy i layoutfilerna för de aktiviteter där denna var önskad. Hade man placerat listvyn i action bar hade den nämligen orsakat att action bar expanderat och därmed puttat undan andra element på skärmen var gång listan visats, se figur 6.1.



Figur 6.1: Till vänster i bild A visas hur applikationen hade betett sig om menyn action overflow inkluderats i layoutfilen för action bar. I bild B till höger syns hur det fungerar när menyn istället ligger separat i aktivitetens layoutfil.

Vidare tilläts åtgång till overflow-menyn genom en lyssnare på overflow-ikonen som växlade menyns synlighet. Dessutom lades lyssnare in på enhetens meny- och bakåtknapp, så att användare med äldre Android-versioner, som inte är vana vid action bar och istället förväntar sig att få upp menyn först när de trycker på menyknappen, lättare skulle kunna sätta sig in i hur applikationen fungerar.

6.1.2 Uppbyggnaden av tids- och datumväljare

Tids- och datumväljarna byggdes upp från grunden för att deras utseende skulle kunna anpassas efter applikationen. I stort sett består de av en `AlertDialog` med anpassat innehåll. En `AlertDialog` är en dialogruta med titel, innehåll, och en till tre knappar. I detta fall användes två knappar för “OK” och “Avbryt”. Layouten för innehållet strukturerades upp i en layoutfil; båda väljarna hade tre textfält för siffror med tillhörande plus- och minusknappar för att öka eller minska värdet i respektive fält, se figur 7.12.

Som rekommenderat på Androids hemsida användes klasser som ärvde från `DialogFragment` för att kalla på dialogrutorna (se *4.8.2 Pickers*). I dessa klasser byggs dialogrutorna upp, och värdena i nummerfälten fylls i utifrån tidigare valda värden. I aktiviteten för redigering av kvitto är det till exempel den tid och det datum som syns i aktiviteten som kommer att fyllas i i dialogrutan. Liksom i alla andra

vyer i applikationen läggs en nolla till framför siffran om den är lägre än tio, för att behålla tids- eller datumformateringen.

På plus- och minusknapparna sattes en `OnTouchListener` istället för det mer normala `OnClickListener`. Detta gjordes för att användaren ska ha möjlighet att hålla in knappen för att snabbare öka eller minska värdena, istället för att behöva klicka på den för varje ändring. Vanligtvis hade problemet lösts genom att lyssna på när knappen trycks ner och när den släpps upp igen, och däremellan använda en timer för att med jämna mellanrum förändra värdet. Vid försök av implementation av detta fungerade det dock inte som tänkt, och istället gjordes en egen lösning där en booleansk variabel för varje knapp reglerar huruvida något ska hända när knappen trycks ner. I metoden som knappen sedan anropar förändras först värdet, och en kontroll av dess giltighet görs. Därefter skapas en timer i en ny tråd för att inte avbryta huvudtråden och därmed användarens möjlighet att göra något. När timern är klar ändras den booleanska variabeln så att metoden återigen går att anropa vid interaktion med knappen.

Utöver lyssnare på knapparna sattes även lyssnare på textfälten då dessa går att redigera direkt. En `OnFocusChangeListener` används för att kontrollera värdena när användaren byter fokus från ett fält till något annat. För att kunna påverka värden utan att användaren först måste byta fokus används `TextWatcher` som är en lyssnare som reagerar på förändring av text i textfält. Även denna lyssnare kontrollerar giltigheten på talen och ändrar dem direkt om de överstiger sitt maximala värde, för att förhindra att strängen som skickas tillbaka till aktiviteten är ogiltig. Av samma anledning sätts fältets värde till default om fältet är tomt. Detta görs dock bara i bakgrunden för att användaren ska kunna använda fältet som vanligt, utan att text dyker upp så fort denne suddar.

6.1.3 Ikoner

För att undvika möjliga upphovsrättsproblem användes i största möjliga utsträckning standardikoner från Android, till exempel overflow-ikonen. I de fall då ikonerna designades av projektgruppen följdes de riktlinjer som satts upp av Android (se *4.8 Designstandarder för Android*).

6.1.4 Färger

Färgvalen som gjordes baserades på Androids standarder. Den blå färgen som genomsyrar applikationen kommer från en av Androids framtagna färguppsättningar (Android 2013h). I action bar används standardnyansen av blått, och när en knapp i action bar trycks ned används en mörkare nyans. Samma mörka nyans används i alla dialogrutor med nyare utseende. Detta utseende är menat att efterlikna temat Holo Light som introducerades i Android 3.0 (Android 2013h). Tillsammans med detta tema introducerades även nya färger, bland annat en ljusgrå bakgrundsfärg som ersatte den tidigare vita färgen. Även applikationens bakgrundsfärg ersattes med denna nya färg för att ge ett modernare utseende. I listvyerna användes selectors för att bestämma radernas utseende då man klickar på dem, då de är valda och då ingenting hänt.

6.1.5 Gränssnittstest

Vid testningen av användargränssnittet användes ramverket Robotium (se *4.3.3 Testning*). Gränssnittstestningen genomfördes i två etapper: vid slutet av den första och den andra iterationen. Troligt användarbeteende emulerades för de vanligaste funktionerna, till exempel för skapande av kvitton.

Som ett resultat av de automatiserade testerna kunde ett antal defekter i applikationen upptäckas och korrigeras.

Ett exempel på en upptäckt defekt är att när priser skulle visas som en text i applikationen uppstod problem med att de avrundades fel i vissa fall. Om till exempel två produkter med kostnaderna 19,90 kr samt 20,91 kr lades in blev totalpriset 40,80 kr istället för det korrekta 40,81 kr. Eftersom flyttal kan vara oändligt långa men bara har 32 bitars minne att sparas på måste de ofta avrundas, vilket leder till att det kan bli fel (Goldberg 1991). Klassen `BigDecimal` rekommenderas att använda när man behöver exakta tal (Oracle 2013a), och ansågs därför vara bra att använda i applikationen istället för `float` eller `double`.

6.2 Implementering av databas

Eftersom det beslutats att en intern databas skulle användas för lagring av information om kvitton (se *5.7 Databasutformning*) implementerades databasen med hjälp av SQLite, som det finns inbyggt stöd för i Android (Android 2013f). I klassen `SQLiteDatabase` i Android finns metoder för bland annat insättning, radering och uppdatering av data i databasen vilket gör att en SQLite-databas är väldigt lättanvänd, även för utvecklare som inte har så stor erfarenhet av databaser.

6.2.1 Koppling mellan applikation och databas

Kopplingen mellan applikationen och databasen sker genom två klasser, `DbManager` och `DbHelper`, se figur 5.3. I `DbManager` sker kopplingen mot databasen ut mot andra klasser genom publika metoder. Metoderna i sin tur använder sig av `DbHelper` för att öppna och stänga anslutningen till databasen samt för att sköta anropen mot den. Detta underlättar vid en eventuell överflyttning till en extern databas, då enbart metoderna i `DbManager` skulle behöva uppdateras eftersom databaskopplingen är begränsad till att enbart ske i den klassen.

I implementeringen öppnas och stängs kopplingen till databasen i varje metod i `DbManager`. Detta har fördelen att utvecklaren inte behöver tänka på om kopplingen är öppen när den ska vara det: allting sker automatiskt i bakgrunden. Det blir dock en nackdel när flera metodanrop till `DbManager` sker i följd, vilket leder till att databasen öppnas och stängs fler gånger än den skulle gjort om utvecklaren själv skulle hanterat databaskopplingen.

6.2.2 Testning av databasgränssnitt

Att testa databasanropen, som är gränssnittet mellan utvecklare och databasen, var viktigt. Därför användes JUnit med vilket tester för varje enskild metod i `DbManager` utfördes. För att underlätta testfallen för varje metod initierades databasen med testdata: några användare med tillhörande kvitton samt produkter skapades. Metoderna testades sedan genom att verifiera data från databasen mot förväntad data.

Med ovan beskrivna teknik hittades och åtgärdades flera utvecklingsbuggar. Brister i databaslogiken upptäcktes även, bland annat att fler metoder behövdes för att utföra uppgifter som inte tänktes på när gränssnittet för databasen byggdes upp. Till exempel var utvecklaren till en början själv tvungen att först kolla upp om en viss kategori existerade i databasen innan den lades till. Med hjälp av noggrann testning av metoderna i `DbManager` upptäcktes att det skulle bli smidigare för utvecklaren om man i respektive

metod för insättning i databasen även kollade om det objekt man försökte sätta in redan fanns. Detta minskade därmed risken för buggar då ansvaret flyttades från utvecklaren till metoder i bakgrunden.

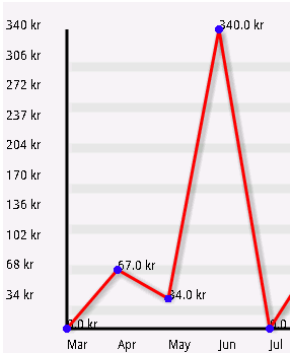
6.3 Grafbibliotek

Grafiska element i Android-applikationer är uppbyggda med olika programmerade vyer som är Java-klasser som ärver av en grafisk basklass kallad `View`. Denna klass gör det möjligt att skapa visuella element och lyssna till användarens anrop (Android 2013j).

Ett delmål i projektet (se *2.3.2 Andra prioritet*) var att implementera grafer över inlagda kvitton eftersom grafer bidrar till en bättre överblick av de inlagda kvitton som existerar i databasen. Användaren kan genom dem tydligt se hur mycket det har handlats för under en viss period.

6.3.1 Klassen ChartView

För att skapa en grafvy implementerades en klass kallad `ChartView`. `ChartView` ansvarar för att rita ut en graf med staplar eller en linje. Dessutom ansvarar den för att rita ut olika tillbehör som axlar, etiketter, och grafiska detaljer i form av bakgrundslinjer och punkter för varje värde på graflinjen, vilket gör det lättare att läsa av grafen. Med metoder från klassen `Canvas` (Android 2013k) ritades alla synliga grafiska detaljer för grafvyn ut, ett exempel på detta finns i figur 6.2.



Figur 6.2: Ett exempel på hur klassen ChartView ihop med färginställningar från ChartSettings ritar ut grafiska detaljer ihop med en graf.

Koordinater i Android består av par i form av flyttal som anger bredd och höjd (Android 2013j). Dessa koordinater utgår från punkten (0,0) som befinner sig längst upp i vänstra hörnet av skärmen. Därför skalas varje punkt om för anpassas till den ritade grafen. Utvecklare av en graf behöver alltså inte bekymra sig om vad skärmkoordinaterna relaterat till en punkt ska vara, punkterna i graflinjen kan byggas utifrån grafens origo och ändå visualiseras korrekt.

6.3.2 Inställningsbara parametrar

När många separata grafikdetaljer som kräver olika färger, storlekar och effekter ritas ut måste många variabler fastslås. Istället för att använda statiska färger i dessa variabler används värden från en separat statisk klass kallad `ChartSettings`. Denna klass kan modifieras via en inställningsvy, vilket gör att användaren själv kan ställa in hur vyn ska se ut.

För att kunna rita en graf måste kvittodata hämtas från databasen. Detta sker genom att användaren först ställer in olika parametrar som önskad tidsperiod genom att ange två datum, en graftyp och i vilken detaljnivå grafen ska uträttas under: dag, vecka, månad eller år.

6.3.3 DragView

För att öka funktionaliteten i grafvyn implementerades en klass som ärver från `ChartView` kallad `DragView`. Denna klass ansvarar endast för att reagera på användarens interaktion med grafen genom att lyssna på hur många fingrar som är på skärmen. Om exempelvis två fingrar befinner sig på skärmen räknas avståndet mellan dessa två ut. Detta avstånd används sedan för att skala om grafen. Om istället ett finger är på skärmen och under rörelse förflyttas grafvyn i sidled. Denna sidledsförflyttning stannar inte direkt när användaren lyfter fingret, utan fortsätter ett litet tag efteråt beroende på hur snabb rörelsen var. Detta sker med en med hjälp av en tråd som skapas, och ger användaren en känsla av friktion vid förflyttningen.

6.4 Inställningar i applikationen

En mobilapplikation har ofta många olika inställningsbara alternativ, exempelvis för utseende, förändring av lösenord eller inställning av standardvärden. I projektet implementeras dessa inställningar genom användning av Androids inbyggda paket `android.preference`. Paketet tillhandahåller klasser för att skapa en grafisk inställningsvy med olika former av inmatningar, som exempelvis textinmatning eller kryssrutor. Det tillhandahåller också metoder och procedurer för att hantera och lagra inställningar (Android 2013m).

Att lagra inställningar sker genom att använda Androids `SharedPreferences` som är en gränssnittsklass vars instans delas mellan alla användare av applikationen (Android 2013n). På detta sätt kan data delas mellan olika aktiviteter och tillstånd, samt mellan olika sessioner för applikationen (Android 2013o). När användaren modifierar inställningsvyn lagras inställningarna automatiskt i `SharedPreferences`. Detta sker med en fördefinierad nyckel i inställningsvyns XML-kod som datan kopplas till. Via nyckeln är det sedan möjligt att komma åt datan för den nuvarande inställningen.

Problem uppstod på grund av att `preference`-paketet inte har stöd för alla olika inmatningar som projektgruppen behövde, som exempelvis ett skjutreglage där man kan ange olika värden genom att justera ett reglage horisontellt. Utöver detta instrument behövde gruppen utökade detaljer i inställningsvyn som att välja en färg på ett intuitivt sätt genom en färgväljare. På grund av detta implementerades Preference-klasser som kunde modifieras efter behov. Klasserna refereras sedan till från dem layoutfiler som bygger upp inställningsvyn.

6.5 Låsmekanism

Enkätresultatet från avsnitt *2.2 Analys av kravundersökning* underströk att säkerhet i applikationen var önskvärt. Säkerheten ställdes också som ett funktionellt krav (se *2.3.2 Andra prioritet*), där lösenordsåtkomst till applikationen skulle krävas och känslig data skulle krypteras.

Kraven uppfylldes med hjälp av en dialog där användaren anger ett lösenord bestående av siffror. Dialogen används vid autentisering av användaren när applikationen har gått in i ett låsningstillstånd.

Tack vare detta behöver inte Androids inbyggda tangentbord användas, något som skulle fördröjt identifieringen och eventuellt skapa ett irritationsmoment hos användaren.

6.5.1 Låsningsprocess

När applikationen startas första gången sker en förfrågan om lösenord ska aktiveras. Detta görs för att ge användaren möjlighet att själv välja om den vill använda funktionen. Följaktligen får användaren reda på att möjlighet till lösenordsskydd av applikationen finns. Valet kan sedan ändras via applikationens inställningar.

I första implementationen av låsningsmekanismen aktiverades låset efter att användaren hade varit inaktiv under ett bestämt tidsintervall. Låsets status bestämdes genom en variabel i `SharedPreferences`, vars värde avlästes vid uppstart av alla aktiviteter. Proceduren fungerade därför inte som tänkt då lösenordsskyddet dök upp först när förflyttning till en annan vy skedde.

I andra versionen av låsningsmekanismen aktiveras autentiseringsdialogen när applikationen går från att vara en bakgrundsprocess till att vara synlig. Detta har fördelen att användaren inte är bunden till en tidsram. Implementationen skedde genom att skapa en klass vars uppgift är att kontrollera om lösenord är aktiverat och status för applikationsprocessen. Statusen kontrolleras med hjälp av klassen `ActivityManager` som är medveten om vilken process som är synlig i mobilen. Autentiseringsdialogen visas beroende på status, i detta fall när applikationen startar eller blir startad efter en annan applikation.

6.5.2 Kryptering

Lösenordet som användaren anger i applikationen krypteras genom en MD5-algoritm. Android tillhandahåller en klass kallad `MessageDigest` genom paketet `java.security` (Android 2013l) som krypterar lösenord till en fördefinierad längd i form av ett hashvärde (Oracle 2013b). Detta värde transformeras i applikationen till en sträng för att kunna kopplas till en användare i databasen.

6.6 Bakgrundsprocesser i Android

Många gånger behövs trådar till tidskrävande operationer. Dessa operationer vill man inte utföra i huvudtråden eftersom det skulle försämra applikationens användarupplevelse; användarna skulle uppfatta applikationens gränssnitt som icke-responsivt.

Android erbjuder ett bra verktyg i form av en klass kallad `AsyncTask` för att hantera bakgrundstrådar som efter avslutad operation kan kommunicera tillbaka till huvudtråden (Android 2013p). För att använda sig av detta skapas en klass som ärver klassen `AsyncTask`. Denna klass innehåller metoder för att utföra bakgrundsoperationer. `AsyncTask` gör det även möjligt att hantera data innan och efter att bakgrundsoperationen utförts.

6.6.1 Användning av AsyncTask och händelsedialoger

Det påpekas i avsnitt *5.8.2 Utformning av användargränssnittet* att användaren upplever stress om inte återkoppling ges under bakgrundsoperationer. Händelsedialoger används därför för att användaren ska veta att det sker något krävande i applikationen. För att inte hindra användarens interaktionsmöjligheter

kan klassen `AsyncTask` utnyttjas för att hantera krävande beräkningar i en separat tråd samt uppdatera händelsedialogen under beräkningens förlopp.

6.6.2 Åtkomst av kvitton

Under Android version 2.2 är det tillåtet att använda nätverksoperationer under huvudtråden. Denna tillåtelse har dock tagits bort under Android version 3.0 och högre (Android 2013q). Skulle nätverksoperationer ändå exekvera under huvudtråden under exempelvis Android version 3.0 kommer ett undantag kastas.

Applikationen använder under nuvarande implementation en intern databas i form av SQLite vilket medför att ingen nätverksåtkomst behövs (se *4.7.1 Intern databashantering*). Eftersom det kan finnas många kvitton i databasen som skulle kunna ta lång tid att hämta, och då det i framtiden skulle kunna vara intressant att använda en extern databas, hämtas kvitton som en bakgrundsoperation med hjälp av klassen `AsyncTask`.

6.7 Förbehandling inför OCR

Innan bilden tolkas av textigenkänningsalgoritmen behandlas den för att få ett så bra resultat som möjligt. Följande avsnitt beskriver dessa förbehandlingar.

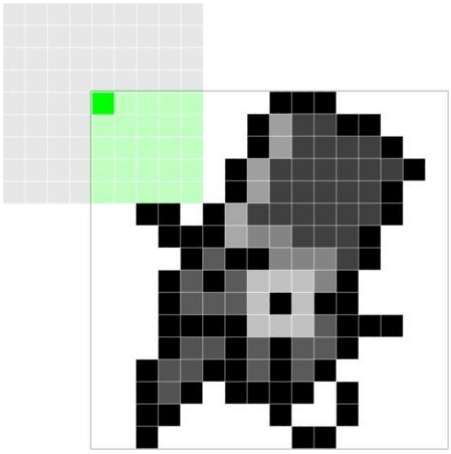
6.7.1 Bildbeskärning

För att enkelt kunna filtrera bort den information som finns i bilden men som inte ska konverteras till text implementerades en vy där användaren kan beskära den tagna bilden. En rektangulär beskäring där användaren markerar det område i bilden som ska behållas med hjälp av pekskärmen användes. Användaren kan antingen använda två fingrar i taget för att välja två motstående hörn eller ett finger i taget för att flytta sidorna i den redan existerande rektangeln. Efter att önskad del av bilden har markerats, sparar användaren bilden och binäriseringsprocessen påbörjas.

6.7.2 Optimering av Sauvolas metod för binärisering

För att ha mer kontroll över OCR-processen förbehandlas bilden internt i applikationen innan den skickas till Tesseract. För att separera text och bakgrund är det lättare att hantera en bild i gråskala, eller ännu hellre en binäriserad bild.

Den största nackdelen med att använda Sauvolas metod över Otsus är de mångfaldigt tyngre beräkningarna. Det som har störst påverkan på Sauvolas beräkningstid är storleken på det fönster som valts att hämta medelvärde och standardavvikelse från. Om fönstret exempelvis är 16 pixlar brett och högt kommer Sauvola att för varje pixel beräkna ett lokalt gränsvärde utifrån ett fönster av storleken 16*16 pixlar, det vill säga 256 pixlar. Om fönstret löper utanför bilden, blir pixelarean mindre och färre än 256 pixlar bestämmer gränsvärdet, vilket illustreras i figur 6.3.



Figur 6.3: En illustration över fönstret för beräkning av medelvärde och standardavvikelse med fönsterstorlek 9*9 pixlar. Fönstret centreras kring den aktuella pixeln. Då pixeln ligger i bildens hörn blir i detta exempel pixelarean mindre än om hela fönstret fått plats i bilden.

```
for imageX : 0 -> image.width
  for imageY : 0 -> image.height {
    pixelSum = 0;
    for windowX : imageX - (w/2) -> imageX + (w/2)
      for windowY : imageY - (w/2) -> imageY + (w/2)
        if isWithinImage(windowX) && isWithinImage(windowY)
          pixelSum = pixelSum + image(windowX, windowY);
    pixelMean = pixelSum / (image.width * image.height);
  }
```

Figur 6.4: Pseudokod över beräkning av medelvärde för varje pixel i Sauvola.

Pseudokoden i figur 6.4 innehåller endast vad som är nödvändig för att beräkna medelvärdet för varje pixel i ett *w***w*-fönster. Approximativt sker i detta exempel *bredd***höjd***w***w* antal beräkningar. Skulle beräkningen ske på en bild som var 3000*2000 pixlar med ett 16*16-fönster, skulle det bli 1 536 000 000 beräkningar på en bild. För att bättre anpassa implementationen av Sauvola efter Android-enheter där det inte finns tillgång till lika stor beräkningskraft gjordes ett antal modifieringar som beskrivs i följande stycken.

Eftersom applikationen endast ska hantera text (svarta pixlar) mot bakgrund (vita pixlar) är det tillräckligt att hantera svartvita bilder, vilket betyder att det räcker med ett färgvärde för att beskriva varje pixel istället för att behöva använda tre (R, G, B). Det första som sker är att bilden görs om till en matris med enkla heltal med värdena 0 till 255 för att slippa räkna ut gråvärdet varje gång algoritmerna vill använda sig av information om en pixel. Detta gör att antalet läsningar för pixelvärdena skalas ner till en tredjedel.

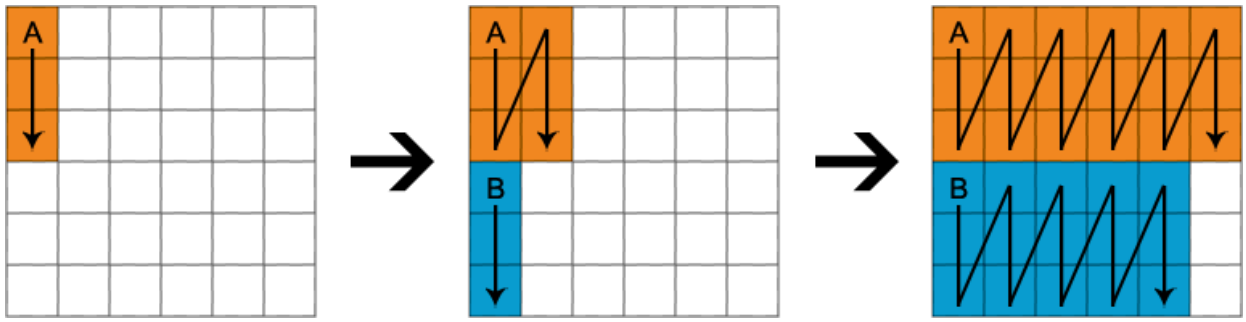
För varje pixel i bilden räknar Sauvolas metod ut medelvärdet av pixlarna i ett *w***w* stort fönster runt den aktuella pixeln. Eftersom detta innebär många genomlöpningar av samma pixel undersöktes möjligheten att förberäkna medelvärden och varians i så kallade integralbilder. Fördelen med integralbilder är att de är enkla att beräkna då varje pixel tilldelats en vikt sådan att den utgör summan av värdena från alla pixlar inom en ram från bildens övre vänstra hörn till den aktuella pixeln. Detta gör det

möjligt att räkna ut medelvärdet av ett helt område genom endast fyra läsningar från den färdiga integralbilden, istället för att göra *w**w* läsningar från originalbilden. Algoritmen går nu från att utföra *bredd**höjd**w**w* beräkningar till *bredd**höjd**4.

Medan beräkningstiden för den gamla modellen är exponentiellt beroende av fönsterstorleken, är beräkningstiden för den nya modellen linjärt proportionerlig mot bildens storlek och helt oberoende av fönsterstorleken. För att sätta det i perspektiv skulle en beräkning av en bild som är 3000*2000 pixlar med en fönsterstorlek på 24 pixlar innebära 3 456 000 000 beräkningar utifrån den gamla modellen, medan den nya endast kräver 24 000 000 beräkningar. Detta innebär att i exemplet kräver den nya metoden endast cirka 0,7 % av beräkningarna hos den gamla.

Det mest tidskrävande efter integralbildsoptimeringen var beräkningen av standardavvikelse, och därför undersöktes möjligheten att använda den beräknade integralbilden över pixelsummor för att skapa en integralbild över varianssummor och få samma förbättring på beräkningarna av standardavvikelse. Efter dessa förändringar visar tester att vid en fönsterstorlek på 16 pixlar utförs endast 1,1 % av de ursprungliga beräkningarna vid användning av integralbilder.

Dessa optimeringar har gjort algoritmen för Sauvola avsevärt mindre beräkningstung, men har istället flyttat beräkningstiden till generering av matriser innan Sauvola används. Även här gjordes en del optimeringar. Applikationen har kodats så att det är möjligt att använda ett antal trådar för att bygga upp integralbilder, vilket innebär att arbetet görs snabbare. Trådarna bygger upp matrisen utifrån det övre vänstra hörnet. Då vikten hos varje pixel beror av vikten hos pixeln direkt ovanför, till vänster, och snett upp till vänster om pixeln, är det viktigt att trådarna jobbar i ordning. Ett exempel med två trådar visas i figur 6.5.



Figur 6.5: Illustrering av uppbyggnad av matriser med hjälp av olika trådar. Varje bokstav representerar en tråd som arbetar självständigt, tillsammans bearbetar de hela bilden.

I figur 6.5 visas två trådar: A och B. Tråd A börjar jobba i kolumn 1 tills den når mitten på bilden, och signalerar därefter tråd B att sätta igång. Efter att B är signalerad fortsätter A på kolumn 2, och tråd B har tillåtelse att arbeta i den kolumn som A jobbat klart i. Denna optimering är bra då den är generell och framtidssäker eftersom den går att justera för valfritt antal trådar.

6.7.3 Antibrus

Något som kan förekomma efter binäriseringsprocessen är visuellt brus i områden med varierande bakgrund. För att ge användaren möjligheten att reducera detta brus implementerades en antibrusalgoritm.

Första versionen som implementerades utgick från integralbilderna som diskuterats tidigare, då all data som behövs för att genomföra antibrus finns i den optimerade datastrukturen. Vid ett senare skede av

utvecklingen upptäcktes ett behov av att kunna applicera antibruset flera gånger för att lättare kunna bestämma när bruset är borta. Detta skulle leda till en ombyggnad av integralbilderna varje gång antibruset användes, och då fönsterstorleken för antibrus är låg jämfört med den i binäriseringen märktes inte tillräckligt stora förbättringar i körningstid för att motivera att använda sig av integralbilder. Metoden utgår därför från det ursprungliga resultatet av binäriseringen.

Antibrusalgoritmen går ut på att bestämma ett medelvärde från ett lokalt fönster runt den pixel som brusreduceras, och ändra pixelns värde till detta. Detta medför dock att bilden blir en aning suddigare och att viss noggrannhet förloras. Då binäriserade bilder används tvingas därefter medelvärdet att anta antingen en svart färg eller en vit, så denna suddighet inte kommer uppfattas.

Detta ger ett resultat som tar bort svarta pixlar som kan anses som brus, men fyller även i vita pixlar som omringas av svarta. Detta innebär både fördelar och nackdelar. I fallet då bokstäver innehåller vita pixlar, och kan läsas fel i den optiska teckenigenkänningen kan det vara en fördel då dessa vita pixlar fylls i av algoritmen. Ett fall som medför nackdelar är då två bokstäver är separerade av en eller flera vita pixelrader, detta eftersom de separerade vita pixelraderna kan fyllas i av antibrusalgoritmen då de är omslutna av svarta pixlar. Scenariot kan undersökas i senare iterationer och därefter kan beslut tas om man ska dela upp antibrusalgoritmen att göra två separata saker, där en del tar hand om att korrigera svarta pixlar, och den andra är mer specifikt implementerad för att fylla i bokstäver som saknar pixlar.

6.8 Tesseract OCR

Tidigt i projektet testades möjligheten att utföra den optiska teckenigenkänningen på Android-enheten, men på grund av för stor minneskonsumption och beräkningstid användes istället en server som utför mer resurskrävande beräkningar för att avlasta Android-enheten och göra binäriseringsprocessen snabbare. Tesseract är ett ramverk för optisk teckenigenkänning som är implementerat i C/C++. Detta bibliotek är ett projekt med öppen källkod som även är enkelt att använda från C/C++. För att komma åt funktionaliteten från Java krävs en wrapper. I denna lösning användes Tess4J (Tess4J 2013).

6.9 Nätverk

I Java används sockets för att skicka data mellan system som befinner på olika nätverk. En socket är en förbindelse mellan två system, antingen via strömmar (TCP, en säkrare metod utan paketförlust), eller via paket (UDP, en snabbare men osäkrare metod). Då applikationen inte är beroende av att anropstiden till servern är kort, och av anledningen att man vill vara säker på att all data kommer fram har gruppen valt att använda TCP som dataöverföringsprotokoll. För att skicka data över strömmar i Java används `InputStreams` och `OutputStreams`. Det finns olika klasser för att skicka olika datatyper över dessa strömmar, men dessa är inte kompatibla med varandra. Applikationen ska skicka data i form av textsträngar, heltal och ren byte-data, och därför implementerades en klass för hanteringen av datatrafiken. Detta ger applikationen fullständig kontroll över vilken data som skickas, och innebär ett så optimerat system som möjligt.

6.9.1 Byte-omvandling

Den uppgift en `Writer` har i Java är att översätta den data som ska sändas till dess byte-representation. En nackdel som upptäcktes var att vissa `Writers`, exempelvis `BufferedWriter`, använder sig av buffrar för att reducera antalet skrivningar till strömmen. På samma sätt buffrar en `Reader` data i omgångar för att minska antalet läsningar. Detta gör att en utvecklare som använder två klasser för att arbeta mot samma ström inte kan veta vilken data som mottages, eftersom den andra klassen redan kan ha buffrat den sökta datan från strömmen. Målet med gruppens egen implementering är att samla den byte-omvandling som är relevant för projektet i en klass. Denna klass används sedan från egen implementering av `Writer` och `Reader` för att kunna skriva direkt till strömmarna utan att behöva använda sig av de fördefinierade klasserna, och därmed undviks den ovannämnda osäkerheten.

6.9.2 Skrivning till strömmar

Den information som applikationen i största utsträckning skickar till servern är information om pixlar. Från mobiltelefonen vill man därför konvertera resultatet från binäriseringen till en byte-sekvens och skicka det genom strömmen. På serversidan vill man bygga upp bilden från denna byte-data och sedan anropa den optiska teckenigenkänningen. Det går att utnyttja det faktum att bilderna som skickas alltid är binäriserade genom att låta protokollet utgå från att varje pixels värde endast kan anta två värden. I och med att en bit endast kan anta två värden var den första implementationen av bitrepresentationen en bitföljd som bestod av bildens dimensioner, följt av bitar som representerar pixlar med värdet 0 om pixeln var svart, och 1 om den var vit. Implementationen använder sig av 32 gånger mindre utrymme tack vare detta, och blir därför 32 gånger snabbare.

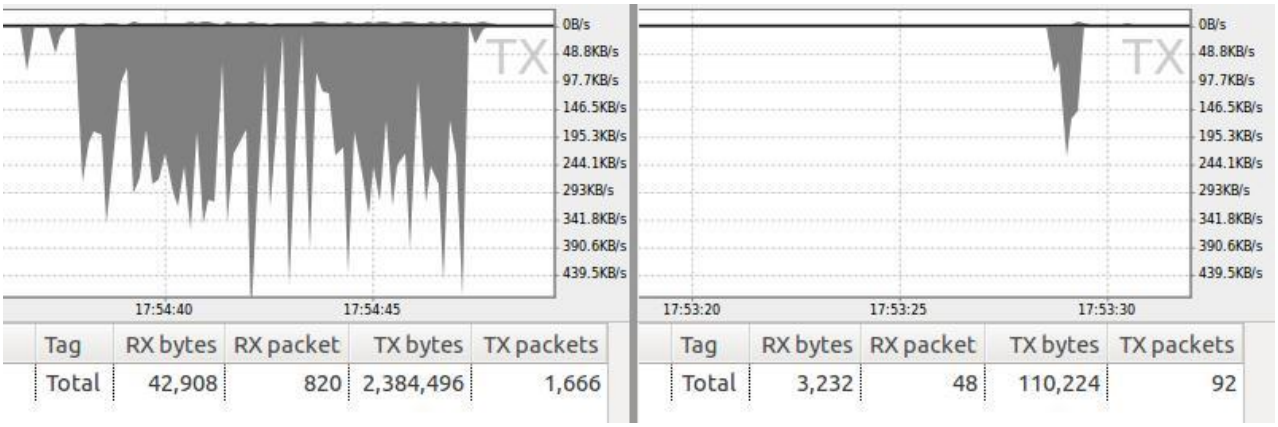
Efter testning märktes att överföring hade stor påverkan på processorn i enheten på grund av de beräkningar som krävdes för att bygga upp byte utifrån pixlarna. Därför testades en ny implementering som istället grundas i att man vet att bilden är binäriserad och att protokollet därför endast behöver beskriva de svarta pixlarna. Mottagarsidan kan sedan anta att de onämnda pixlarna är vita. Det går inte att direkt uppskatta tidsåtgången för denna metod eftersom bilden inte har en konstant andel svarta pixlar, men då bakgrunden ofta tar upp majoriteten av den binäriserade bilden skickas så lite data som möjligt utifrån förutsättningarna. Denna lösning är också beräkningseffektiv då det inte är nödvändigt att beräkna något annat än omvandlingen från `Integer` till byte. Samma teknik används för textsträngar som skickas från server till mobil enhet, och innehåller resultatet från den optiska teckenigenkänningen. Varje tecken i en textsträng kan representeras av två byte. Varje textsträng görs därför om till en byte-sekvens bestående av först fyra byte som beskriver hur många kommande byte som ska göras om till en `short`-sekvens, som i sin tur går att tolka om till den ursprungliga textsträngen.

6.9.3 Läsning från strömmar

Med den slutgiltiga implementeringen blir återskapningen av bilder från byte-sekvensen enkel. Applikationen kan redan från början läsa in de första två värdena från strömmen som definierar bildens storlek, och därefter skapa en bild där alla pixlar är vita. Därefter läses 8 byte, som representerar vilken pixel som ska bli svart, in i taget. Detta görs tills alla värden är mottagna. Lösningen medför minimal beräkning för både server och klient, och överföringshastigheten är märkvärt snabbare än att skicka den fullständiga bildrepresentationen.

6.9.4 Utvärdering av optimeringen

Som beräknat minskades datamängden som skickas över nätverk till stor del. I figur 6.6 visas skärmdumpar från programmet Android Device Monitor som ingår i Android SDK. I bild A visas hur mycket data som skickats från enheten då all data från ett kort skickas, utan några förbehandlingar. Bild B visar den optimerade metoden för att endast skicka den data som är relevant, i detta fall svarta pixlar. Samma kvittoavbild användes i båda testen, och applikationens tillstånd var lika i båda testerna innan nätverkstrafiken skickades. Det är möjligt att se att bilden innehöll ungefär 0,46 % svarta pixlar från det direkta förhållandet mellan den högra och vänstra bildens överförda byte, då båda fallen beskriver en pixel med hjälp av en `Integer`, Detta stöder beslutet att endast skicka svarta pixlar och förutsätta att resten är vita.



A **B**

Figur 6.6: Jämförelse över nätverkstrafik före och efter optimeringar. I bild A visas mätningar där obehandlad originalbild skickas från mobil enhet till server över nätverk, i bild B mätningar där endast positioner som beskriver svarta pixlar skickas över nätverk.

6.10 Minneshantering

På grund av de begränsade minnesresurserna uppstod problem med att hålla alla datastrukturer i minnet under binäriseringsprocessen. I ett test tog en användaren ett kort som blev 2448x3264 pixlar stort, vilket kräver ungefär 31 mb utrymme i minnet. Utöver detta behöver applikationen även lagra en lika stor mängd data för att beskriva integralbilder för medelvärden och varians, samt för att spara resultatet. I testet hann inte applikationen allokera all data innan den fick ett `OutOfMemoryException`, men hade koden exekverat klart hade den uppskattade minnesåtgången varit 155 mb. Detta innebar problem i början av projektet då en Android-enhet som använder API-nivå 2.2 med säkerhet endast har tillgång till 16 mb minnesutrymme (Google 2010).

Då det tillgängliga minnesutrymmet varierar mellan telefonmodeller används en lösning där applikationen läser in en bild med lägre upplösning för att få plats med bilden i minnet. Då det inte går att säkert räkna ut hur mycket minne applikationen använder som mest, eftersom minneshanteraren i Android kallas först när systemet tycker att det är lämpligt, testar sig applikationen fram. Först görs ett försök att ladda in bilden i full upplösning. Går inte detta provas en mindre storlek tills all data är inläst utan att ha överskridit sin maximala gräns.

Denna strategi har löst de flesta minnesproblem som omfattade binäriseringsprocessen, dock kvarstod ett problem. Då bilden och tillhörande datatyper är inladdade i minnet har inte applikationen nämligen ingen någon kontroll över återstående allokeringar. Om applikationen har laddat in allt nödvändigt och endast har 1 byte kvar i minnet är det därefter omöjligt att ens deklarera en siffra till. Därför beslutades att i början av binäriseringsprocessen allokera plats för alla datastrukturer som används i samband med inladdningen av bilder, vilket verkar vara den enda lösning som garanterar en felfri körning av binäriseringsprocessen.

6.11 Server

Serverns huvudsakliga uppgift är att lyssna på anrop från mobila enheter, ta emot data, utföra nödvändiga beräkningar och svara med ett resultat. För att kommunicera över nätverk används nätverksprotokollet TCP (se *6.9 Nätverk*), och när datan är mottagen avgörs vilken beräkning som ska utföras. I nuläget hanterar servern endast förfrågningar om att genomföra optisk teckenigenkänning, men är byggd för att klara uppgifter som kan komma att implementeras i ett senare skede.

Servern använder sig av ett kösystem för att hantera situationer där många användare försöker ansluta samtidigt. Anropen hanteras sekventiellt och prioriterar det anrop som väntat längst tid i kön. Tack vare detta kommer servern ej överbelastas och det är lätt att justera hur många anrop i taget som ska hanteras och att föra statistik över hur länge en användare väntar innan dess anrop behandlas. Detta gör det är lätt att fatta beslut rörande när uppgradering av hårdvaran behöver ske.

Det upptäcktes i ett senare skede av projektet att ramverket som använts för den optiska teckenigenkänningen får irreguljära segmentfel som inte går att hantera. När dessa sker stoppas exekveringen av programmet eftersom processen som hanterar programmet avslutas. Därför utvecklades ett stödprogram som övervakar servern och känner av om ett segmentfel skett. Har ett sådant fel uppstått startar stödprogrammet upp servern igen och fortsätter lyssna efter fel. På så sätt kommer servern alltid hållas körande. Denna lösning togs vid då det inte går att påverka segmentfelen från Tesseract eftersom det inte sker i projektets kod.

6.12 Korrigering av felaktiga tecken från teckenigenkänning

En algoritm för att hitta och korrigera fel efter den optiska teckenigenkänningen implementerades under projektets gång. För att användaren ska tycka att rättningarna som görs av programmet är så korrekta som möjligt baseras besluten på den matematiska modellen Markovkedjor som byggts upp av tidigare rättningar gjorda av användaren.

Varje gång användaren ändrar ett ord som teckenigenkänningsprocessen gjort fel på lagras ändringen i databasen som korrigering av ett ord till ett annat. Finns redan övergången ökas vikten för denna korrigering, och ordet blir därmed mer sannolikt att föreslås som rättning. Innan resultatet från teckenigenkänningen visas för användaren beräknas vilka korrigeringar som har störst sannolikhet att eftersökas. Denna metod valdes före en rättningsprocedur med fördefinierad ordlista då gruppen ville att rättningsförslagen skulle vara mer personliga.

På grund av att idén om att rätta felavlästa tecken uppkom sent i projektet, och att andra funktioner prioriterats högre, har rättningsfunktionen endast implementerats utan att integreras i användargränssnittet, då det hade varit för tidskrävande att göra funktionen användarvänlig.

6.13 Parsning av resultat från teckenigenkänning

När applikationen har fått ett resultat i form av text behöver det parsas för att kunna passa in de olika delarna (produkter, datum, tid och affär) från ett kvitto. Detta görs genom att varje rad i resultatet jämförs mot ett antal olika regex.

Implementering av parsningen skedde gradvis allteftersom det upptäcktes hur de olika delarna av kvittot skulle kunna identifieras i resultatet från teckenigenkänningen. Parsningen är utformad att efterlikna sättet människan känner igen mönster, i detta fall de olika delarna på kvittot. För övergripande pseudokod över hur parsningen går till, se appendix D. Alla resultat av parsningen skickas vidare till en vy i vilken användaren själv kan redigera slutresultatet (se *7.4 Tilläggning av ett kvitto*).

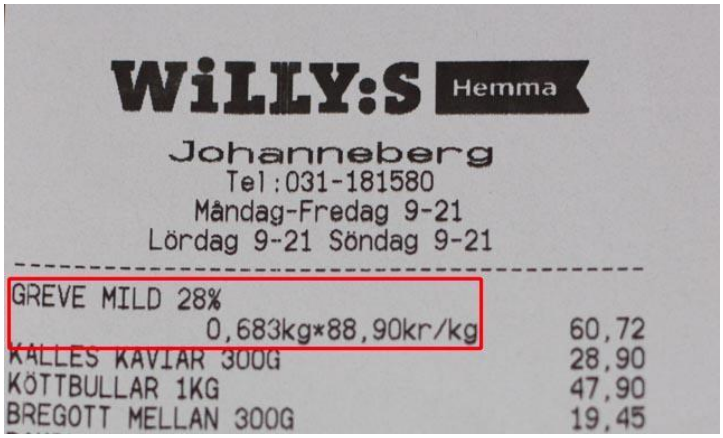
6.13.1 Produktrader

Det konstaterades att en produktrad oftast känns igen på att den slutar med ett pris och inleds av produktnamnet. Undantagsfall finns dock, exempelvis avslutas produktrader på kvitton från IKEA med bokstaven A, se figur 6.7. Med hjälp av detta utformades det reguljära uttrycket för produktrader, som består av en del för produktnamnet, en del för priset och en del för de eventuella bokstäverna som kommer efter priset.

Figur 6.7: Ett exempel på avvikande utseende på produktrader. Den röda markeringen visar på element som inte finns på de flesta kvitton.

Eftersom vissa produkter har långa namn finns det fall då produktnamnet sträcker sig över två rader, se figur 6.8. Som en följd av detta valdes att, i de fall där en rad inte matchar en produkt, även kontrollera nästa rad. Om den matchar regex för produkter ska de två raderna läggas ihop till en produktrad.

Nackdelen med denna teknik är att raden ovanför den första produktraden på kvittot inte är en produkt. Detta innebär att den och första produkten i vissa fall läggs ihop till en produktrad, beroende på hur kvittot är utformat. Detta är ett svårlöst problem som det ännu inte har hittats en bra generell lösning på. De gånger öppettiderna står på raden precis ovanför första produktraden, se figur 6.8, kan fel dock förhindras genom att kontrollera om raden ovanför den första produktraden innehåller namn på en veckodag.



Figur 6.8: En produkt vars namn är så långt att det sträcker sig över två rader. Värt att notera är att Tesseract bortser från skiljeraden bestående av enbart bindestreck, så denna räknas inte med som en rad i resultaten från teckenigenkänningen.

6.13.2 Övriga rader

Parsningen av de övriga fälten på kvittot implementerades genom att raden, om den inte är en produktrad, matchas mot de olika reguljära uttrycken för datum och tid. Om det inte finns någon information om datum eller tid på kvittot så lämnas de fälten tomma så att användaren själv får fylla rätt uppgifter.

Affärsnamn har ingen särskild form som gör det lätt att extrahera dem från ett kvitto. Därför sköts parsningen genom att den aktuella raden söks igenom efter något av de redan tillagda affärsnamnen i databasen. På så sätt blir avläsningen av affärsnamn bättre ju fler kvitton användaren har i sin databas, och användaren har själv större möjlighet att påverka att affärsnamnet blir rätt parsat.

6.14 Exportering av data

För exportering av kvitton används filformatet CSV, som kommer från engelskans *comma-separated values* (på svenska *kommaseparerade värden*). Formatet valdes eftersom det kan läsas på ett stort antal plattformar och av många program. Applikationen använder sig av OpenCSV, ett bibliotek med öppen källkod som används för att läsa och skriva CSV-filer (OpenCSV 2011).

När e-post skickas från en Android-applikation skapas ett objekt med all information som e-brevet ska innehålla, som sedan skickas till en applikation för e-post. Detta innebär att användaren måste ha en sådan applikation installerad på sin telefon, samt att man manuellt måste klicka på skickaknappen i den. Tanken var från början att applikationen skulle sköta allt automatiskt med hjälp av biblioteket JavaMail (Oracle 2013c). Problemet med att använda denna metod var att ytterligare ett bibliotek samt ett separat e-postkonto åt applikationen hade varit nödvändigt. Därför ansågs det vara bättre att använda metoden som redan fanns i Android.

Vid första implementationen uppstod problem med att filen inte inkluderades i e-brevet, trots att det såg ut som att den var inkluderad. Detta berodde på att filen sparades den på applikationens interna minne när den skapades, och detta gjorde att e-postapplikationen inte hade behörighet att läsa den. Det löstes genom att spara filen på minneskortet istället.

Ett annat problem var att filen skulle tas bort efter att den exporterats, för att den inte skulle ta upp plats på minneskortet. Om detta gjordes direkt efter att man skickat filen till applikationen för e-post

uppstod ett problem med att filen inte kom med i e-brevet, eftersom man då tog bort filen innan den blivit läst av den andra applikationen. Istället valdes det att vid start av applikationen undersöka om det finns filer i mappen där de sparas, och då ta bort dem.

6.15 Användartest av första prototypen

Som nämndes i kapitel *2 Kravspecifikation för applikationen* beslutades det tidigt att genomföra användartester när det fanns en fungerande prototyp. Efter att iteration två avslutades i vecka 15 fanns en prototyp, men testet senarelades på grund av förberedelser och finjustering av funktionaliteten.

I slutet av vecka 17 genomfördes användartester på tre personer. Detta kan tyckas alldeles för få för att det ska gå att få någon vettig information från testet, men enligt Barnum uppnås ett optimalt kostnads-användbarhetsratio när testgruppen består av tre till fem personer. Det går inte att bygga ett statistiskt underlag på så få, utan små tester resulterar i listor med saker som borde åtgärdas (2011). Detta är användbart för att kunna höja kvaliteten på projektet inom en liten tidsram.

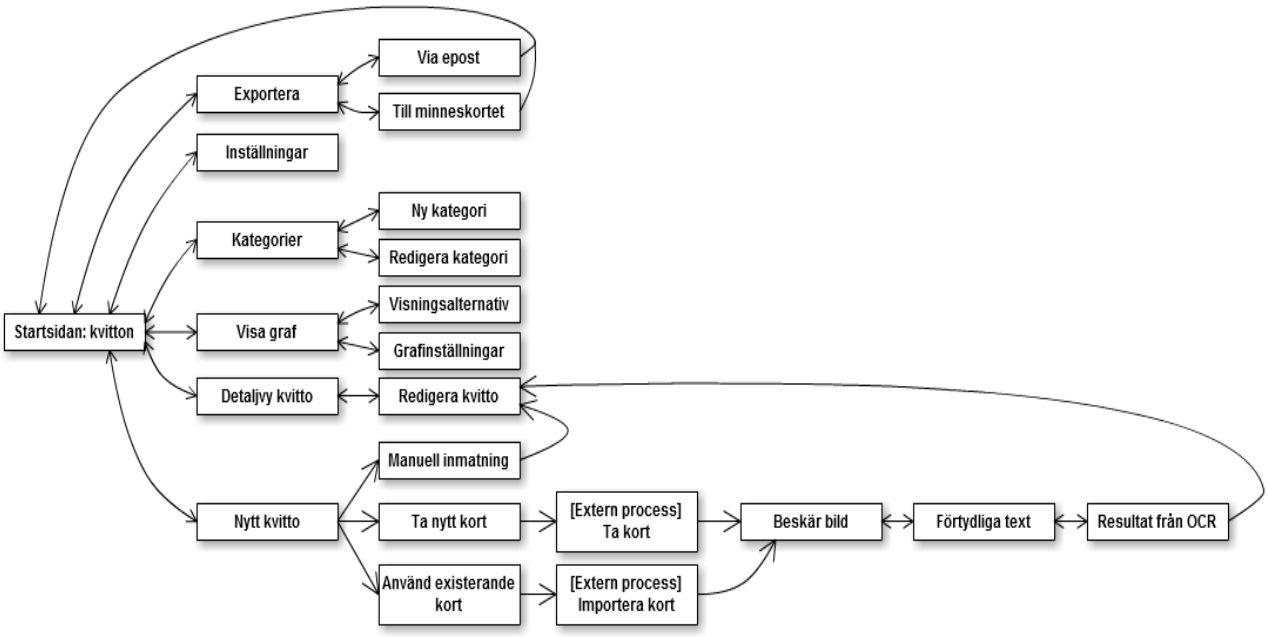
Inför testen förbereddes fem scenarion för att testa en bred del av applikationens funktionalitet, protokoll för testledaren, samt tre olika formulär för användaren att fylla i under olika stadier av testet. Dessa formulär kan hittas i appendix C. Dokumenten användes för att få en så konsekvent testsituation som möjligt för de olika testpersonerna, samt för att dokumentera resultaten och reaktionerna. Djupare, öppnare frågor ställdes även för att utforska noterade reaktioner under testet.

Testgruppen bestod av två högerhänta män, 20-30 år gamla, och en högerhänt kvinna, 20-30 år gammal. Funktionaliteten som var mest uppskattad var möjligheten att spara kvittot genom att ta kort på det, medan uppskattningen av de övriga funktionerna varierade mellan användare. Navigeringen i applikationen uppfattades överlag som lätt att förstå och använda, och även användandet av grafer upplevdes som tydligt av användarna. Applikationens svaga punkter var till exempel tydligheten i hur kategorier är tänkta att användas; användarna lade knappt märke till dem på egen hand. Under testen upptäcktes ett antal problem med programmet, och användarna gav även förslag på saker som kunde förbättras. Som ett resultat av detta kopplades till exempel hårdvarans menyknapp till den overflow-meny som finns i applikationens menyrad, och grafernas alternativ sågs över.

Förutom dessa strukturerade användartest har applikationen genomgått mindre, ostrukturerade test för kontinuerlig utvärdering. Exempel på detta är att en individ fick prova att beskära en bild för att gränssnittet skulle kunna utvärderas, och under testet upptäcktes att användaren inte förstod att man var tvungen att dra i en specifik linje för att justera storleken på fönstret för beskärning. Utifrån detta ändrades ytan som vara möjlig att dra i.

7. Resultat

I följande avsnitt presenteras den färdiga applikationen, under utvecklingen döpt till Kiwitton, och dess olika delar. Applikationens gränssnitt och funktioner kommer diskuteras utifrån olika vyer, och för att kunna placera de olika delarna i ett sammanhang visas flödet mellan applikationens olika vyer nedan i figur 7.1. Notera att förutom vyer visas även kritiska dialogrutor i figuren.

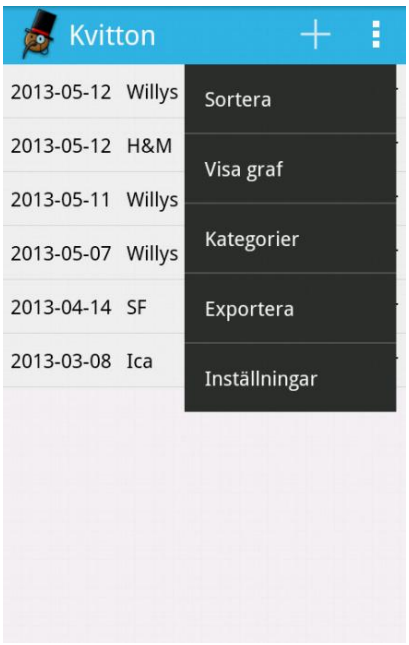


Figur 7.1: En översikt över applikationens centrala vyer och dialogrutor, samt flödet mellan dem.

7.1 Applikationens genomgående action bar

Genomgående för så gott som alla vyer i applikationen är att det visas en action bar längst upp i fönstret, se figur 7.2. Innehållet i denna action bar förändras beroende på vilken vy som visas, men i alla fall förutom på förstasidan finns en uppknapp längst till vänster. I de fall menyraden innehåller en action overflow aktiverad av symbolen längst till höger i figur 7.2 är denna även kopplad till enhetens menyknapp. Ett exempel på action overflow finns i figur 7.3. Eftersom innehållet i action bar varierar mellan olika vyer kommer detta diskuteras närmare i relevanta avsnitt.

Figur 7.2: Ett exempel på action bar, taget från grafvyn. Notera overflow-ikonen längst till höger.



Figur 7.3: Ett exempel på en action overflow, taget från startsidan. Menyn aktiveras genom tryck på overflow-ikonen.

7.2 Representation av ett kvitto

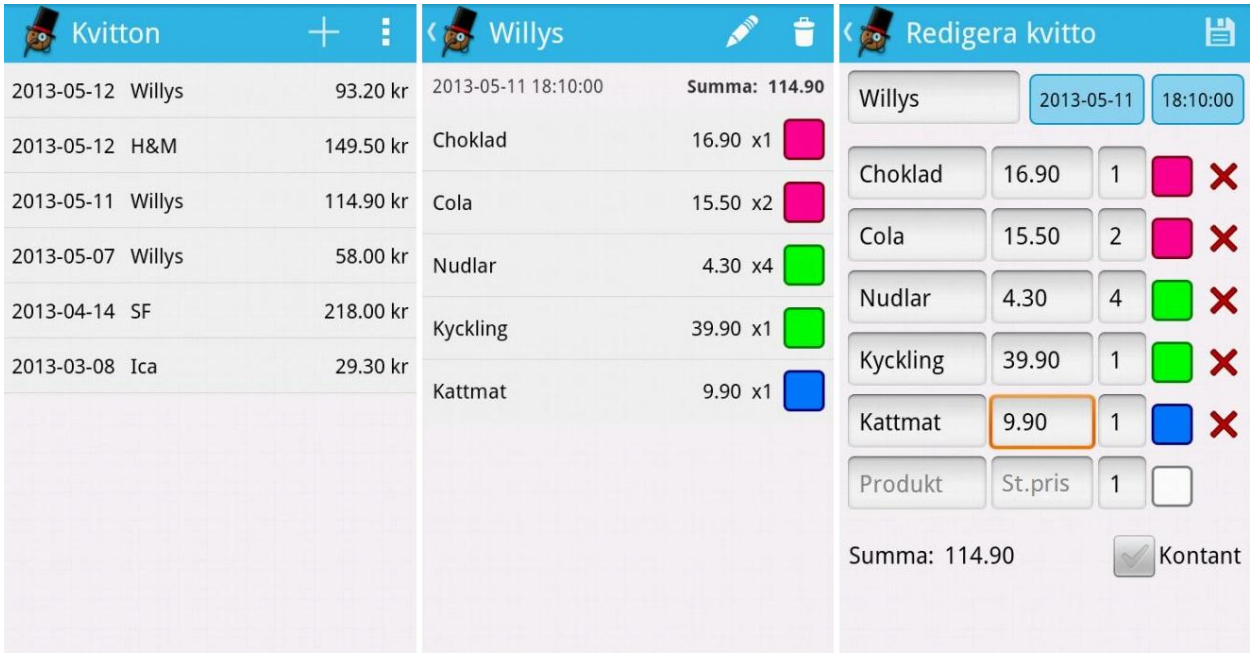
Alla kvitton inlagda i applikationen listas på förstasidan, vilket kan ses i figur 7.5 A. För varje kvitto visas datum för inköp, affärens namn och totalkostnaden. Listan kan sorteras efter datum, pris eller affärens namn i stigande eller fallande ordning. Via menyraden kan ett antal alternativ som inte direkt berör representationen av kvitton nås, vilket beskrivs i senare avsnitt.

Vid tryck på ett kvitto visas en detaljvy innehållande kvittots alla produkter och tid för inköpet, se figur 7.5 B. Varje produkt har ett namn eller beskrivning, ett pris, en förteckning över hur många av produkten som köpts in, samt en kategori. Som standard visas styckpriset gånger antalet, se figur 7.4 A, men vid tryck på en produktrad visas istället den totala kostnaden, se figur 7.4 B. Vid tryck på en kategori, den färgade rutan längst till höger på en produktrad, visas en meddelanderuta innehållande kategorinamnet. Vid tryck på soptunnan erbjuds användaren att ta bort kvittot.

A **B**

Figur 7.4: I bild A visas en produkt med styckpris och antal, i bild B med totalpris.

Om en användare väljer att redigera kvittot genom att trycka på pennan i action bar visas en redigeringsvy synlig i figur 7.5 C. Här finns möjlighet att redigera kvittots alla attribut och sedan spara det. Vid tryck på en kategori visas en dialogruta med alla tillgängliga kategorier, se figur 7.6. En hel produktrad kan raderas genom tryck på det röda krysset längt till höger. Om ett attribut sparas i ett felaktigt tillstånd, till exempel om en produkts namn lämnas tomt, visas ett felmeddelande och en markering av var felet skett. Om en redigering avbryts genom tryck på bakåtknappen måste användaren bekräfta, men endast om något fält har förändrats.



A **B** **C**

Figur 7.5: Bild A visar startsidan innehållande en lista på alla kvitton. Vid tryck på ett kvitto visas bild B som innehåller en mer detaljrik förteckning över kvittot. Bild C visas då användaren väljer att redigera kvittot.

Figur 7.6: Den lista över alla kategorier som används för att välja kategori till en produkt.

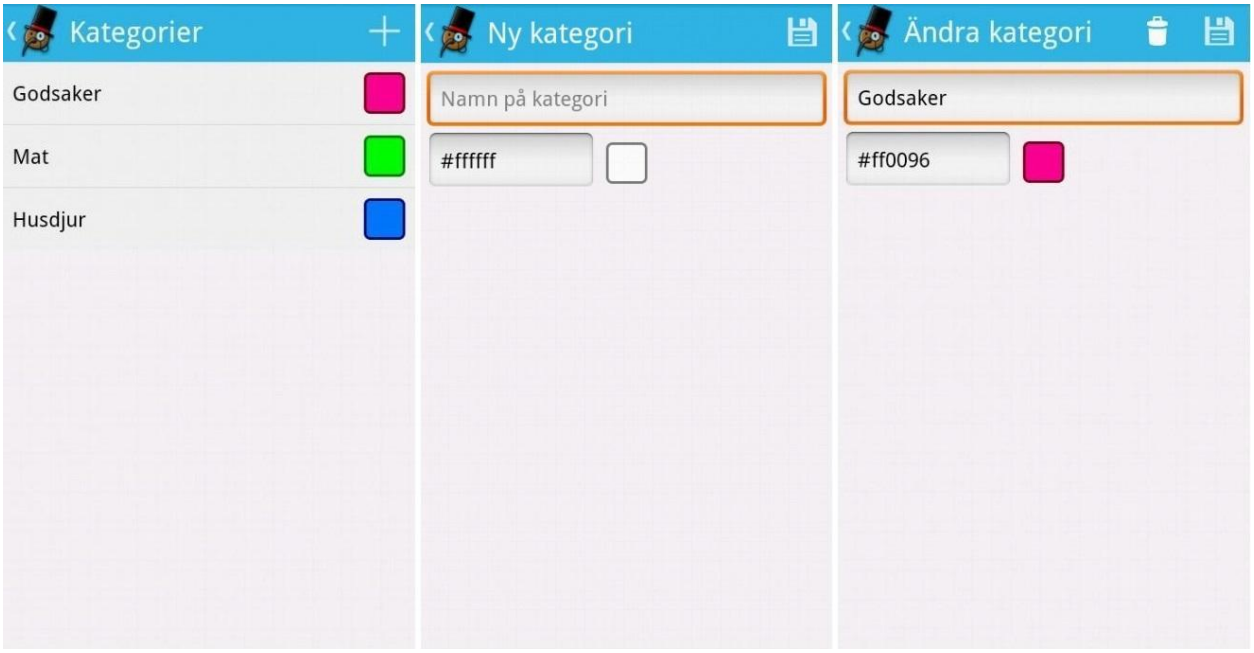
7.3 Kategorier

Som nämndes i bland annat avsnitt *7.1 Action bar* har varje produkt en kategori. Kategorier är avsedda att dela in produkter i grupper för att förenkla översikten över vad som köpts. Genom den action overflow som syns i figur 7.3 A är det möjligt att komma till en vy med översikt över alla inlagda kategorier, förutom kategorin “Ingen kategori”, se figur 7.7 A. I action bar visas ett plus för tilläggning av nya kategorier.

Vid tilläggning av en ny kategori är det möjligt att välja ett namn och en färg, se figur 7.7 B. Färgen kan antingen väljas med hjälp av ett färghjul, se figur 7.8, eller genom att skriva in en hexadecimal färgkod. Som standard är färgen vit.

Vid tryck på en kategori i figur 7.7 A visas en vy för redigering, se figur 7.7 C. Denna vy är så gott som identisk som den för tilläggning av en ny kategori, förutom att kategorins existerande namn och färg visas istället för tomt namn och standardfärg. Det är även möjligt att ta bort kategorin genom tryck på soptunnan som finns i vyns action bar. Användaren måste då bekräfta sitt val. Liksom i redigering av kvitton går det att avbryta genom att trycka på bakåtknappen, och om något fält har förändrats måste användaren bekräfta avbrottet.

Som nämnts i ett tidigare stycke finns det en kategori kallad "Ingen kategori". Av databasskäl måste en produkt alltid ha en kategori, och "Ingen kategori" infördes för att användaren inte skulle tvingas att ha kategorier mot sin vilja. Kategorin listas inte i översiktsvyn då det ej ska vara möjligt att ändra eller ta bort den.



A **B** **C**

Figur 7.7: Bild A visar en översiktsvy över kategorier. Vid tryck på plus-ikonen i vyns action bar visas bild B för tilläggning av en ny kategori. Vid tryck på en kategori visas istället bild C för redigering av kategorin.

Figur 7.8: Färghjul för val av färg. Färgvalet visas i den mittersta cirkeln, och bekräftas genom tryck på densamma.

7.4 Tilläggning av ett kvitto

Vid tryck på plus-ikonen i startsidans action bar, se figur 7.5 A, presenteras tre olika alternativ för tilläggning av kvitton, se figur 7.9. Användaren väljer ett alternativ genom att trycka på det, och därefter visas lämplig vy.

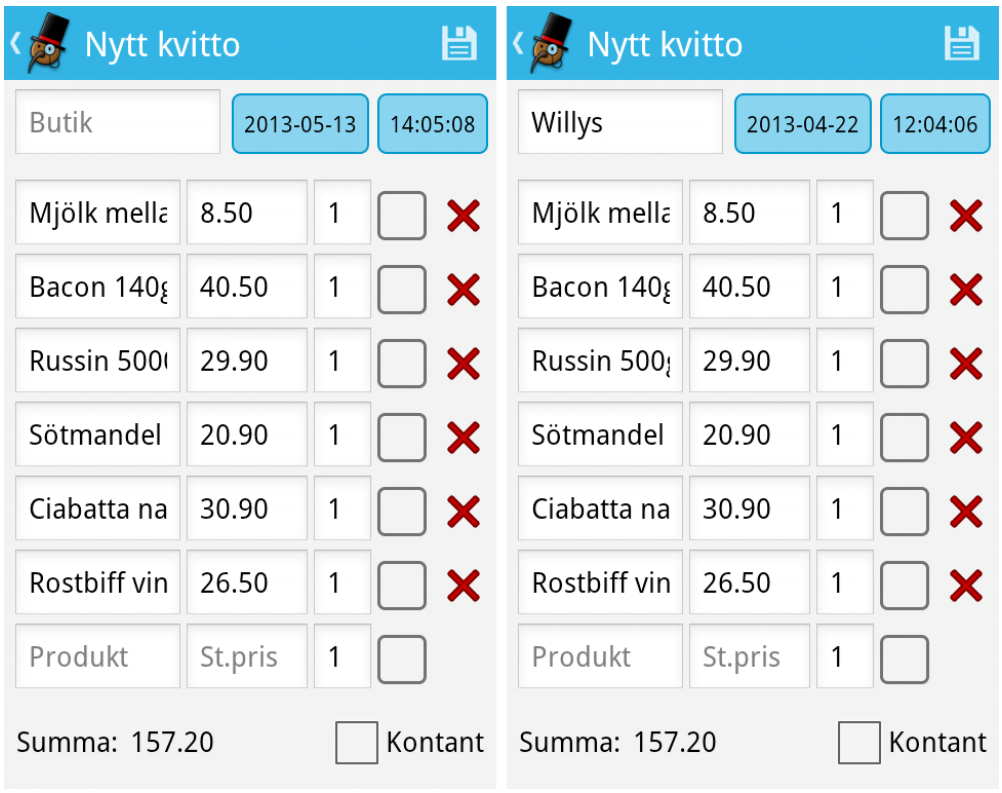
Figur 7.9: Tre olika alternativ för tilläggning av kvitton.

Trycker användaren på "Ta kort på kvitto" skickas den till enhetens kamera för att ta kort på det fysiska kvittot. Därefter visas en vy för att beskära bilden och rensa bort onödig information, se figur 7.10 A. Ramen för beskärning justeras genom att dra i linjerna som utgör den, eller i den grå delen av bilden. I action overflow ges möjlighet till rotation av bilden. När beskärningen sparas genom tryck på spara-ikonen i action bar visas en vy för binärisering av text, se figur 7.10 B. Användaren uppmanas justera bilden för att få så tydlig text som möjligt med hjälp av reglagen längst ner i fönstret. I action overflow är det möjligt att se originalbilden samt applicera antibrus på bilden. Vid tryck på spara-ikonen utförs OCR på bilden och en statusbar visas. Resultatet visas i en dialogruta, se figur 7.10 C. Vid tryck på knappen "Gör om" går applikationen tillbaka till vyn för binärisering.

A **B** **C**

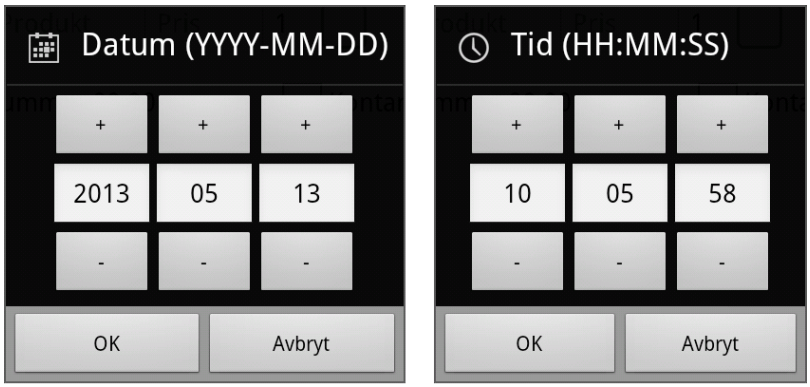
Figur 7.10: I bild A visas hur beskärningsvyn ser ut. Bild B innehåller vyn för binärisering av den beskurna bilden. Resultatet från OCR visas slutligen i bild C.

Om användaren väljer att acceptera resultatet visas en vy liknande den för redigering av kvitton, se figur 7.11 A. Alla produkter fylls i automatiskt från OCR, men användaren måste oftast välja affär, datum, tid och kategorier på egen hand. Datum och tid väljs genom att trycka på respektive knapp och välja värde i en dialogruta, se figur 7.12. Det finns även möjlighet att välja kontant betalning genom att bocka i rutan längst ner, vilket gör att totalpriset rundas av till närmaste hela krona. Ett exempel på ett kvitto som är redo att sparas syns i figur 7.11 B.



A **B**

Figur 7.11: Bild A visar hur applikationen parsar information från OCR-resultatet och fyller i informationen i lämpliga fält. I bild B visas den slutgiltiga versionen efter mindre ändringar från användaren.



A B

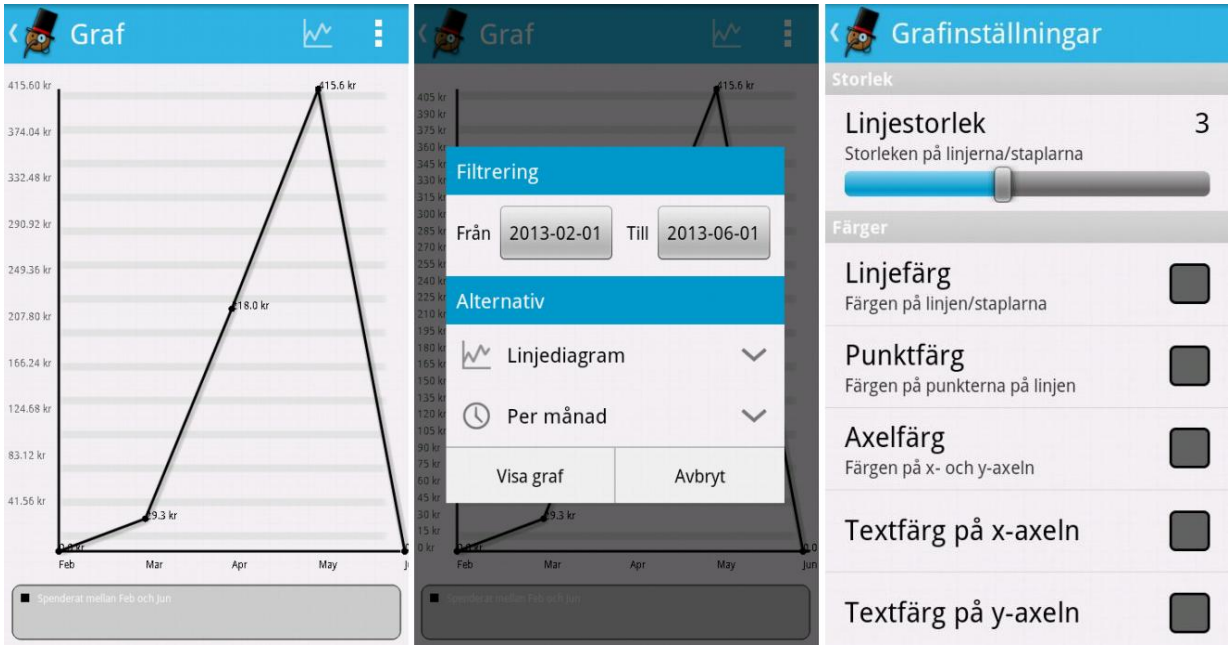
Figur 7.12: De dialogrutor som används vid val av datum och tid.

Väljer användaren “Välj en befintlig bild på ett kvitto” ges möjligheten att importera en bild till applikationen, till exempel från telefonens minne. Därefter sker samma process som beskrivits ovan. Om användaren istället väljer “Lägg in ett kvitto för hand” visas vyn i figur 7.11 A direkt, men utan några ifyllda produkter. All information måste då föras in manuellt av användaren.

7.5 Grafisk representation

För att grafiskt representera den data som lagrats i kvitton har applikationen en grafvy som nås genom att välja “Visa graf” i startsidans action overflow, se figur 7.13 A. En graf baserad på information från de kvitton som finns inlagda visas, och kan anpassas genom tryck på grafikonen i action bar. Då öppnas en dialogruta innehållande val för visning av grafen, se figur 7.13 B. Förutom val av tidsspann finns det möjlighet att välja mellan linje- och stapeldiagram, samt hur detaljerad grafen ska vara. Detaljnivån regleras genom att visa information dags-, vecko-, månads- eller årsvis. Det finns även möjlighet att dela upp utgifterna efter affär.

Genom grafvyns action bar går det att nå grafinställningar, och en del av valen finns synliga i figur 7.13 C. I nuläget är en stor del av valen kosmetiska, till exempel går det att ändra färg på linjerna som syns i grafen.



A **B** **C**

Figur 7.13: I bild A visas ett exempel på en graf. Vid tryck på grafsymbolen i action bar visas bild B för visningsalternativ. I action overflow går det att välja inställningar, och de visas i bild C.

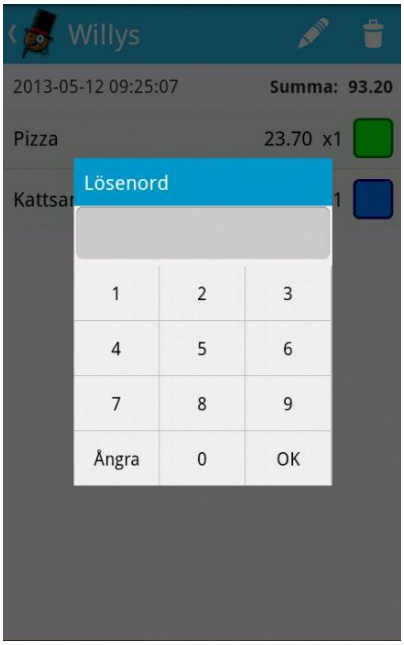
7.6 Inställningar

Genom tryck på "Inställningar" i startvyns action bar nås applikationens inställningar, se figur 7.14. Där går det bland annat att aktivera och deaktivera lösenordsskydd av applikationen. När funktionen aktiveras får användaren möjlighet att skapa ett lösenord. Efter uppstart krävs lösenordet för att kunna använda applikationen, se figur 7.15. Även vid avaktivering av funktionen krävs lösenordsbekräftelse.

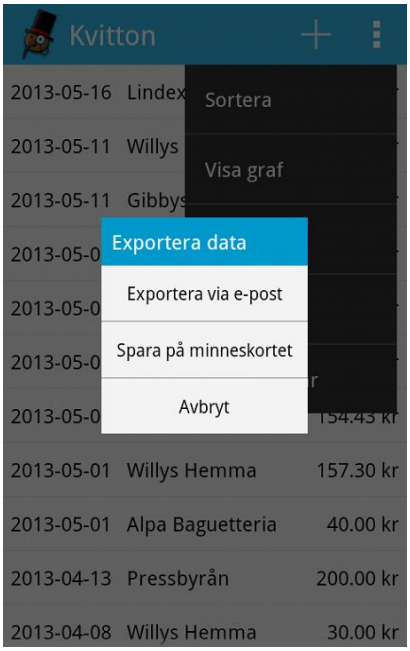
Det finns även möjlighet att ställa in en e-postadress för användning vid exportering av data. Om "Exportera" väljs i startsidans action bar visas en dialogruta synlig i figur 7.16. Vid val av exportering via e-post finns den valda e-postadressen förifylld, men det finns möjlighet att byta ut den tillfälligt.

Inställningen för permanent sparning av bildfiler är i nuläget alltid aktiverad, då det inte finns funktionalitet för att ta bort bilder ännu. Tanken bakom inställningen är att användaren ska kunna spara minnesutrymme, och inställningen visas i nuläget för att markera att alla tagna kort verkligen sparas.

Figur 7.14: Vyn för applikationens inställningar.



Figur 7.15: Applikationens utseende när lösenordsskyddet aktiveras. Endast rutan för ifyllning av lösenord går att använda.



Figur 7.16: Dialogrutan som visas när "Exportera" har valts i startsidans overflow-meny.

8. Diskussion

Projektets mål var att skapa en fungerande prototyp av en kvittoskannare, vilket har uppfyllts. Alla funktioner ur första och andra prioritet, samt vissa ur prioritet tre, har implementerats. Viss funktionalitet som inte alls nämnts i kravlistan har också inkluderats i applikationen, till exempel beskärning av bilder. Denna funktionalitet var nödvändig för att uppnå målen, men hade inte förutsetts i planeringsstadiet.

Trots att projektet resulterade i en fullgod prototyp finns det fortfarande möjlighet till utveckling och förbättring. De problem och funderingar som uppstått kommer tillsammans med möjligheter till förbättring att diskuteras i följande avsnitt.

8.1 Design av systemet

Inledningsvis lades mycket tid på att designa applikationens databas och uppbyggnad, med fokus på domänmodell och klassdiagram. Detta strider i viss mån mot det agila arbetssätt som användes, och när implementationsfasen inleddes blev till exempel klassdiagrammet snabbt föråldrat. Då informationen om systemet utbyttes verbalt under arbetet fanns inte ett behov av att hålla dokumentationen uppdaterad, vilket har lett till att det slutgiltiga klassdiagrammet i avsnitt *5.6 Systemets arkitektur* har en begränsad detaljnivå. Tiden som användes till den långa inledande planeringen kunde med andra ord utnyttjats på ett bättre sätt.

Även de krav som specificerades för systemet uppdaterades under projektets gång, men i mindre utsträckning. Den kravlista som finns tillgänglig i kapitel *2 Kravspecifikation för applikationen* har med andra ord förändrats sedan projektets uppstart, till exempel låg kravet rörande färgkodning av kategorier (se *2.3.2 Andra prioritet*) tidigare i tredje prioritet. Ny funktionalitet föreslogs även efter designfasen, vilket indikerar att gruppen inte lyckades skapa sig en komplett bild av systemet i designfasen. Trots detta var den grundläggande designen flexibel och modulär nog för att implementera förändringarna, och förhoppningsvis gäller detta även för framtida utökningar.

8.2 Metod och tillvägagångssätt

Projektet inleddes med en enkät för att få insikt i vad användare förväntar sig av en kvittoapplikation. Enkäten bekräftade många av de visioner gruppen hade och lade grund för användarvänlighet. I efterhand upptäcktes vissa brister med enkäten, till exempel hade det varit intressant med frågor om vilket operativsystem den tillfrågade använde på sin mobila enhet för att undersöka hur stor del av målgruppen som använde Android. Det kan även tilläggas att då enkäten endast gick ut till studenter på Data- och IT-sektionen på Chalmers är det endast personer med god datorvana som besvarat den och det är därför inte säkert att resultatet är representativt för hela målgruppen.

I avsnitt *5.1 Arbetssätt* nämns att nyckeldelar ur det agila arbetssättet Scrum har använts, dock har inte dessa nyckelelement ur Scrum följts helt korrekt. Till exempel har sprintarna varit enveckorsperioder, men vissa större uppgifter har sträckts ut till längre perioder. Även ändringar i prioriteringslistan har gjorts i efterhand och inte heller detta är vanligt i Scrum. Denna anpassning av arbetssättet har passat gruppen bra.

Att jobba agilt har varit en stor fördel för projektet då det har underlättat samarbetet. Större delen av tiden har hela projektgruppen arbetat tillsammans och kunnat hjälpas åt att lösa problem. Detta

förhindrade många missförstånd, och alla medlemmar har hela tiden varit medvetna om vad som hänt i projektet. Inledningsvis var det svårare att kommunicera på ett bra sätt då det var svårt att hitta arbetstider som passade alla deltagare, eftersom de flesta hade fler åtaganden under den första perioden. Viss frustration och missförstånd uppstod ur detta, men genom regelbundna möten kunde de problem som uppstått lösas.

Gruppen arbetade hela tiden med hjälp av verktyget Trello och de uppgifter som lagts in där. Detta var en bra rutin eftersom det var enkelt att prioritera uppgifter, se vad som skulle göras och när det skulle vara klart. Det var även lätt att hela tiden ha uppsikt över vad resten av projektgruppen jobbade med. En annan fördel var att nya funktioner och funna buggar kunde illustreras genom att skapa en uppgift för det, och sedan tilldela rätt gruppmedlem uppgiften.

8.3 Resultatdiskussion

Delar av resultatet anses vara tillfredsställande, men vissa delar kan ännu förbättras. I detta avsnitt diskuteras projektets resultat, vad som gjorts bra, och vad som skulle kunnat göras bättre.

8.3.1 OCR

I avsnitt *1.1 Syfte* nämndes att gruppen skulle undersöka existerande OCR-tekniker och möjligheterna att utveckla en egen teknik. Efter granskning av resultaten från redan färdiga tekniker med hjälp av Tesseract togs beslutet att optimera användningen av denna teknik istället för att utveckla en egen. En annan motivering till detta var den komplexitet som implementering av en OCR-teknik innebär. Resultaten är bra, förutsatt att den förbehandlade bilden som analyseras är tydlig. Detta beror på hur bra bilden blivit efter binärisering, och hur mycket av det kvarstående bruset som gått att filtrera bort.

Gruppen är nöjd med den optimering som genomfördes för att få resultat snabbt på Android-enheten, men ser även möjligheter att förbättra resultatet. Istället för att alltid använda sig av standardavvikelser för att räkna ut ett gränsvärde hade det i vissa fall blivit bättre resultat om man använt andra metoder. Gällande efterbehandling fanns tankar rörande korrigering av fel från textigenkänningsalgoritmen, men detta färdigställdes ej. Hade funktionen påbörjats tidigare hade förmodligen också ett bättre resultat uppnåtts.

Hela textigenkänningsprocessen består av att ett kort tas med kameran, bilden beskärs och binäriseras, sedan appliceras anti-brus, datan komprimeras med hjälp av byte-omvandlingar, datan skickas över nätverk, OCR genomförs av en server, resultat skickas tillbaka till Android-enheten, resultatet parsas, och slutligen skapas ett kvitto från resultatet. Detta är fler steg än vad som förutsågs i planeringsstadiet, men i nuläget är de nödvändiga för att få en så bra genererad kvittodata som möjligt. Inom projektets tidsram är gruppen nöjd med det resultat som uppnåtts, men är medvetna om att processen inte är tillräckligt exakt. De genererade kvittona är i många fall suboptimala, vilket indikerar att efterbehandlingen måste förbättras.

8.3.2 Gränssnitt

Gränssnittet ser till största delen ut som planerat. Vissa vyer har tillkommit under projektets gång, men Androids standarder har följts i största möjliga mån. Utseendet är menat att efterlikna temat Holo Light som introducerades i API-nivå 11. I de fall då dialogrutor utformats efter samma tema är dialogrutan i själva verket en egen aktivitet, och därför har dess utseende kunnat anpassas genom layoutfiler. Vanliga

dialogrutor, till exempel bekräftelserutan som dyker upp då användaren försöker ta bort ett kvitto, har fortfarande det temat från Android-versioner innan 3.0. Detta utgör inte bara en konflikt i form av estetik, utan även i konsekvensen hos knapparnas placering. I Android 3.0 sitter knappen för accepterande på höger sida, medan knappen för nekande sitter till vänster. I tidigare Android-versioner är det tvärtom. Då gruppen ännu inte funnit ett sätt att designa om de vanliga dialogrutorna valdes att följa de gamla placeringarna, så att åtminstone konsekvensen inom applikationen hålls intakt.

A **B**

Figur 8.1: I bild A visas en dialogruta innan API-nivå 11, och i bild B visas en dialogruta från efter API-nivå 11 (Android 2013h). Notera att knappen för att acceptera en handling och knappen för att neka har bytt plats.

I nuläget kan användaren sortera kvitton på huvudsidan via en dialogruta med två så kallade `Spinners`, på svenska *rullgardinsmenyer*. En `Spinner` visar nuvarande val, och då användaren klickar på den dyker en lista med alla tillgängliga val upp. Användaren väljer sedan ett av dessa och `Spinnern` uppdateras. För att sortera kvitton i nuläget kan användaren alltså behöva så mycket som fem klick: två för varje spinner och ett för "OK"-knappen i dialogrutan. Då sorteringsfunktionen först infördes fanns bara tre val och endast ett klick krävdes, men då fallande och stigande sortering infördes delades valen upp i två `Spinners`, och därmed fick också en "OK"-knapp införas. En mer användarvänlig design skulle kunna vara att lägga alla möjliga kombinationer av alternativ i en vy, så att användaren bara behöver klicka en gång på den kombination som önskas.

Att använda `DialogFragment` för att visa tids- och datumväljare var i retrospekt ett dåligt beslut, främst då Androids Pickers ändå inte används. Det resulterade mest i lång, invecklad och onödig kod. Dessutom skapades en ny klass som ärvde från `DialogFragment` för varje plats där en tids- eller datumväljare används, då fragmentet behövde hålla kontakt med aktiviteten det skapats från genom att föra in just en sådan aktivitet som parameter i fragmentets konstruktor. Istället för att kalla på en dialogruta med hjälp av ett fragment, hade man till exempel kunnat skapa en vanlig aktivitet i form av en dialogruta såsom gjorts bland annat i grafalternativen, och på så sätt kontrollerat rutans utseende. Även metoden för att kontinuerligt öka eller minska ett värde då användaren trycker ned en knapp är i dagsläget lång och invecklad. Skulle båda dessa problem lösas kan koden för tids- och datumväljarna kortas ned avsevärt.

8.4 Framtida arbete

Projektgruppen är öppen för fortsatt utveckling av applikationen, troligtvis av en subgrupp av projektgruppen eller genom överlämning till tredje part. Gruppen ser att det finns ytterligare arbete att utföra på applikationen innan kommersiell lansering.

8.4.1 Säkerhet

I nuläget skickas kvittoavbilder till servern i en form som inte direkt går att översätta till bilder om man inte vet hur protokollet definieras, men i framtiden skulle en säkrare lösning implementeras med säker kryptering. Även data som returneras från servern skulle krypteras för att säkerställa att användarens data ej går att komma åt vid avlyssningsförsök. Detta skulle göras innan applikationen släpps publikt då det var ett av kraven som användare prioriterat högt i enkäten.

I framtiden skulle även den data som lagras i den interna databasen krypteras för att skydda den data som lagras på enheten. Detta är menat att skydda applikationens integritet mot andra applikationer och manuella intrångsförsök.

8.4.2 Förslag på ytterligare funktionalitet

Förutom den funktionalitet som föreslagits och ej implementerats uppstod under projektets genomförande ett antal idéer rörande ytterligare funktionalitet till applikationen. Bland annat föreslogs att två användare kan ha gemensam ekonomi, och att det därför skulle vara intressant att kunna synkronisera databasen av kvitton mellan olika enheter. Det skulle även vara intressant att kunna anpassa parsningen av ett kvitto efter en mall för att få bättre resultat. Då många affärer har en intern standard på kvitton skulle användaren kunna uppge vilken affär som besökts för att applicera denna mall. Ytterligare funktionalitet skulle kunna utöka den existerande inriktningen mot privatekonomi.

Det finns även möjlighet till utveckling av existerande funktioner. Till exempel är det intressant att kunna visa två dataset samtidigt i en graf, och visst arbete har påbörjats med detta. En funktion som redan är delvis implementerad och som vore bra att ha är möjligheten att söka efter kvitton. Det finns stöd för det i databasen men det har inte lagts in i applikationen än.

9. Slutsats

Resultatet av arbetet fyller syftet, beskrivet i *1.1 Syfte*, då en färdig och fungerande alfaversion av applikationen har utvecklats. Alla krav listade under *2.3.1 Första prioritet* och *2.3.2 Andra prioritet* har slutförts, men även en del av *2.3.3 Tredje prioritet*. Detta planerades inte från början då det var tänkt att en prioritetsnivå skulle färdigställas per iteration och två iterationer har genomförts.

Vidare genomfördes en kravundersökning för att analysera målgruppens behov, vilket användes vid färdigställandet av kraven. Med hjälp av detta blev det tydligt vad studenter tycker är viktigt i en applikation för kvittoskanning. Även ett avslutande användartest genomfördes för att utvärdera applikationen, och överlag var responsen god.

Under planeringsfasen togs beslutet att undersöka Tesseract som ramverk för optisk teckenigenkänning (se *5.4 Optisk teckenigenkänning (OCR)*). Detta har underlättat arbetet då det gav ett bra resultat och tiden har kunnat läggas på optimeringar och övrig funktionalitet istället för utveckling av en egen teckenigenkänningsmotor.

Under arbetets gång har funktionalitet som ej planerats från början anammats men inte hunnit färdigställas. Därmed finns stora möjligheter för vidareutveckling av applikationen, vilken troligtvis kommer skötas av gruppen.

Referenslista

4ViewSoft Company (2012) *AChartEngine*. http://www.achartengine.org/ (2013-02-23).

ABBYY (2013a) *ABBY*. http://www.abbyy.com/ (2013-05-16).

ABBYY (2013b) *Mobile description*. http://www.abbyy.com/mobileocr/ (2013-05-16).

ActionBarSherlock (2012) *ActionBarSherlock*. http://actionbarsherlock.com/ (2013-05-06).

Android (2013a) Licens. *Android developers*. http://source.android.com/source/licenses.html (2013-05-06).

Android (2013b) Dashboards. *Android Developers*. http://developer.android.com/about/dashboards/index.html (2013-02-12).

Android (2013c) Welcome to Android. *Android Open Source Project*. http://source.android.com/ (2013-02-23).

Android (2013d) SDK. *Android Developers.* http://developer.android.com/sdk/index.html (2013-02-16).

Android (2013e) Content Provider. *Android Developers* http://developer.android.com/guide/topics/providers/content-providers.html (2013-02-18).

Android (2013f) Saving Data in SQL Databases. *Android Developers*. http://developer.android.com/training/basics/data-storage/databases.html (2013-05-08).

Android (2013g) Design. *Android Developers.* http://developer.android.com/design/index.html (2013-03-13).

Android (2013h) User Interface. *Android Developers*. http://developer.android.com/guide/topics/ui/index.html (2013-04-22).

Android (2013i) TimePicker. *Android Developers*. http://developer.android.com/reference/android/widget/TimePicker.html (2013-05-13).

Android (2013j) View. *Android Developers*. http://developer.android.com/reference/android/view/View.html (2013-04-20).

Android (2013k) Canvas and Drawables. *Android Developers.* http://developer.android.com/guide/topics/graphics/2d-graphics.html#on-view (2013-05-08).

Android (2013l) MessageDigest. *Android Developers*. http://developer.android.com/reference/java/security/MessageDigest.html (2013-05-10).

Android (2013m) Preference package. *Android Developers.* http://developer.android.com/reference/android/preference/package-summary.html (2013-05-08).

Android (2013n) SharedPreferences. *Android Developers*. http://developer.android.com/reference/android/content/SharedPreferences.html (2013-05-09).

Android (2013o) Data-Storage. *Android Developers*. http://developer.android.com/guide/topics/data/data-storage.html#pref (2013-05-09).

Android (2013p) AsyncTask. *Android Developers.* http://developer.android.com/reference/android/os/AsyncTask.html (2013-05-13).

Android (2013q) NetworkOnMainThreadException. *Android Developers.* http://developer.android.com/reference/android/os/NetworkOnMainThreadException.html (2013-05-16).

Barnum, C. (2011) *Usability Testing Essentials - Ready, Set...Test!*. Boston: Elsevier Inc.

Bhaskar, S. och Lavassar, N., Green, S. (2010) *Implementing Optical Character Recognition Recognition on the Android Operating System for Business Cards*. Stanford: Stanford University.

Björkholm T. och Brattberg H. (2010) *Prioritera, fokusera, leverera : din snabbguide till Lean, Agile, Scrum och XP*. Stockholm: Vulkan.

Breuel, T. (2008) The OCRopus Open Source OCR System. I *Proceedings of SPIE*. 27 januari, 2008, San Jose.

Charland, A. och Leroux, B. (2011) Mobile Application Development: Web vs. Native. *Communications of the ACM*, vol. 54, nr. 5, ss. 49-53.

Chartdroid (2010) *Chartdroid*. http://code.google.com/p/chartdroid/ (2012-02-23).

Cheriet, M. et. al. (2007) *Character Recognition Systems - A Guide for Students and Practioners*. New Jersey: John Wiley & Sons, Inc.

Davis, A. (2005) *Just enough requirements management: Where software development meets marketing.* New York: Dorset House Publishing.

Eriksson, U. (2008) *Kravhantering för IT-system*. Malmö: Studentlitteratur.

Gamma, E. (2009) *JUnit*. http://junit.sourceforge.net/ (2013-02-23).

Git (2013) *Git*. http://git-scm.com/ (2013-05-09).

Goadrich, M. och Rogers M. (2012) A hands-on comparison of iOS vs. android. I *Proceedings of the 43rd ACM technical symposium on Computer Science Education*; 29 februari-3 mars 2012, Raleigh, s. 663.

Goldberg, D. (1991) What Every Computer Scientist Should Know About Floating-Point Arithmetic. *Computing Surveys*. Mars 1991. http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html (2013-05-13).

Google (2010) Android 2.2 Compatibility Definition. *Compability program*. http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/en//compatibility/2.2/android-2.2-cdd.pdf (2013-05-15).

Google (2012) *Tesseract Historia*. http://tesseract-ocr.googlecode.com/svn-history/r719/trunk/doc/tesseract.1.html (2013-04-04).

Google (2013a) Bankdroid. *Bankdroid - Android-appar på Google Play*. https://play.google.com/store/apps/details?id=com.liato.bankdroid&hl=sv (2013-02-18).

Google (2013b) *Tesseract Open Source* http://code.google.com/p/tesseract-ocr/ (2013-04-04).

Herlitz, C. (2011) Forskaren får inte resignera inför samhällets »enkättrötthet«. *Läkartidningen*. http://www.lakartidningen.se/07engine.php?articleId=17172 (2013-02-02).

Kvittar (2013) *Kvittar*. http://www.kvittar.se. (2013-03-16).

Leffingwell, D. (1997) Calculating the Return on Investment from More Effective Requirements Management. *American Programmer*, vol. 10, nr. 4, ss. 13–16.

Menard, B. (2008) Optical Character Recognition. *Quality*, vol. 47, nr. 5, ss. 18-20.

Mo, S. och Mathews J. (1998) Adaptive, Quadratic Preprocessing of Document Images for Binarization. *IEEE transactions on image processing*, vol. 7, nr. 7, ss. 992-999.

Moraveji, N. och Soesanto, C. (2012) Towards Stress-less User Interfaces: 10 Design Heuristics Based on the Psychophysiology of Stress. I *CHI 2012*; 5-10 maj 2012, Austin, ss. 1643-1648.

OCR SDK (2013) *ABBYY OCR Cloud*. http://ocrsdk.com/ (2013-05-16).

OCRopus (2013a) *OCRopus*. http://code.google.com/p/ocropus/ (2013-04-12).

OCRopus (2013b) FrequentlyAskedQuestions. *OCRopus*. http://code.google.com/p/ocropus/wiki/FrequentlyAskedQuestions (2013-04-12).

OpenCSV (2011) *OpenCSV*. http://opencsv.sourceforge.net/ (2013-05-13).

Oracle (2013a) Primitive Data Types. *The Java tutorials*. http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html (2013-05-13).

Oracle (2013b) Class MessageDigest. *MessageDigest*. http://docs.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html (2013-05-13).

Oracle (2013c) *JavaMail*. http://www.oracle.com/technetwork/java/javamail/index.html (2013-05-13).

Otsu, N. (1979) A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, nr. 1, ss. 62-66.

Patel, C., Patel, A. och Patel, D. (2012) Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study. *International Journal of Computer Applications*, vol. 55, nr. 10, ss. 50-56.

Pham, A. och Pham, P. (2012) *Scrum in Action: Agile Software Project Management and Development*. Boston: Cengage Learning.

Recibo (2013) *Recibo*. http://www.recibo.se/ (2013-03-16).

Riehle, D. (2000) *Framework Design: A Role Modeling Approach*. Zürich, Switzerland: ETH Zürich. Ph.D. Thesis, nr. 13509. ss. 1-2.

Rivera, J. och Van der Meulen, R. (2013) Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012. *Gartner*. http://www.gartner.com/newsroom/id/2335616 (2013-02-17).

Robotium (2012) *Robotium.* http://code.google.com/p/robotium/ (2013-02-23).

Sadun, E. (2011) *The iOS 5 Developer's Cookbook - Core Concepts and Essential Recipes for iOS Programmers.* Boston: Pearson Education, Inc. ss. 3-4, 11.

Sauvola, J. och Pietikainen, M. (2000) Adaptive document image binarization. *Pattern Recognition*, vol. 33, nr. 2, ss. 225-236.

Sharp, H., Rogers, Y. och Preece, J. (2009) *Interaction Design: beyond human-computer interaction.* 2nd editon. England, Chichester: John Wiley & Sons Ltd, ss. 298, 591.

Singh, T.R. (2011) A New Local Adaptive Thresholding Technique in Binarization. *International Journal of Computer Science Issues*, vol. 8, nr. 2, ss. 271-277.

Smith, R. (2007a) An Overview of the Tesseract OCR Engine. *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*. 23-26 september, 2007, Parana. ss. 629-633.

Smith, R. (2007b) Tesseract OCR Engine. *O'Reilly Open Source Convention.* 23-27 juli, 2007, Portland. ss. 6-7.

Sparakvittot (2012) *Spara kvittot*. http://www.sparakvittot.se/ (2013-03-16).

Tauschek, G. (1935). *Beading machine.* US2026329.

Tess4J (2013) *Tess4J*. http://tess4j.sourceforge.net (2013-05-20).

Trello (2013) *Trello*. https://trello.com/ (2013-05-16).

Whitaker, K. (2009) *Principles of Software Development Leadership : Applying Project Management Principles to Agile Software Development*. Boston: Course Technology / Cengage Learning.

Wiegers, K. (1999) *Software Requirements*. N.p.: Microsoft Press. (2013-02-14).

Zarra, M. (2005) A Java Programmer's Introduction to Objective-C: Memory Management. *Peachpit*. 23 september 2005. http://www.peachpit.com/articles/article.aspx?p=377302 (2013-05-06

Appendix A: Enkät till kravundersökning

A1: Enkät

Vi har planer på att utveckla en applikation till Android som på ett smidigt sätt ska digitalisera kvitton och kunna visa en enkel budget.
Vi vore otroligt tacksamma om du vill vara med och ge dina synpunkter om funktioner denna applikation skulle tänkas ha.
Om du lämnar din mail-adress så är du med i utlottningen av biobiljetter!

* Required

Mail-adress

Krävs om du vill vara med i utlottningen av en biobiljett

Har du en smartphone som stödjer användande av applikationer? *

En mobil som kör Windows Phone, iPhone, Android etc.

- Ja
- Nej

Har du använt dig av applikationer som hanterar din privatekonomi? *

Exempelvis en elektronisk kassabok, plånbok, budgetplanerare.

- Ja
- Nej

Vad skulle du tycka var viktigast om du använde en mobilapplikation inom privatekonomi? *

- Bra prestanda, den är snabb att använda
- Den ser bra ut, bra grafik
- Användarvänlig, smidig och enkel att använda
- Hög säkerhet

Vilken funktionalitet skulle du vilja ha med om du använde dig av en applikation inom budget/ekonomi? *

- [ ] Automatisk insättning av kvitton till programmet genom kameran.
- [ ] Ha ett kvittobiblioteket; kunna se, ändra eller ta bort kvitton.
- [ ] Säker identifikation, inloggning till applikationen.
- [ ] Kunna se inköp och budget i smidiga grafer.
- [ ] Kunna dela information till facebook etc.
- [ ] Manuellt kunna skriva in kvitton till programmet.
- [ ] Kunna koppla ihop data om inkomster/utgifter m.m. från bankkonto till applikationen.
- [ ] Frihet att kunna göra egna inställningar.
- [ ] Kunna exportera kvitton/data till datorn.

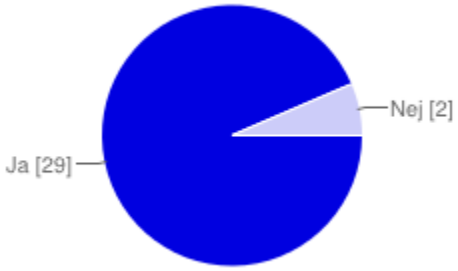
Eget förslag på funktioner:

Övrigt

Har du en egen kommentar, idéer kring en sådan här applikation. Vad tycker du vi bör tänka på?

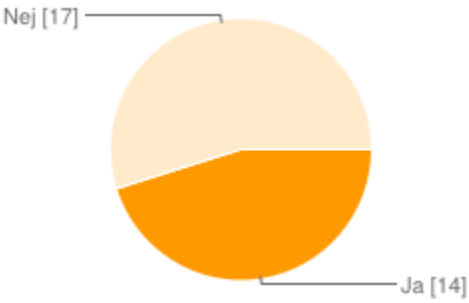
A2: Resultat av enkät

Har du en smartphone som stödjer användande av applikationer?



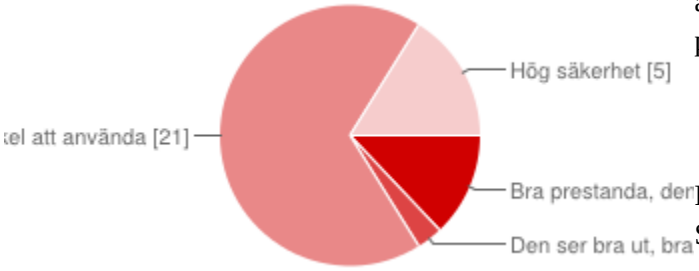
Ja 94%, Nej 6%

Har du använt dig av applikationer som hanterar din privatekonomi?



Ja 45%, Nej 55%

Vad skulle du tycka var viktigast om du använde en mobilapplikation inom privatekonomi?



Prestanda 13%
Ser bra ut, bra grafik 3%
Användarvänlig 68%
Säkerhet 16%

Vilken funktionalitet skulle du vilja ha med om du använde dig av en applikation inom budget/ekonomi?

Automatisk insättning av kvitton till programmet genom kameran.	**10**	32%
Ha ett kvittobiblioteket; kunna se, ändra eller ta bort kvitton.	**11**	35%
Säker identifikation, inloggning till applikationen.	**6**	19%
Kunna se inköp och budget i smidiga grafer.	**8**	26%
Kunna dela information till facebook etc.	**0**	0%
Manuellt kunna skriva in kvitton till programmet.	**10**	32%
Kunna koppla ihop data om inkomster/utgifter m.m. från bankkonto till applikationen.	**9**	29%
Frihet att kunna göra egna inställningar.	**9**	29%
Kunna exportera kvitton/data till datorn.	**10**	32%

Sammanfattning av förslag på funktioner

- Kunna se hur mycket man handlat och av vad
- Kunna exportera data till JSON-format
- Kunna se vad man har kvar på budgetposten för varje månad
- Få notifieringar om man går över budgetgränsen
- Skapa egna inköpslistor av de varor användaren lagt in från olika affärer
- Känna igen om ett kvitto kommer från t.ex. en restaurang och då automatiskt filtrera det som mat.

Övriga kommentarer:

- Alla kvitton har ju inte samma format så det behöver ju flexibilitet på det sättet att man kan hitta en mall för sitt kvitto utan att skriva om det till en annan kvittoformat. Om ni tar in kvitton via kameran så kanske ni behöver tänka på det här med.
- Bra idé men hur gör man det så lätt som möjligt? Vill inte hålla på med sånt här, det ska bara hända av sig själv. Då vore det bra!

- För min egen del tycker jag att det viktigaste med en sån här applikation är att den går sjukt snabbt och smidigt att använda (inga onödiga klick, det bästa vore om man direkt när man startade appen kom till kameran, och att man kan stänga ner appen direkt efter att ha tagit en snabb bild). Jag tycker även det är viktigt att kunna gruppera utgifterna i olika kategorier snabbt och enkelt.
- Lättviktsapplikationer är bättre än snygga applikationer.

Appendix B: Ursprunglig tidsplanering

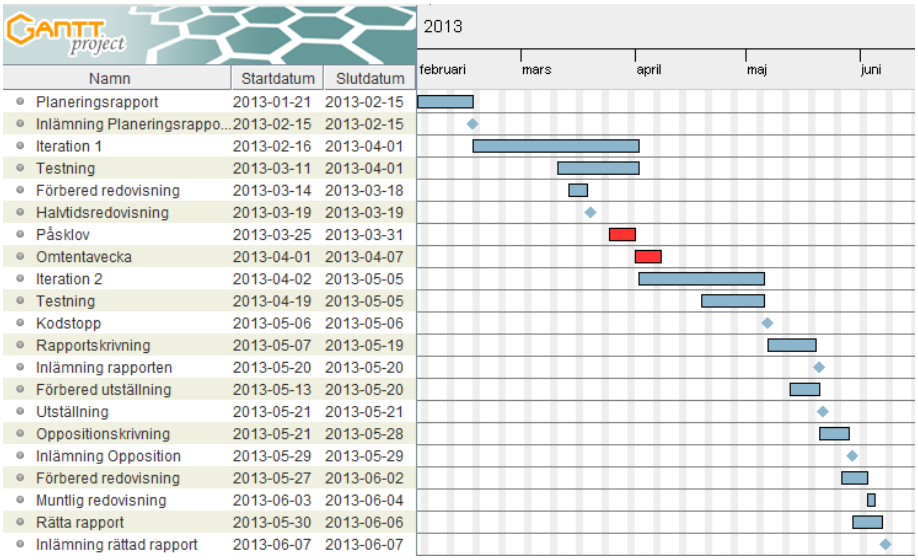
För att planera projekt har ett Gantt-schema använts. Detta är modellerat utifrån de inlämningar som ska göras samt milstolpar och rimliga tidsspann som gruppen beslutat.

Projektet inleds med skrivning och inlämning av en planeringsrapport. Under denna tidsperiod sker kravställning och planering av implementationen. Efter perioden sker implementationsfasen, och det är beräknat att gruppen hinner genomföra två iterationer med motiveringen att ej inkludera fler funktioner med bekostnad på produktens kvalitet. Målet är att det skall finnas en användbar prototyp vid slutet av var iteration. Om det visar sig vid ett senare skede att det finns tid till godo kommer en tredje iteration att övervägas, men överbliven tid kan även läggas på att finputsa den existerande funktionaliteten. Iteration 1 kommer att fokusera på de krav som ligger som *Första prioritet* samt utformning och implementation av gränssnittet, och iteration 2 kommer att fokusera på kraven under *Andra prioritet*.

I den senare hälften av varje iteration finns det tid avsatt för testning. Tanken är att testning skall pågå simultant med programmeringen hela tiden, men att planera in tid för det är en kraftfull påminnelse och motivering till att se till att det blir genomfört.

Innan varje redovisning och inlämning finns det tid till att framställa en presentation eller liknande. Detta innebär ej att ingen utveckling kommer ske i det skedet, utan snarare att den tillgängliga tiden kommer delas mellan framställning av produkt och presentation. Undantaget till detta är kodstoppet som inträffar 6 maj. Även om rapporten kommer behandlas under hela projekttiden är det viktigt att dedikera tid endast åt rapporten innan dess inlämning.

I schemat är påsk- samt omtentaveckan inlagda (visas i rött). Under dessa veckor finns inget arbete schemalagt. Gruppmedlemmar får arbeta med projektet på egen hand under denna tid, men det krävs ej. Detta gör att slutet av iteration 1 hamnar mitt i ledigheten, vilket kan kommas att ändras under projektets gång. Allt detta kombinerat kan ses i Gantt-schemat nedan.



Namn	Startdatum	Slutdatum
Planeringsrapport	2013-01-21	2013-02-15
Inlämning Planeringsrappo...	2013-02-15	2013-02-15
Iteration 1	2013-02-16	2013-04-01
Testning	2013-03-11	2013-04-01
Förbered redovisning	2013-03-14	2013-03-18
Halvtidsredovisning	2013-03-19	2013-03-19
Påsklov	2013-03-25	2013-03-31
Omtentavecka	2013-04-01	2013-04-07
Iteration 2	2013-04-02	2013-05-05
Testning	2013-04-19	2013-05-05
Kodstopp	2013-05-06	2013-05-06
Rapportskrivning	2013-05-07	2013-05-19
Inlämning rapporten	2013-05-20	2013-05-20
Förbered utställning	2013-05-13	2013-05-20
Utställning	2013-05-21	2013-05-21
Oppositionskrivning	2013-05-21	2013-05-28
Inlämning Opposition	2013-05-29	2013-05-29
Förbered redovisning	2013-05-27	2013-06-02
Muntlig redovisning	2013-06-03	2013-06-04
Rätta rapport	2013-05-30	2013-06-06
Inlämning rättad rapport	2013-06-07	2013-06-07

Appendix C: Dokument till användartester

C1: Testscenarion och uppgifter

Scenario 1
Utgångsläge: Startskärmen
Målläge: Kvittovy
Uppgift: Lägg till nytt kvitto genom att ta kort på det

- Beter sig crop på ett intuitivt sätt?

Scenario 2
Utgångsläge: Startskärmen
Målläge: Startskärmen
Uppgift: Ändra ett kvitto

1. Gå in i kvittot från Willys i februari
2. Lägg till att du köpt tre korvar för 30,8 kr
3. Ta bort majsen
4. Lägg in en rabatt på 5 kr
5. Ställ in att du betalade kontant
6. Gå sedan tillbaka till startsidan

- Automatisk ny rad: Många nya rader av misstag? Beter det sig som förväntat?
- Förstår de att de kan välja kategori?

Scenario 3
Utgångsläge: Starskärmen
Målläge: Kategoriskärmen
Uppgift:

1. Lägg till en kategori med namnet ”Godis” och valfri färg
2. Spara
3. Ändra namn på kategorin till ”Godsaker”
4. Ta bort kategorin ”Skor”

Scenario 4
Utgångsläge: Startskärmen
Målläge: Startskärmen
Uppgift: Titta på grafer

1. Visa en graf över inköpen
2. Ändra färg på graflinjen till purpur
3. Visa stapeldiagram istället för linjediagram
4. Titta närmare på mars
5. Gå tillbaka till start

- Zooma/skala om; är det tydligt hur man kan göra det?

Scenario 5
Utgångsläge: Startskärmen
Målläge: Startskärmen
Uppgift:

1. Ta bort Kvittot från Willys i mars och kvittot från Willys i februari
2. Sortera om listan efter affärsnamn
3. Ställ in fet text på kvittotiteln

- Vad förväntas av long press?

C2: Förberedande frågor

Datum:
Namn: _____
Ålder: 15-20 20-30 31-40 40+
Kön: Man Kvinna
Högerhänt Vänsterhänt

Har du en smartphone?
[ ] Ja
[ ] Nej

Om ja, vilket operativsystem använder den?
[ ] Android
[ ] iOS
[ ] Annan: _____

Vilken sorts applikationer har du använt? Kryssa för alla som stämmer.
[ ] Spel
[ ] Navigering
[ ] Ekonomi
[ ] Sociala nätverk
[ ] Övrigt: _____

Hur länge har du använt en smartphone?
[ ] 0-6 månader
[ ] 7-12 månader
[ ] 1-2 år
[ ] Mer än 2 år

Vad tycker du är viktigast i en applikation för privatekonomi?
[ ] Bra prestanda
[ ] Ser bra ut
[ ] Lätt att använda
[ ] Hög säkerhet

C3: Scenarioformulär

Uppgift:
Betygssätt hur lätt uppgiften var:

Lätt				Svår
1	2	3	4	5

Hur lång tid tog det att utföra uppgiften?

Kort tid				Lång tid
1	2	3	4	5

Hur troligt är det att du skulle använda dig av denna funktionaliteten?

Inte så troligt				Väldigt troligt
1	2	3	4	5

C4: Avslutande frågor

Överlag, hur lätt var det att använda applikationen?

Lätt			Svårt	
1	2	3	4	5

Överlag, hur nöjd är du med applikationen?

Inte nöjd			Nöjd	
1	2	3	4	5

Hur lätt var det att hitta vad du letade efter?

Lätt			Svår	
1	2	3	4	5

Hade du rekommenderat applikationen till en vän?
Ja Kanske Nej

Hade du använt applikationen?
Ja Kanske Nej

Vad skulle vara ett lagom pris för applikationen?

Hur lång tid tog det tills du kände dig säker i att använda applikationen?

Vad var din favoritdel med applikationen?

Vilken del av applikationen tyckte du minst om?

Övriga kommentarer

Appendix D: Pseudokod för parsning av text från teckenigenkänning

```
List productStrings;
for each row of the receipt {
     if row matches productRegex {
          add row to productStrings
          move to next row
     }
     if store not set {
          if row contains existing store in database {
               set matching existing store as receipt store
          }
     }
     if date not set {
          if row matches dateRegex {
               set matching date as receipt date
          }
     }
     if time not set {
          if row matches timeRegex {
               set matching time as receipt time
          }
     }
     if (none of store, date or time is set) and (not on last row) {
          if next row matches productRegex {
               add (row + next row) to productString
               skip next row in for loop
          }
     }
}
// Lägg till raderna i productStrings som produkter på det nya
// kvittot och sätt affär, datum och tid för kvittot.
```