MyBar – Reflektion

[Introduktion 2](#_Toc338523953)

[Androidutveckling 2](#_Toc338523954)

[Eclipse 2](#_Toc338523955)

[GitHub 2](#_Toc338523956)

[Git 2](#_Toc338523957)

[Egit 3](#_Toc338523958)

[Git Shell 3](#_Toc338523959)

[Problem och utmaningar 3](#_Toc338523960)

[Scrum 3](#_Toc338523961)

[Problem och utmaningar 3](#_Toc338523962)

[Kodstruktur 4](#_Toc338523963)

[se.turbotorsk.mybar 4](#_Toc338523964)

[se.turbotorsk.mybar.controller 4](#_Toc338523965)

[se.turbotorsk.mybar.model 5](#_Toc338523966)

[Databas 5](#_Toc338523967)

[Content provider 5](#_Toc338523968)

[JSON 5](#_Toc338523969)

[Extern kod 5](#_Toc338523970)

[Gruppdynamik 5](#_Toc338523971)

[Möten 5](#_Toc338523972)

[Sharepoint 6](#_Toc338523973)

[Mailflöde 6](#_Toc338523974)

[Handledning 6](#_Toc338523975)

[Testning 6](#_Toc338523976)

[Sammanfattning 6](#_Toc338523977)

# Introduktion

Vi tycker alla att det är roligt med en praktisk kurs som bryter av de teoretiska. Men det var mycket att lära sig samtidigt, när man inte programmerat i Android tidigare. Versionshanteringsprogram har vi aldrig använt på detta sätt innan och en hyfsat (för oss) ny utvecklarmiljö att skriva kod i. Det kom mycket på samma gång, och det blev inte lättare när man sedan skulle implementera massa funktioner i ett program man vill göra. Vi förstår meningen och tanken med att vi skulle göra en applikation för Android eftersom många inte gjort det innan, och det är helt klart ett bra sätt att ha en kurs på eftersom man lär sig något nytt plus att de flesta ligger på samma nivå. Men mycket tid gick åt till att leta reda på hur man ska skriva koden och rota igenom utbudet av features och liknande som man vill/kan ha i programmet.

Att välja projekt var rätt lätt, alla ville ha något som i våra ögon var användbart för oss. Något som man kan använda även utanför skolan, som förhoppningsvis sträcker sig till en bred målgrupp. Det svåra var att välja vilka features och liknande vi skulle ha när projektet började vilket försvårade arbetet med SCRUM (det kommer en del angående SCRUM senare) eftersom man inte visste vilka möjligheter som fanns inom ämnet och hade svårt att sätta upp delmål.

# Androidutveckling

Stuff om androidutveckling här…

# Eclipse

På Lindholmen blev vi rekommenderade att bara använda BlueJ i stället för Eclipse, detta för att lära oss grunden till OOP språk så som Java då våra första kurser var i ren C och assembler. Ett byte till Eclipse som utvecklingsmiljö gjorde vi helt enkelt ville använda de alla andra använde sig av.

Inlärningskurvan var till en början flack men efter många timmars arbete i detta IDE lärde vi oss mer och mer. Kortkommandon som var nyttiga att kunna vi fick veta i 7:e veckan av vårt arbete. Att kunna formatera text och så vidare underlättade väldigt mycket i slutskedet av projektet då vi lade ner tid på att hantera violations. Även javastandarder samt androidstandarder med längd på kod raderna var innan helt okänt, bland mycket annat. Vi skulle gärna se att ni informerade oss om detta i ett tidigare skede, att det finns standarder vi ska följa. Mer om detta i senare rubriker.

# GitHub

Anledningen till varför just GitHub valdes som versionshanteringsprogram var i vårat fall att vi tidigare inte använt oss utav något versionshanteringsprogram (Dropbox räknas knappast). Så eftersom erfarenheter inom sådant inte fanns så tyckte vi att det bara var att köra på det som flest tyckte verkade bra. I efterhand tycker vi att det var ett bra val, då vi tycker att det inte varit några större problem med att använda det när man väl kunde grunderna. Däremot tog det ett tag i början att lära sig det. Men det senare är en väldigt återkommande sak i detta projekt (allt blir ju lättare när man lär sig det).

# Git

Här kommer bristen på erfarenhet även också in. Vi tyckte att det verkade svårt att skriva in alla kommandon i ett terminalfönster, och hörde oss därför runt om det fanns något man kunde iplementera direkt in i Eclipse som vi kodade i. Egit dök upp rätt så tidigt och även där (som i fallet med Github) körde vi på första bästa. Även i detta fall är vi nöjda med valet. Lätt att använda när man väl lärt sig grunderna, vilket dock krävdes några veckor att komma in i.

Ett fel vi gjort i Git nu mot slutet är att vi har commit:at våra editerbara filer. Vår fokus låg på att knyta ihop projektet mer än att hantera text på optimalt sätt. Anledningen till att vi gjort på detta sätt är för att vara säkra på att det inte blir några krockar om två eller fler personer editerat samma fil. Optimalt är ändå att sitta tillsammans med t.ex en projektor och spåna fram texten. Då undviker man krockar samtidigt som man får en högre kvalitet på texten.

## Egit

Majoriteten av gruppen har använt sig av tilläget Egit till Eclipse för att hantera vårat Git-repo. Detta har fungerat bra men var inte helt användarvänligt, och tog tid att lära sig. Dock är Egit ett väldigt smidigt verktyg för den som inte vill lära sig Git Shell.

## Git Shell

Två av oss använde Git Shell istället för Egit. Verktyget har fungerat väl men det kan t.ex. uppstå kollisioner; vid ett sånt tillfälle är Git Shell inte optimalt för nybörjare som oss.

Däremot var det inga problem att t.ex. byta branch, push, pull, commit, och diverse simpla uppgifter som vi primärt använt oss av.

## Problem och utmaningar

# Scrum

Ingen i vår grupp hade tidigare hört talas om varken SCRUM eller Vatenfallsmodellen. Men vi kände att Vattenfallsmetoden riktade sig mer till större projekt, och valde därför SCRUM. Att veta ”allt” man skulle göra inom SCRUM och hur sprint planning, sprint backlog och product backlog skulle utformas tog ett bra tag. Vi var tvugna att uppdatera utformningen på våra dokument nästan varje vecka för att det alltid var något som såg fel ut på grund av missförstånd eller bristande alternativt fel information angående det. Men vi hade ändå våra möten i början av veckan och tog upp vad som skulle göras, och i slutet av veckan och summerade veckan som gått.

Undet projekts början insåg vi inte de många fördelarna som finns med att ha väldefinierade ”user stories” som innehåller information om hur programmet skulle fungera och hur. Detta är något som alla i gruppen i efterhand är överens om att vi skulle spara tid och resurer på att redan tidigt skapa dess dockument men öven att hålla dem updaterade under projektets gång.

## Problem och utmaningar

Då detta är första gången vi använt Scrum så har det varit svårt att se till att det finns User Stories för allt. Det finns några funktioner i vår applikation som tyvärr inte har en User Story. Detta är något vi skulle sett till att de har om vi skulle göra om projektet. Det känns där emot löjligt att åtgärda detta nu.

Vi känner dock att Scrum lämpar sig bättre i större/längre projekt eller i en företagsmiljö. Flera av dokumenten har vi skrivit endast för att följa Scrum-standard, mer än att vi har behövt dem. Det är nyttigt att ha följt Scrum, men det hade inte behövts i detta projekt.

# Kodstruktur

## se.turbotorsk.mybar

Detta paket innehåller kod som har med design och view att göra. Till exempel finns här vår MainActivity som är grunden till programmet.

Till en början ville vi använda oss av fragments, detta för att få möjligheten att snabbt kunne föflytta oss i applikationen. Vi lade ner extremt mycket tid på att försöka få detta att fungera men misslyckades med det. Efter mycket om och med bestämde vi oss för att istället använda oss av en tab layout som är föregångaren till swipe. Det blev då mycket enklare att komma vidare i applikationsutvecklandet och kunde lägga tid på annat.

Då man använder sig av tabs så behöver man inte använda fragments, det var just denna del vi hade svårt med. Vi hade alltså väldigt gärna velat använda fragments men tyvärr fick vi inte detta att fungera. Självklart är detta ett perfekt fall för framtida refaktorering.

Under de första veckorna blev vi tvungna att läsa på extremt mycket för att får en uppfattning på hur layouter fungerar och hur man kopplar dessa till en Android applikation. Till en början utecklade vi många GUI-funktioner själva som vi sedan insåg att de redan fanns i det API som Android tillhandahåller.

Därför blev det många branches som skapades utan att de har blivit merge:ade in i vår master. Vi hoppas att ni har dett i åtanke.

Designmässigt är vi nöjda med vår applikation och tycker att den ser bra ut. Där emot finns det självklart mer att göra, men så är det alltid. Valet till att använda oss av många listor är självklart med tanke på vad vi vill uppnå med vår applikation. Vi märkte snabbt att en vanlig lista blev för tråkig och simplel, vi valde däröfr att göra vår egna. Vi utvecklade ett eget listelement (rowlayout) för att få den att se ut som vi ville, med bilder, rating, ingredienser med mera.

Just eftersom att vi använde oss av en snyggare listvy för våra drinkar, än den som är standard, så blev vi tvungna att utveckla en egen adapter för att lättare hantera den data en drink innehöll. I ett senare skede fick vi också göra en adapter för ingredienser som är snarlik den för drinkar.

Att skapa en layout och adaptrar till dessa var extremt tidsödande och krävde stor förståelse för hur Androids GUI hanteras. Vi känner inte riktigt att den research vi gjort för att få en snygg layout speglas i koden då man nu kan tycka att det var enkel kod. Dock tyckte vi verkligen inte detta de första veckorna!

En annan detalj i applikationen vi hade problem med var options-menyn (den som visas i form utav tre stycken kvadrater staplade på varandra). Vi läste på mycket om hur det fungerar och hur menyerna inuti skulle skapas, och fick fram ett resultat efter några dagar. Senare insåg vi att den visades på olika ställen/sätt beroende på vilken telefon man använde, eller om man körde i emulator på datorn. Detta problem tog även det lång tid att lösa, men till sist kunde vi göra så att den i alla fall visade sig på allas telefoner och även i emulatorn. Var någonstans den däremot visas är fortfarande ett mysterium som vi inte kunnat åtgärda. Därför ser det fult ut på vissa telefoner sominte använder sig utav en ”fysisk” options-menu-knapp på telefonen, utan befinner sig i en grå rad längst ner på telefonens skärm.

Vi (Adam, Mathias B och Viktor) blev tvungna att lägga ner mycket tid på att försöka förstå hur layouter byggs upp innan vi ens kunde göra något konkret. Det vi trodde skulle vara enkelt blev istället en av våra största utmaningar.

Att förstå java kod är en sak, det har vi gjort tidigare, förstå xml är en helt annan. Att sedan väva samman dessa två standarder var inte det lättaste.

Det krävdes många timmars tålamod innan vi förstod hur man använde sig av R filen ordentligt och på så vis kunde binda xml filer till kod.

## se.turbotorsk.mybar.controller

## se.turbotorsk.mybar.model

Vi hade redan i projektes inledning en gankska tydlig idé på hur vi skulle hantera data. Denna idé gick ut på att ladda in data från en extern databas dirrekt med SQL när porgrammer installerades eller startade vid första tillfället och sprar denna data lokalt i en fil.

När vi underökt detta närmare insåg vi att detta inte var någon optimal desing. Det medförst säkerhetsrisker och kunde göra appen kännslig för uppdateringar av SQL. Men även problem när det gällde uppdateringar av databas och när nya drinkar och ingridienser skulle läggas till.

Genom att använda ett skickt mellan applikation och databsen kunde vi bygga bort säkerhetsriskerna. Detta skickt består av att vi låter PHP skapa en HTML-fil som innehåller en JSON struktur som representerar Drink och Ingrediens datan. Detta gör att applicationen inte längre har någon möjlighet att utföra förfrågningar direkt mot våran externa databas. Vi kan även göra så att sidan är statisk under vissa perioder och då minska belastningen på databasen.

Lokalt så valde vi att använda den inbyggda databsene SQlite. Detta ger oss flera fördelar med...

Snabba SQL frågor iställer för itteration av listor...

Säker lagring... mot ex. Någon form av datastruktur i en lokal fil.

# Databas

## Content provider

MyBar var designad runt tanken att vi alltid är uppkopplade mot molnet. Därför är den lokala databasen tom vid start och en automatisk hämtning av data sker från vår externa databas över JSON network calls. Vi tänkte med automatik hämta systembolagets XML-fil med alla deras produkter och parsa det till vår remote data store. Den externa servern skickar en förfrågan till mobilklienterna om de vill uppdatera och en sync sker till den lokala databasen.

Efter granskning av sätten att prata med databaser i android API:t valde vi Content Provider framför att prata direkt med vår lokala data store. Med en Content Provider hade då andra mobilapplikationer kunnat använda våra tabeller och vi hade kunnat göra fler appar som använder MyBars Content Provider. Vi alla gillar att dela upp allting i mindre objekt som inte känner till varandra och att strikt följa MVC-modellen. Content Provider stämde överens med vårat sätt att tänka, samt fungera som ett extra lager mellan databasen och applikationen i övrigt. Vi ville stark undvika att prata direkt med raw SQL queries. Providern exponerar tabellerna på ett standardiserat sätt mellan applikationer och kontrollerar att krockar ej sker om flera appar sänder frågor samtidigt. Vi får även tillgång till funktioner vi annars inte haft som kopiering av komplex databasdata mellan applikationer. Vi slipper också att tänka på malicious SQL queries från andra applikationer då Providern tar hand om detta.

När vår applikation vill prata med vår data store får vi använda en Content Resolver för att hämta data med CRUD (insert, update, delete, query), men de är endast abstrakta och implimentationen av dessa hämtas ur Content Providern. Det var väldigt svårt att komma igång då jag var tvungen att lära mig nya främmande konkept och få en förståelse för hur Content Provider, Content Resolver och SQLite Database Helper objekten kommunicer med varandra.

Jag ångrar mig inte att vi skapade en Content Provider trots att det var mycket svårare än att gå den lätta vägen med att skapa en SQLite Database Helper och sända SQL queries direkt in i den. Därför att jag lärde mig väldigt mycket om Content Providers och hur testning av denna fungerar.

## JSON

## Extern kod

# Gruppdynamik

Under projektets gång har vi använt oss av flera olika sätt att kommunicera, hantera, administrera och utveckla på. Detta är något vi är väldigt nöjda med. Det har under projektetes gång inte varit några som helst problem att få kontakt med resten av gruppen. Vi har enkelt kunnat förmedla information och kod med hjälp av de verktyg vi har valt att använda oss av.

## Möten

Vi har varje söndag haft ett långt telefonmöte (VoIP) där vi varje vecka har gått igenom vad vi gjort, och vad vi ska göra för att komma framåt. Vi har också gått igenom de User Stories och ärenden vi haft. På dessa möten har vi också diskuterat till exempel layouter och hur kopplingar till databaser skall se ut. Vi har gemensamt kommit fram till beslut som sedan genomsyrat hela projektet.

Innan vi började programmera bestämde vi på ett av våra första möten att lägga ner mycket tid på att ta reda på information. Detta för att behöva göra om så lite som möjligt.

Under veckorna har vi träffats i grupprum för att utveckla och diskutera applikationen tillsammans, det är lättare att komma fram till beslut om man kan ses. Det är på dessa möten vi kommit fram till många avgörande beslut.

Vi har också haft möten med vår handledare, mer om dessa möten under rubriken handledning.

## Sharepoint

Under de första veckorna försökte vi oss på att använda Microsofts produkt Microsoft Sharepoint. Detta program används för att hantera dokument och projekt i allmänhet. Tanken var att vi med detta program skulle ha allt på en plats. Allt eftersom projektet fortskred så fasades Sharepoint ut. Vi kan än idag inte säga varför det blev så men mailflödet tog över kommunikationen och git tog över koden allt eftersom att vi lärde oss git ordentligt. Vi känner dock att detta är en produkt som är optimal vid större projekt så som på företag eller längre projekt på Chalmers.

## Mailflöde

Från dag ett skapade vi ett mailflöde som vi egentligen skulle använda som komplement till Sharepoint, detta tog dock över som kommunikationskanal då det blev extremt smidigt att kunna maila fram och tillbaka i till exempel telefonen. Flödet har använts frekvent under hela projekttiden och vi är nu uppe i drygt 300 mail. Detta innebär i sig att vi har haft bra kommunikation inom gruppen.

# Handledning

Vi känner till viss del att inlärningskurvan var extremt platt till en början och sedan brant på slutet. Vi känner nu att vi förstår en hel del. Något som vi verkligen skulle se från Er sida är att lägga mer krut på att möjligtvis ha en föreläsning om hur man utvecklar ett snyggt UI och till exempel sätter samman xml och java kod. Detta för att på ett snabbare sätt få in den information som krävs för att komma igång att koda. Visst, det finns guider på nätet och det var dit vi gång på gång blev hänvisade. Men ni som lärare måste ta ett större ansvar och ge oss relevant information i ett forum där man kan ställa frågor, en föreläsning är ett exemplariskt sätt att göra detta.

Det var många gånger man inte fick ett svar som hade någonting med frågan att göra. Det kändes som att vi inte fick ett svar på våra frågot för att handledaren helt enkelt inte hade ett svar. Det var många gånger vi blev hänvisade till internet och det är bra, till en viss grad. Det kändes väldigt oproffsigt även om det kanske är så att ni ser det som att vi lär oss mycket på att leta fakta. Vi skulle i sådana fall behövt någon vi skulle kunnat verifiera våra fynd. Detta måste finnas.

Vi skulle gärna se en mer genomtänkt handledning där vi faktiskt kan få hjälp med de krav ni har satt upp.

# Testning

## Robotium

Se separat bilaga.

## Content Provider

För att testa Providern extendade jag ProviderTestCase2 som förser oss med en MockContentProvider som imiterar en Content Provider. Då sker alla testar mot Providern i en isolerat Context. Hade vi i stället testat med den vanliga Content Providern hade oönskade ändringar skett i databasen, speciellt om inte testcasen gått igenom korrekt. Eftersom vi använder en MockedContentProvider behövde jag inte lägga till något i tearDown() utöver det ProviderTestCase2 gör. MockContentResolver klassen isolerar testerna helt från det riktiga innehållet. Inga ändringar i manifestet har heller varit nödvändiga.

# Sammanfattning

Todo…

Vi tycker kursens maillista har fungerat illa. Mindre än 50% i gruppen har fått mailen.