System Design Document

for

Grupp 24, Brädgård

1 Introduction

Detta dokument syftar till att redogöra för applikationens arkitektur med syftet att läsaren ska ges en god inblick i dess uppbyggnad och funktionalitet. Applikationen har utvecklats för plattformen Android och är bakåtkompatibel till Android SDK 26.

1.1 Definitions, acronyms, and abbreviations

MVVM - Förkortning för designmönstret Model-View-View Model
Android OS - Operativsystem för android

Lifecycle - Android aktivitets livscykel

Uthyrare - Person som tillhandahåller en cykel för uthyrning.
Hyrestagare - Person som hyr en cykel.
Användare - Utgörs av både uthyrare och hyrestagare. En användare kan dessutom utgöra båda dessa typer.
Databas - extern lagring av information/data.

STAN - applikation som analyserar struktur i java-applikationer.

Instans - objekt som existerar oftast när applikationen körs.

API - Application Program Interface, förenklar kommunikation med andra program.

GUI - Graphical User Interface (Grafiskt användargränssnitt).

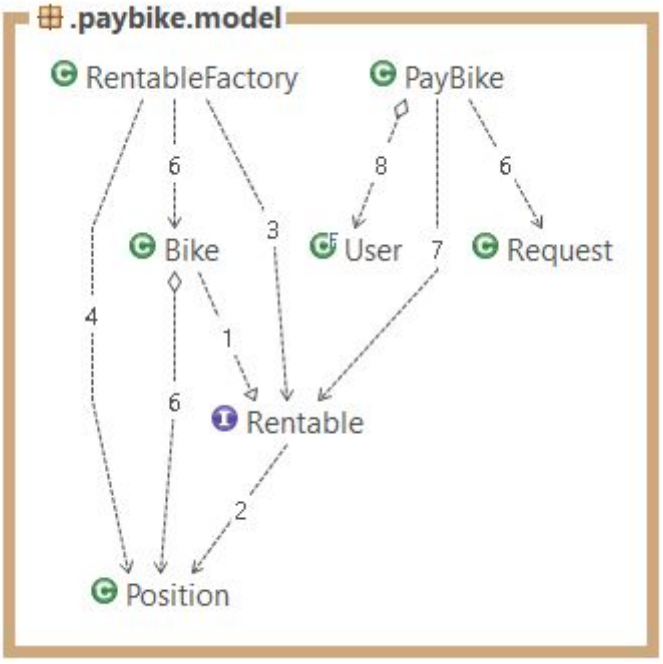
2 System architecture

Vår applikation använder en Model-View-ViewModel architektur kombinerad med Repository-mönstret. Varje viewmodel har en repository för att hämta den information som önskas av dess motsvariga activity från modellen, repository i sin tur är medlare mellan modell och den externa databas vi använder och ser till att dessa stämmer överens. Modellen som representerar domänen är oberoende av både viewmodels och repository.

2.1 Subsystem decomposition

Describe in this section each identified system component (that you have implemented).

2.2 Model



Modellen är ansvarig för att hålla applikationens grundläggande logik och tillfällig data. Här hålls de hyrbara objekt, applikationens nuvarande användare och hyrförfrågningar som sedan presenteras visuellt av android aktiviteterna. Modellen vill hållas oberoende från resterande system-komponenter.

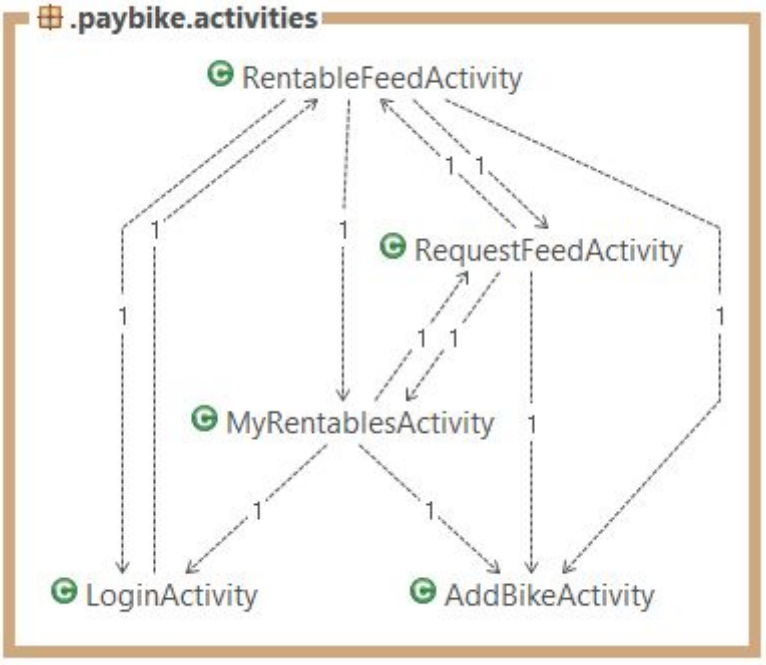
Följande klasser går att hitta i modellen:

- Bike: Klass som representerar cykel objekt, innehåller information om cykeln själv, position och vem dess ägare är.
- Rentable: Ett interface som är mall för alla hyrbara objekt. Tanken är att bygga den så fler klasser än "bike" ska kunna implementeras vid behov. T ex har applikationen i

framtiden möjligheten att även erbjuda hyrning av användbara tillbehör t ex hjälmar och annan säkerhetsutrustning eller stödhjul.

- User: Användarklassen, den ska innefatta all väsentlig information om användaren, bl a email, användarnamn och lösenord.
- Request: Klassen representerar det förfrågans-objekt användare har möjlighet att skicka vid intresse av att hyra specifik cykel.
- Position: Modellerar en svensk adress utifrån stad, postkod och gata. Landet sätts i nuläget till Sverige i och med att den enbart hanterar svenska adresser.
- RentableFactory: Gömmer logiken bakom skapandet av hyrbara objekt för klienten och öppnar även denna upp för utökning av framtida objekt som behöver skapas.
- PayBike: Singleton som håller alla rentables, request och nuvarande användaren.

2.3 Activities



Activity-klasserna håller användargränssnittet och representerar därmed View i MVVM. Med dessa visas interagerbara och synliga objekt på användarens android enheten. Eftersom en aktivitet kan pausas eller avslutas när som helst utav användaren eller android-systemet bör dessa inte hålla data utan istället hämta denna från en känd viewmodell. Bland annat ser detta till att Seperation of Concern principen uppehålls då Activities eller Fragments ej längre behöver hålla annat än logik gällande UI interaktion eller android-specifika uppgifter. Både activities och fragment har korresponderande .xml filer som håller informationen till deras visuella objekt. Objekten hanteras sedan av metoder i aktiviteten/fragmentet.

2.4 ViewModels

Var activity använder en viewmodel för att kalla på andra komponenter och framför förfrågningar att ändra/lägga till/ta bort data. Viewmodellerna känner ej till aktiviteterna och på så sätt förlorar inte användaren data ifall enhetens OS eller användare avbryter motsvarande aktivitet. Viewmodellerna har en repository som de med hjälp av skickar och tar emot data från modellen, repositoryn ser även till att denna data från vyobjekten sparas i databasen. För att det ska gå smidigt och för att följa Dependency Inversion där moduler på hög nivå inte ska vara beroende av moduler på låg nivå används Repository-klassen av ViewModellerna istället för att dessa ska vara direkt beroende av modellen eller databasen.

2.5 Repository

Repository är klassen som sköter kommunikation mellan den externa databasen och den lokala modellen. Vid start ber huvud-aktivitetens viewmodel att repositoryn uppdaterar modellen med den lagrade informationen tillgänglig i databasen och sätter den inloggade användaren som aktuell användare. När en lokal ändring sker i modellen uppdaterar repository databasen så att denna ändring är tillgänglig för alla kopplade enheter. Vår repository delegerar databas-specifika uppgifter till en trio "handlers".

2.6 Fragments

I Fragments hålls de fragment som är kopplade till specifika activities och används för att utöka layouten av en aktivitet. Fragment kan visas som en del av skärmen eller täcka hela.

2.7 Adapters

De adapters som används i vårt applikation, möjliggör många objekt av samma typ, fast med olika information, kan visas i form av lista eller rutnät. Istället för att Activities skulle lösa detta problemet, finns en mellanklass som "översätter" alla olika attribut till ett objekt som ges till activity-klassen som i sin tur visar alla olika objekt.

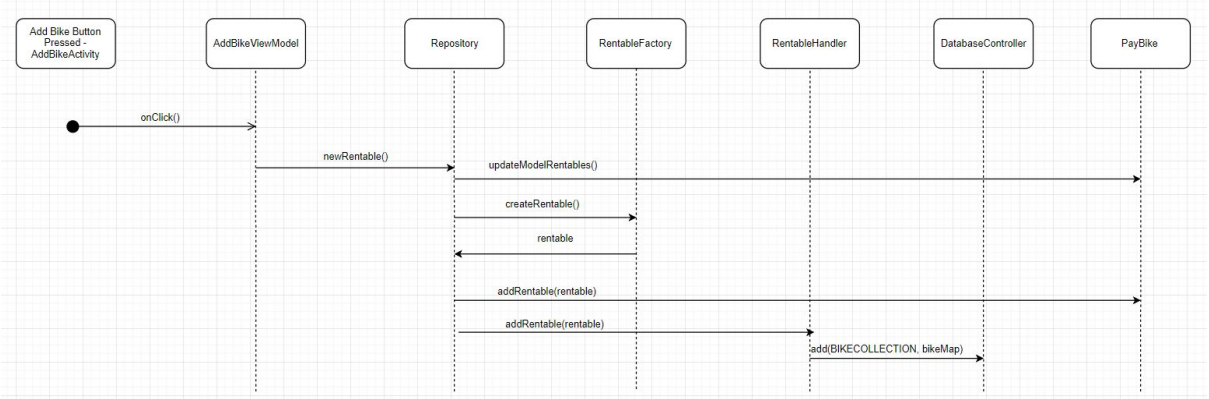
2.8 Handlers

Applikationens handlers används av repository för att utföra specifika databasrelaterade uppgifter gällande att konvertera information till och från javaklass samt lägga till och ta bort från databasen. Förutom UserHandler använder dessa databaskontrollern i deras uppgift.

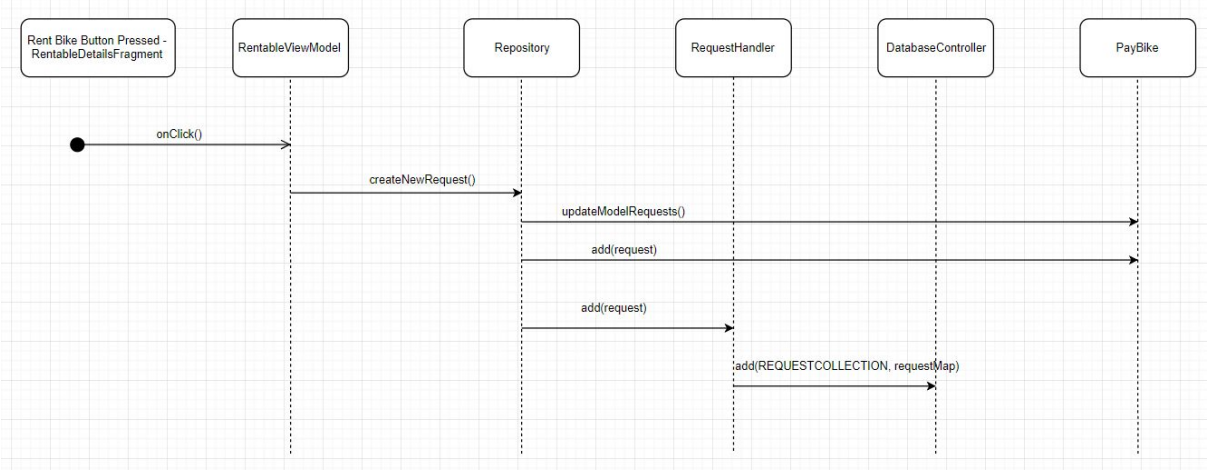
- UsersHandler: Hanterar alla metoder kopplade till Users och kommunicerar med databasen. Inloggning och skapande av nya användare sker med hjälp av UserHandler.
- RentableHandler: Hanterar hämtande och givande av objekt av typen Rentable från och till databasen. Främst "bikes" i nuläget men utökning för fler hyrbara objekt är möjlig.
- RequestHandler: Hanterar hämtning och tilläggning av Requests - förfrågan att hyra - i databasen.

Sequence diagrams

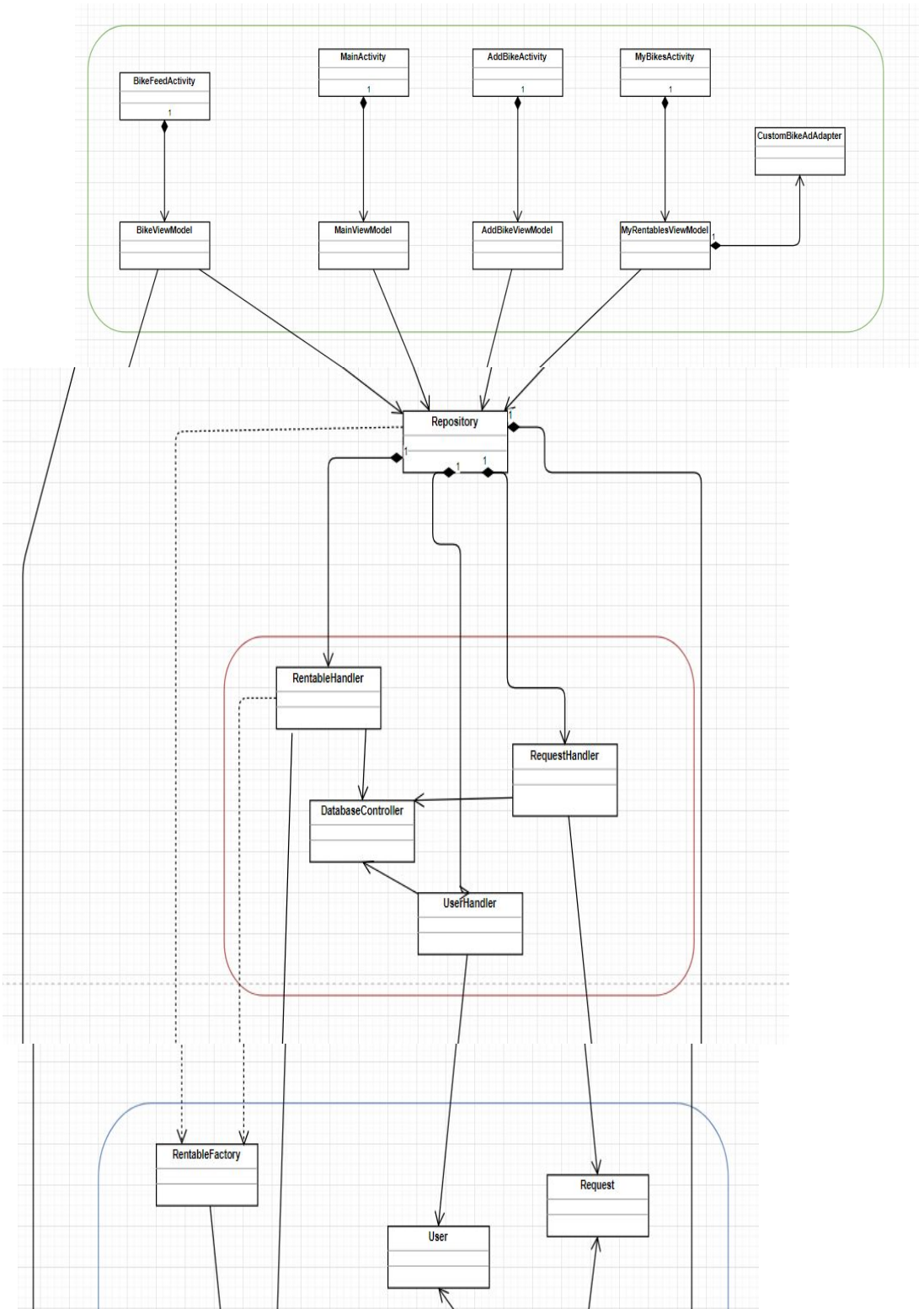
User adds a new rentable:



User sends request:



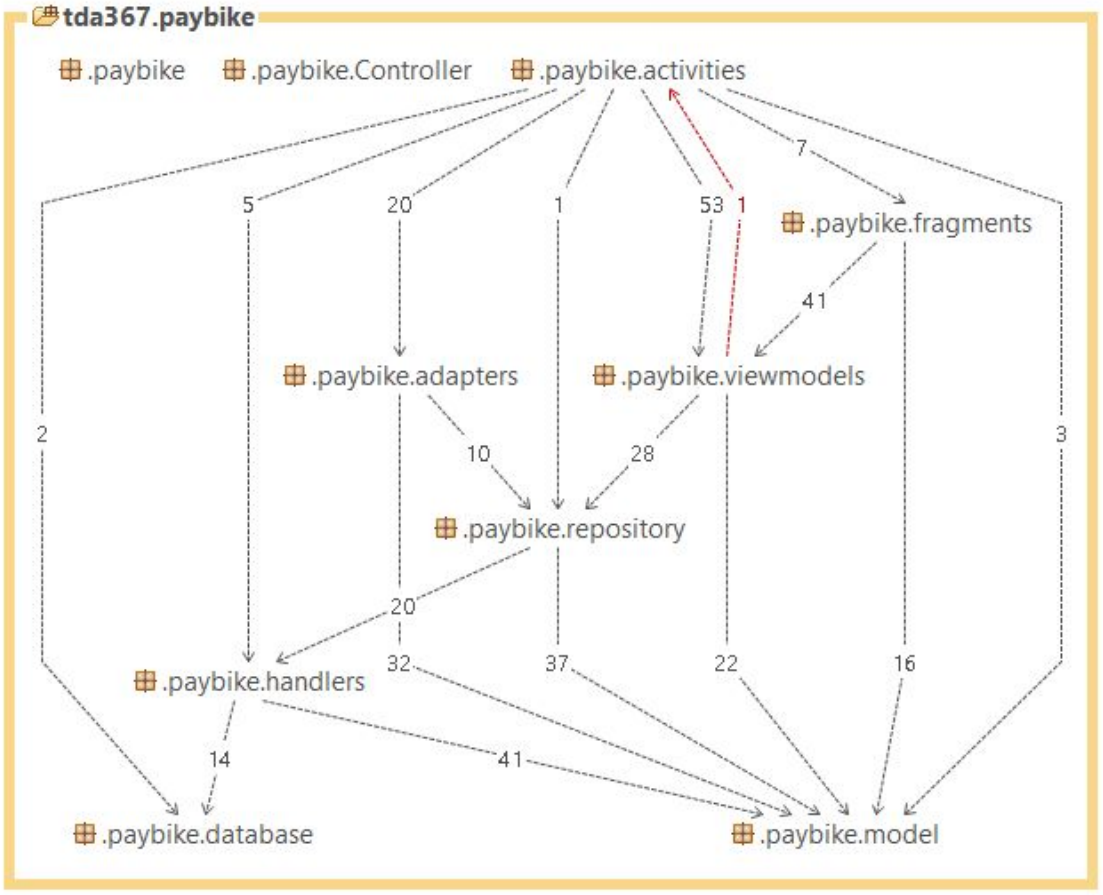
UML diagram över designmodell: (OBS - Position samt fragmenten saknas i denna bild)



Dependency analysis

Följande bild från STAN beskriver beroenden inom applikationens struktur.

(OBS: Notera cirkulärt beroende mellan activities och viewmodels, detta problem åtgärdas snarast)



3 Persistent data management - Pontus

Applikationen lagrar data med hjälp av Google's Firebase Storage. En instans av Firebase Storage initieras i *DatabaseController.java* och används sedan i metoden *uploadToStorage* i samma klass. *uploadToStorage* tar en sökväg som input, laddar upp filen sökvägen pekar på till Firebase Storage med ett slumpgenererat namn och returnerar en Firebase Task av typen URI. Denna Task kan sedan användas för att hämta information om filen samt en nedladdnings-URL.

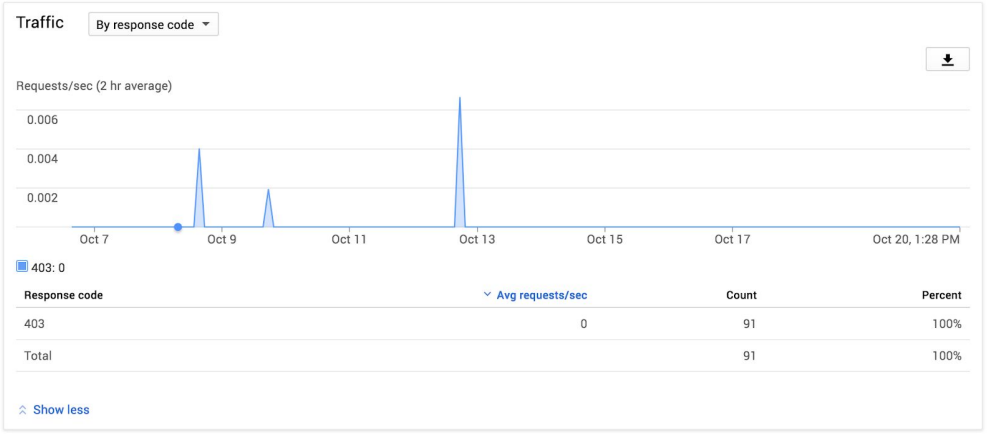
Uppladdningen till Firebase Storage används i *RentableHandler.java* när en rentable läggs till i databasen. I metoden *addRentable* hämtas sökvägen till en bild från en given Rentable och laddas upp till Firebase Storage via *uploadToStorage*-metoden i *DatabaseController.java*.

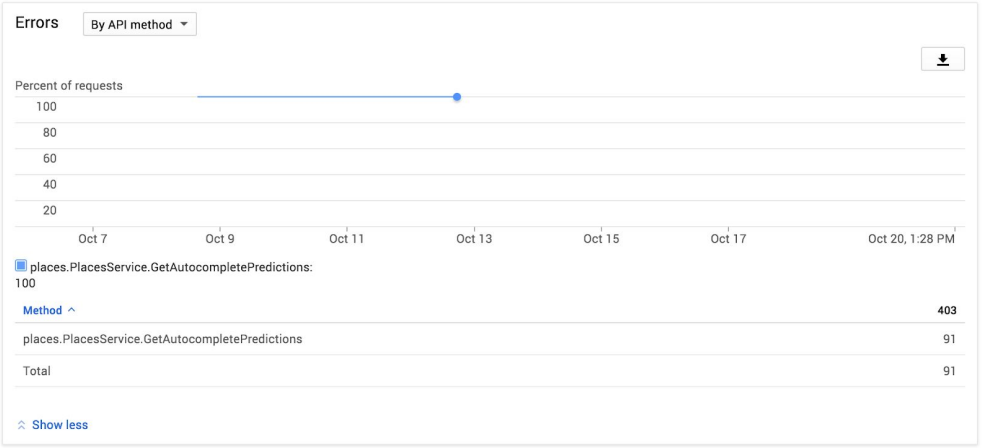
4 Access control and security - Pontus

Inget som hanteras, alla användare har samma åtkomstnivå.

Kända issues

För att lagra information om cyklarnas positioner blev lösningen att skapa en egen Position-klass för att modellera en position med gatuadress, postkod samt stad (med land Sverige som klassvariabel). Denna lösning innehåller en rad begränsningar då utformning av adresser varierar kraftigt världen över. För att fånga in den variationen hade ett betydligt mer komplext positionssystem krävts. Longitud och latitud hade även varit bra komplement för att enkelt kunna placera ut positionen på en karta vid vidare utveckling. Ett API som löser denna problematik är Googles API för Platser och Kartor- Google Places and Maps. Det innehåller över 100 miljoner platser jorden över och möjliggör hantering av Place-objekt som kan modellera samtliga av dessa platser. Tanken var att detta API skulle nyttjas vid skapande av nya cykelobjekt i AddBikeActivity för att tillåta autocompletion när användare angav en adress. Vid implementering uppkom dock komplikationer som inte löstes. Google mottog de anrop som gjordes från applikationen till deras servrar men skickade tillbaka Unknown Error som enda svarskod.





Skärmdumpar från Google Cloud Console. 91 anrop mottagna för GetAutocompletePredictions varav samtliga besvarades med Error.

Sökningar på forum såsom Stackoverflow samt Googles Github repo gav ingen förklaring till vad som var orsaken. Om denna applikation byggts för kommersialisering hade det varit nödvändigt att vända sig till ett API likt Googles för att hantera positions-information. I syftet för en prototyp fungerar dock den tillfälliga Position-klassen hjälpligt.

Ett annat problem som hanteras på ett mindre optimalt sätt är navigeringen mellan olika vyer i applikationen. Navigeringssystemet prioriterades bort i inledande utvecklingsfas vilket ledde till att applikationen utformades på ett sätt som inte gynnade navigering med en meny i nedre delen av GUIt. Därmed installerades istället en meny i samtliga Activities som behöver tillåta navigering. Nackdelen med detta tillvägagångssätt är att menyn måste definieras i samtliga Activities istället för att definieras separat och sedan med hjälp av referenser, återkomma i fler vyer.

Nuvarande applikation tillhandahåller ingen möjlighet att kontrollera om en cykel hyrs ut flera gånger vid samma tidpunkt. Det är upp till uthyraren att avgöra om en cykel är tillgänglig vid tidpunkten som en förfrågan gäller. Önskvärt hade varit om en kalender implementerats för samtliga cykel-objekt för att lagra information om tillgänglighet vid bokning.

Ytterligare en konsekvens som följer av det faktum att applikationen ännu är en prototyp, är avsaknaden av integrerad betalningstjänst. Priser förekommer trots detta i applikationen för att tydliggöra applikationens syfte - att möjliggöra uthyrning av cyklar. Applikationen syftar inte till att utgöra en plattform för gratis utlåning.

5 References