CHALMERS

Elbipo

- Ett IT-system för en fritt flytande elbilspool

Kandidatarbete inom Data- och informationsteknik

ANDREAS ANDERSSON CHRISTOFFER KARLSSON
LUDWIG KJELLSTRÖM NILS OLOFSSON

Institutionen för Data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2012
Kandidatarbete/rapport nr 2012:009

Abstract

This report describes the process behind the development of a system that primarily handles rentals of electric cars in a city like Gothenburg. In a Sweden there is approximately one car per household, where the majority are powered by fossil fuel in combustion engines, which leads to air pollutants and high noise levels. To mitigate these problems this project is aimed towards the creation of an electric carpool system and a software platform to support this system. The software platform consists of a webapplication with APIs and an Android application, which enables renting the cars in the car pool in an easy manner. Use cases were created from which the software was developed. The user interface has been chosen to be as simple as possible so that any user can feel comfortable to use and understand it. The quality of the system is assessed based on software testing criteria derived from ISO 9126-1:2001, which is an international standard for software quality. Finally conclusions were drawn on the choice of software development process and quality assessment process, and what should have been done differently in this project.

Keywords: Distributed system, electric cars, Android, Ruby on Rails, API, ISO 9126

Sammanfattning

Denna rapport beskriver processen bakom utvecklingen av ett system som hanterar bokningar av elbilar inom ett avgränsat område, exempelvis en stad som Göteborg. I Sverige finns ungefär en bil per hushåll, av dessa bilar drivs majoriteten av fossila bränslen som använder förbränningsmotorer vilket i längden innebär föroreningar i luften, samt en hög bullernivå. För att reducera dessa problem syftar projektet till att skapa ett system för en elbilspool och mjukvara för att stödja systemet. Den mjukvara projektet ämnar sig att utveckla består av en webbapplikation, en Androidapplikation samt ett API som på enkelt sätt möjliggör hyrning av elbilar. Med hjälp av framtagna användningsfall utvecklades mjukvaran. Gränssnittet designades för att vara enkelt och intuitivt, så att användare ska känna sig bekväma med användning av systemet oavsett användarnivå. Därefter bedömdes systemet utifrån mjukvarutestningskriterier baserade på ISO 9126-1:2001, vilket är en internationell standard för mjukvarukvalité. Projektet resulterade i en fullt fungerande webbapplikation där grundläggande funktionalitet är implementerad, även Androidapplikationen är färdigutvecklad om än med något mindre funktionalitet. Avslutningsvis drogs slutsatser om valet av utvecklingsmetod och utvärderingsmetod, samt vad som borde gjorts annorlunda i det här projektet.

Nyckelord: Distribuerade system, elbilar, Android, Ruby on Rails, API, ISO 9126

Innehållsförteckning

Ordlista

Användningsfall	eng. *Use case*. Ett scenario som beskriver hur ett system interagerar med sin omvärld.
API	*Application Program Interface*. Ett abstrakt sätt för att visa funktionalitet utan att avslöja själva implementeringen.
Distribuerade system	Flera parallellt arbetande system som gemensamt för att utföra en uppgift.
Fritt flytande bilpool	En bilpool där det är tillåtet att lämna en hyrbil på en godtycklig parkeringsplats.
HTTPS	*Hypertext Transfer Protocol Secure*. Krypterad HTTP-anslutning.
ISO 9126:2001	En standard för mjukvarukvalitetsutvärdering skapad av International Organization of Standardization.
Java	Objektorienterat programmeringsspråk.
JSON	*JavaScript Object Notation*. Strukturerad data som är läsbart för människor.
Komponent	Byggsten i en Androidapplikation.
MVC	*Model, View, Controller*. Ett designmönster som används inom applikationsutveckling.
Rails	Ramverk för webbutveckling genom programmeringsspråket Ruby.
RFID	*Radio Field Identification*. Används för att skicka data över radiovågor korta sträckor.
Ruby	Objektorienterat programmeringsspråk.
Webbservice	En webbaserad applikation för direkt interaktion med andra applikationer.

Figurförteckning

1 Inledning

1.1 Bakgrund

Personbilen har sedan många årtionden varit en självklar del av det vardagliga livet. Den ger oss möjligheten att förflytta oss långa sträckor på relativt kort tid och underlättar vid flytt av otympliga eller tunga föremål. Personbilarna är så viktiga i vår vardag att det i dagsläget går i snitt 461 bilar per 1000 invånare i Sverige, vilket med våra 2,1 invånare per hushåll ger ungefär en bil per hushåll (Statistiska Centralbyrån, 2011). Många bilresor görs av bekvämlighet och 45 % av alla resor är kortare än 5 km, ändå kördes 232,3 miljoner kilometer på en genomsnittlig dag i Sverige 2006 (Abramowski L. och Holmström A., 2007). För att sätta det i perspektiv är jordens omkrets 40 075 km vid ekvatorn, följaktligen körde det svenska folket över 5700 varv runt jorden per dag.

Antalet personbilar tillsammans med de utsläpp de avger resulterar i både miljö- och hälsoproblem. Majoriteten av dessa bilar drivs av en förbränningsmotor som antänder fossila bränslen i en kontrollerad miljö. Vid förbränning av fossila bränslen avges bland annat diverse cancerframkallande partiklar och stora mängder koldioxid, som bidrar till ökad växthuseffekt. Vidare utnyttjas få bilar 100 % av tiden och står således ofta parkerade. Eftersom det finns ett så pass stort antal bilar bidrar det till mycket stora asfalterade parkeringsytor. Detta är inte problemfritt, då exempelvis avrinning för med sig föreningar till lokala vattendrag. Flera problem har visats av Chester M. et al (2010) och Davis A. Y. et al (2010).

Det finns flera alternativ till ägande av personbil vilka kan reducera transportrelaterad miljöpåverkan. Det är mycket vanligt att i stora städer cykla, resa kollektivt eller gå till fots i stället för att ta bilen. Ibland uppstår exempelvis behovet att nå en avlägsen plats utanför kollektivtrafikens räckvidd, vid dessa tillfällen kan det kännas motiverat att äga en personbil. Bilpooler ämnar minska det privata ägandet av personbilar genom att låta en samling personer dela på en eller flera bilar. De kan på så vis öka den enskilda bilens användningsgrad, vilket leder till färre bilar totalt. Ett lägre antal bilar leder till ett minskat behov av parkeringsplatser, därmed kan även antalet befintliga parkeringsplatser minskas. Genom att dessutom använda elbilar i bilpoolen blir det möjligt för allmänheten att sluta köra bilar drivna på fossila bränslen, utan att de för den delen ska behöva lägga stora summor på att köpa en elbil eller tänka mycket på huruvida de har tillgång till en laddstation eller inte.

1.2 Problemformulering

En tjänst för delat användande av elbilar kan göras attraktiv genom att låta användare ställa bilen på en godtycklig parkeringsplats, med en så kallad fritt flytande elbilspool. Tjänsten för en fritt flytande elbilspool kommer att behöva ett IT-system. De primära utmaningarna för ett sådant IT-system är hur det kan göras enkelt, bekvämt och attraktivt för en användare att använda det, samt hur mjukvaran ska utvecklas för att det ska vara så enkelt som möjligt att vidareutveckla den.

1.3 Syfte

Rapportens syfte är att beskriva och presentera projektet Elbipo, hur utvecklingen av systemet gått till och även hur den slutgiltiga produkten ser ut. Med problemformuleringen i avsnitt 1.2 som grund, syftar utvecklingsprocessen till att utreda hur ett IT-system kan implementeras för en fritt flytande elbilspool.

1.4 Avgränsningar

Detta projekt avgränsas till att enbart ta fram en webbapplikation, en Androidapplikation, dokumentation och ett API. Mjukvaran kommer att utvecklas till alfastadiet med kärnfunktionalitet.

2 Metod

Detta kapitel beskriver kortfattat den metod som använts vid genomförande av projektet.

Analys Projektet påbörjades med att fastställa hur ett system för en elbilspool kan fungera. Analyser av andra bildelningstjänster gjordes för att inhämta inspiration. Utifrån egna tankar tillsammans med dessa analyser utformades ett nytt system, som fick namnet Elbipo. Med hjälp av användningsfall definierades systemet ytterligare. För realisering av detta system undersöktes ett antal programmeringsspråk, både för hemsida och för Androidapplikation, som kunde tänkas ge gruppen de tekniska verktyg som krävs vid utveckling.

Utveckling Efter analys påbörjades en fas då gruppen utbildade sig inom det valda utvecklingsspråket med hjälp av diverse källor, både online och i böcker. Denna fas övergick naturligt till utvecklande i takt med att gruppen blev alltmer bekväm med språket och miljön. Parallellt med denna utbildning och utveckling specificerades andra detaljer för systemet, som hur säkerheten och kommunikationen mellan hårdvaruenheter skulle gå till rent tekniskt. De användningsfall som tagits fram prioriterades och implementerades sedan i prioritetsordning. Vid utvecklandet nyttjades användningsfallen för att vägleda gruppens implementering av funktionalitet. Webbapplikationen utvecklades först och därefter utvecklades Androidapplikationen.

Utvärdering För att utvärdera systemets värde och kvalité granskades den färdiga produkten kritiskt av gruppen utifrån de egenskaper som finns beskrivna i ISO 9126-1:2001, en modell för mjukvarukvalité.

3 Teknisk bakgrund

I detta kapitel presenteras den tekniska bakgrund som denna rapport bygger på.

3.1 Relaterade system

Det finns i dag flera system som syftar till att hyra ut bilar vilka drivs av företag specialiserade på biluthyrning, men även av andra aktörer med uthyrning som sidoverksamhet. Vad de ofta har gemensamt är att de bara har en eller ett väldigt begränsat antal platser för att hämta eller lämna en hyrd bil. För att göra bilarna mer lättillgängliga finns det i dag en del aktörer med distribuerad flotta, vilket innebär att de har ett större antal platser för hämtning och lämning av hyrd bil. De flesta av dem kräver dock att en bokning av en bil har gjorts innan bilen är tillgänglig för användning. Exempel på sådana system är Sunfleet och Zipcar (Sunfleet, 2012) (Zipcar, 2012).

Car2go är ett företag som levererar en biluthyrningstjänst där kunden kan hämta och lämna bilen på en godtycklig och laglig parkeringsplats i närheten av sin destination, en så kallad fritt flytande bilpool (Car2go, 2012). Det kräver heller inte att en bokning har gjorts innan användning av bil, det enda som krävs är att du är medlem i Car2go. Eftersom systemet tillåter sina kunder att lämna bilar på valfri laglig parkering, och således kan stå i stort sett var som helst, är det viktigt för systemet att mottaga och lagra information från bilar om deras positioner.

En viktig del i de system som använder en distribuerad flotta är hur användaren identifierar sig mot bilen och får tillgång till den. Flera av de system som granskats använder sig av RFID-taggar för detta. Genom att exempelvis låta varje användare ha ett eget kundkort med en RFID-tagg kan identifiering ske med hjälp av en läsare som sitter lättåtkomligt från utsidan av bilen.

3.2 Ruby on Rails

Ruby on Rails, hädanefter kallat Rails, är ett webbramverk till det objektorienterade programmeringsspråket Ruby (Ruby on Rails Guides, 2012a). Ett webbramverk är ett verktyg som ger stöd för utveckling av webbapplikationer genom exempelvis inbyggda funktioner och en grundläggande klasstruktur som ska se till att webbapplikationen kan utvecklas och fungera på ett bra sätt. Rails kan användas för att utveckla en fullt fungerande webbapplikation på förhållandevis kort tid tack vare två principer som Rails följer, *Convention over Configuration* (CoC) och *Don't Repeat Yourself* (DRY) (Ruby on Rails

Guides, 2012b). CoC innebär att utvecklare endast ska behöva skriva och definiera det som avviker ifrån det konventionella standardbeteendet. DRY säger att all definiering av data eller logik enbart ska finnas på en plats i systemet. Det innebär att det inte ska finnas redundant kod alls. Det innebär också att koden blir lättare att förstå och underhålla. Dessa båda principer är enkla att följa som utvecklare och avvikelser ifrån konventionen kan ofta leda till mer arbete än nytta.

Strukturen i en Railsapplikation är uppbyggd enligt designmönstret MVC. Syftet med MVC är att separera den grafiska presentationen från datan och logiken. MVC står för Model-View-Controller vilket hädanefter kommer att benämnas vid den svenska översättningen Modell-Vy-Kontroller. Detta gör utvecklingsprocessen enklare då det finns en mycket logisk ansvarsuppdelning inom applikationen.

I Rails byggs en webbapplikation upp enligt MVC-mönstret på följande vis (Ruby on Rails Guides, 2012a):

Kontroller. När ett anrop skett till en Railsapplikation skickas det först till en kontroller. Där sker manipulering av modeller och sedan skickas relevant data vidare till korrekt vy. Kontroller ärver ifrån Railsklassen `ActionController` och får på så vis tillgång till metoder som hjälper den utföra saker så som omdirigering och rendering av vyer. Kontrollern ansvarar även för logiken i systemet vilket innebär att den översätter interaktion med systemet till data i modellen. Kontrollern implementeras ofta så att varje kontroll har ett ansvar för vissa specifika vyer eller för manipulering av en speciell modell.

Modell. De klasser som ärver ifrån Railsklassen `ActiveRecord` kallas för Railsapplikationens modeller. Modellerna innehåller information om vilka attribut de har och vilka värden dessa attribut får ha, för att skapa en logisk entitet i applikationen. Typiska modeller är `User` för ett system med användare eller `Post` för ett forum. Dessa har en tät koppling till databasen och `ActiveRecord` tillgodogör många funktioner vilka hanterar interaktion med databasen, så som `save`. `ActiveRecord` lägger även till viss information till varje modellinstans, så som `created_at` och `updated_at`.

Vy. Vyn presenterar data för användare när begäran om visning gjorts. Vyerna använder sig av Railsklassen `ActionView` och är skriptfiler av typen *embedded ruby*. Dessa filer innehåller oftast HTML blandat med Rubykod, för att presentera dynamisk data. Vyer presenterar data som hämtats i kontrollern.

3.3 Android

Androidapplikationer byggs upp av ett antal komponenter. De typer av komponenter som finns är *activities*, *services*, *content providers* och *broadcast receivers*. Activities är det gränssnitt som användaren ser. Services är något som körs i bakgrunden, en dold process utan grafiskt gränssnitt. Content providers gör det möjligt att dela data mellan applikationer. Broadcast receivers är komponenter som lyssnar på systemanrop, ett exempel är att systemet skickar ut ett anrop om låg batterinivå. För alla komponenter finns Javaklasser som tar hand om logiken, det kan vara allt ifrån att hantera vad som ska hända när en användare trycker ner en knapp i en activity till vad som ska ske när skärmen automatiskt släcks ner.

För att få hög säkerhet är Android är uppbyggt med principen *principle of least privilege*, som kortfattat innebär att en applikation inte ska ha mer rättigheter än den behöver (Android Developers, 2012b). För att skydda användaren går det i grundutförandet inte att få tillgång till funktioner som skulle kunna missbrukas, vilket gäller för de flesta hårdvarudelarna i enheten. Behövs åtkomst till låsta delar av systemet måste det specificeras och efterfrågas. Genom att ha på det här sättet kan användare se vilka rättigheter en applikation vill komma åt och själv bedöma om de tycker det verkar rimligt eller ej.

För varje applikation finns även en speciell fil, `AndroidManifest.xml`, där hela applikationen specificeras, exempelvis Androidversion, vilka komponenter som används, extra rättigheter applikationen behöver och externa bibliotek (Android Developers, 2012c). I slutändan kompileras applikationen, tillsammans med bilder och andra resurser, till en apk-fil som kan installeras på en Androidenhet.

3.4 ISO 9126

För att värdera kvalité hos mjukvara finns en standard i fyra delar utgiven av ISO. Standarden heter *Software engineering - Product quality* och har dokumentnummer 9126 (Swedish Standards Institute, 2001) (SQA Software Quality Assurance, 2012). Första versionen kom 1991 men en reviderades 2001 och nu är det den reviderade upplagan den som används. Standardens huvudsyfte är att minimera mänsklig bias vid utvärdering av mjukvara som kan negativt påverka resultatet och benämns vanligtvis med ISO 9126:2001. För att referera en specifik del sätts dess delnummer efter dokumentnumret, exempelvis ger del ett ISO 9126-1:2001.

Del två till fyra behandlar kvantitativa mått på en mjukvaras kvalité ur olika infallsvinklar och tanken är att goda resultat enligt del två ger goda

resultat i del tre, som i sin tur ger goda resultat i del fyra. De behandlar en mjukvaras kvalité i källkod, kvalité vid exekvering respektive hur mjukvaran fungerar i produktionsmiljö.

Del ett, vid namn *Quality Model*, specificerar en samling önskvärda egenskaper hos en mjukvara, indelade i sex grupper med undernivåer. Egenskaperna har i specificeringen definierats sådant att ett visst spelrum finns för att passa olika projekt, därför är det vanligt att undernivåerna delas upp ytterligare av företag och organisationer som gör mjukvaruutvärderingar. Nedan följer de sex grupperna, tillhörande undernivåer och beskrivande text.

Functionality - *Suitability, Accurateness, Interoperability, Compliance och Security*
För att uppskatta funktionalitet mäter mjukvarutestare om mjukvaran har de funktioner som angivits och om given indata ger förväntad utdata. Här behandlas också hur mjukvaran samarbetar med annan mjukvara, hur den följer lagar och riktlinjer och dess säkerhet mot otillåtna intrång.

Reliability - *Maturity, Fault tolerance och Recoverability*
Behandlar hur pålitlig en mjukvara är genom att fastslå att mjukvaran inte upplever några, eller ytterst få fel samt hur systemet tål och återhämtar sig från de fel som kan ske. När tålighet och återhämtning mäts tas även utomstående faktorer som hårdvarufel eller strömavbrott med i beräkningarna. Mjukvarutestare mäter också möjligheterna att starta upp ett system efter ett kritiskt fel och hur mycket data som förlorats.

Usability - *Understandability, Learnability och Operability*
För att mjukvara ska få bra betyg i användbarhet undersöks hur lätt det är att förstå systemets funktionalitet. Mjukvarutestare uppskattar även hur mycket inlärning som krävs för olika användarnivåer och hur lätt det är att använda systemet.

Efficiency - *Time behavior och Resource behavior*
Behandlar hur effektivt mjukvaran utför de uppgifter den klarar av. Mjukvarutestare undersöker hur lång tid tar det för systemet att utföra en operation och hur mycket processorkraft och minne som krävs för att utföra en operation.

Maintainability - *Analyzability, Changeability, Stability och Testability*
Behandlar hur komplicerat det är att göra ändringar i systemet, hur känsligt det övriga systemet är mot förändringen och hur lätt det är att testa en ändring. Mjukvarutestare undersöker även möjligheten att finna var ett fel härstammar ifrån och hur mycket ansträngning som krävs för att finna källan.

Portability - *Adaptability, Installability, Conformance och Replaceability*
För att bedöma portabilitet undersöks hur mycket konfiguration som krävs vid installation och hur många externa beroenden systemet har. Mjukvarutestare tittar också på hur system hanterar ändringar i dess operativa miljö och hur det hanterar utbyten av mjukvarukomponenter. Hur mjukvaran följer lagar och riktlinjer undersöks också.

4 Tillämpning

Detta kapitel beskriver detaljerat tillämpningen och utvecklingen av systemet. Först beskrivs systemet och därefter utvecklingen av webbapplikationen respektive Androidapplikationen. Sist i kapitlet presenteras kvalitetsutvärderingen.

4.1 Systembeskrivning

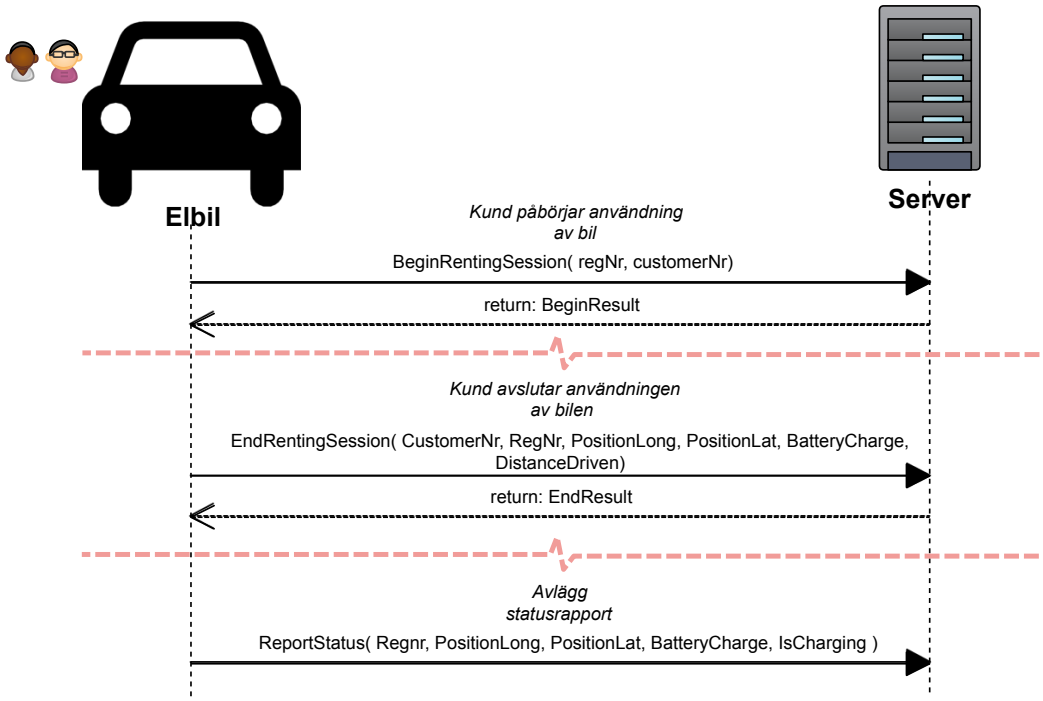
Huvudsyftet för det system som utvecklas är att låta kunder gemensamt nyttja en samling bilar genom att boka en bil och ha möjlighet att låsa upp ej upptagna bilar. En bokning kan göras i samband med användning av bil, det vill säga att det inte behövs en bokning sedan innan, förutsatt att bilen är ledig. Det är också möjligt att skapa en bokning i förtid för att vara säker på att bilen verkligen kan användas av den aktuella kunden. Om en bokning önskas göras i förtid görs den från en bokningsplattform, exempelvis en hemsida och en mobilapplikation. Har bokningen gjorts i förtid har den en fast tidpunkt då ej använd bokning tas bort och bilen görs åter ledig.

Anledningen till att en reservering tas bort är att en förtidsbokning betyder att en bil står obrukad och den sortens ineffektivitet är något projektet ämnar minska, det ger också en minskad vinst. När en kund i systemet har lokaliserat en bil och vill påbörja användning av den, identifierar sig kunden mot bilen med exempelvis RFID-teknik. Är bilen inte reserverad av någon annan och identifikationen accepteras låses bilen upp, oavsett om kunden har reserverat aktuell bil, en annan bil eller ingen bil alls. Om en annan bil är förtidsbokad av kunden kommer förtidsbokningen släppas även om tidsgränsen inte gått ut. När bilens dörrar väl är upplåsta kommer kunden på något vis att få tillgång till bilens nyckel, genom att exempelvis ta den ur ett låsbart fack inuti bilen.

Alla bilar i systemet har egen hård- och mjukvara som kommunicerar med en central server, vilket presenteras i figur 1. När kunden identifierar sig med ett kundkort[a] skickas data till servern, som skickar svar om kunden har rättighet att använda bilen. Finns startnyckeln inte i det låsbara facket kommer mjukvaran i bilen informera kunden om att den är bokad. Det leder också till att kunden har möjlighet att hålla sin bokning aktiv även om bilen parkeras och låses genom att ta med sig startnyckeln.

Data skickas även från bilen till servern på ett förutbestämt intervall för att lämna statusrapport. Statusrapporter innehåller information om var bilen befinner sig och dess batteristatus, intervallet är satt sådant att batteristatusen hålls uppdaterad. Efter användning av bilen skickas en slutlig statusrapport som avslutar bokningen, data som skickas innehåller bland annat hur

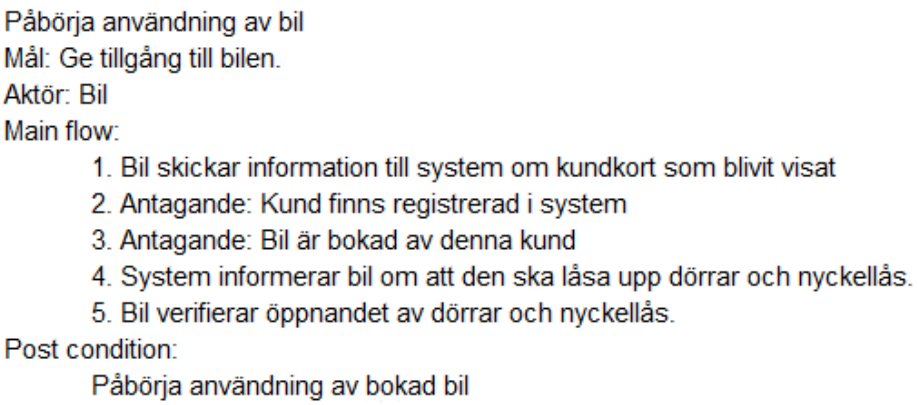
lång tid bokningen varit aktiv, bilens position och batteristatus. Fakturering kan sedan ske baserat på total tid eller körsträcka.



Figur 1: Illustration som visar hur kommunikation mellan elbil och server är tänkt att gå till.

När kunden önskar göra en förtidsbokning från hemsidan eller Androidapplikationen ges möjlighet att söka lediga bilar utifrån en adress eller den GPS-koordinat kunden befinner sig på. När en sökning gjorts visas de närmsta lediga bilarna, sorterat på avstånd från kundens sökning. Hemsida och mobilapplikation ger också kunden möjlighet att söka laddstationer då kunden kan tilldelas en premie om bilen parkeras vid en laddstation efter användning. Detta för att minska behovet för anställda att flytta bilar med låg batterinivå.

Användningsfall Baserat på systembeskrivningen har användningsfall tagits fram då de ger en klarare och mer välstrukturerad bild av hur systemet fungerar. Användningsfall beskriver interaktionen mellan en roll och systemet där en roll kan vara en människa eller ett externt system. I figur 2 presenteras ett exempel på ett användningsfall som behandlar en kund som vill låsa upp en bil. Fall när antagande inte håller, samt fler användningsfall finns att läsa i appendix A.



Figur 2: Exempel på användningsfall.

4.2 Tekniska val

Detta avsnitt presenterar de tankar som låg bakom valen av utvecklingsmiljöer och versioner.

4.2.1 Språk och ramverk

Det språk som valdes för utvecklingen av webbapplikation var Ruby tillsammans med ramverket Rails, vilket kan läsas mer om i avsnitt 3.2. Valet av språk baserades på gruppmedlemmarnas individuella övervägningar av för- och nackdelar hos olika språk, vilket sedan diskuterades för att nå fram till ett beslut som passade projektet och gruppen. De alternativ till Rails som övervägdes var bland annat ASP, ASP.net, PHP och node.js men samtliga avslogs. Anledningen till att Rails valdes framför de andra alternativen var till stor del det höga intresse för att lära sig just Rails, samt att den allmänna uppfattningen var att Rails skulle uppfylla de krav som fanns på programmeringsspråket.

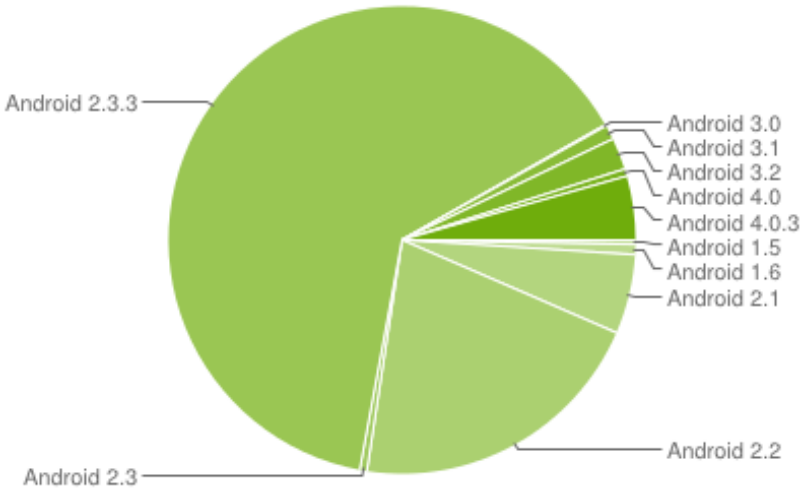
Olika språk undersöktes även för Androidapplikationen. Vedertagen standard för Androidutveckling är Java, dock fanns en önskan inom gruppen att inte låsa sig till Java utan att först överväga vilka alternativ som fanns tillgängliga för utvecklingen. Ruboto, en programvara som gör det möjligt att köra Ruby i Androidmiljöer, sågs som en möjlig kandidat till utvecklandet. Anledningen var att webbapplikationen skulle skrivas i Ruby och det kan vara önskvärt att utveckla all kod i samma språk. Ruboto jämfördes med Java och det blev tydligt att Ruboto i kontrast till Java inte är en tillräckligt utvecklad och stabil miljö för ett projekt med fastslagen deadline. För

Java och dess utvecklingsmiljö i Android finns det däremot väldigt mycket dokumentation, guider och hög grad av mogenhet. Således bestämdes det att Androidapplikationen kommer att utvecklas i Java.

4.2.2 Androidversion

Uppdaterade versioner av Android har sedan första release släppts minst en gång per år. Då mobiltillverkarna anpassar Android för att fungera på sina telefonmodeller tar det ofta ett tag innan nya versioner går att använda i praktiken. Det är främst nyare telefoner som får tillgång till de nya versionerna av Android, äldre uppdateras inte längre alls eller får bara kritiska uppdateringar. En applikation som är kompilerad för en äldre version kommer generellt sett att fungera i nyare versioner av systemet (Android Developers, 2012d). Med den vetskapen ville gruppen göra applikationen kompatibel med en så gammal version som möjligt, för att majoriteten av Androidtelefoner ska kunna köra applikationen.

I figur 3 kan utläsas att version 2.1, kodnamn Eclair, eller högre är installerad på 99 % av alla telefoner som används. Att ha den som mål för utvecklingen föll sig då naturligt och efter att ha undersökt nyare versioner (2.2, 2.3.3, 2.3.4 och 4.0 [ej 3.x, då dessa endast används för surfplatta]) kom gruppen fram till att den nyare funktionalitet som tillkommit troligen inte kommer behövas för den applikation som ska utvecklas. Undersökningen resulterade således i att Androidapplikationen ska utvecklas och kompileras för version 2.1 eftersom den passade bäst efter de kriterier som fanns.



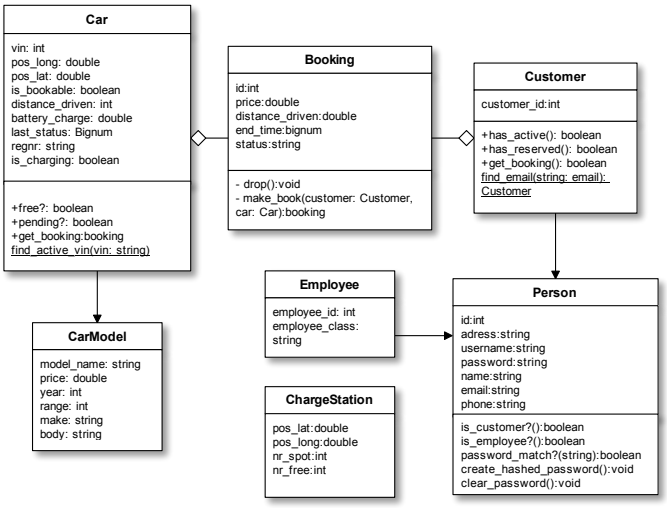
Figur 3: Fördelning över Androidversioner på installerade enheter (Android Developers, 2012a).

4.3 Webbapplikation

Detta avsnitt presenterar utvecklandet av webbapplikationen och även tankarna bakom vissa val som gjordes.

4.3.1 Systemmodellering

För att realisera det teoretiska system som beskrevs i avsnitt 4.1 valdes att översätta systemet till en klasstruktur. Denna klasstruktur ger en logisk uppdelning av den data och logik som finns och gör det enkelt och överskådligt att arbeta med applikationen. Som nämnts i avsnitt 3.2 kallas vissa klasser inom en Railsapplikation för modeller. I figur 4 presenteras de modeller som används i applikationen.



Figur 4: UML-diagram över de modellklasser som finns i systemet.

`Booking` används för att koppla samman en bil och en kund till just en bokning. Instanser av klassen skapas då en kund väljer att boka en bil eller då användande av ledig bil påbörjas. Bokningsinstansen har som attribut en sträng som heter `status`, som reflekterar nuvarande status på bokningen. För ett normalt flöde av en bokningsprocess kommer denna anta värdena `pending`, `active` och `expired`, i den ordningen. Om en bokad bil ej blir hämtad

inom angiven tid kommer bokningen att gå direkt ifrån `pending` till `expired`. Varje bokning har en mängd attribut, exempelvis `distance_driven`, `created_at` och `end_time`, vilka beskriver en boknings körsträcka, starttid och sluttid.

`Person` representerar en registrerad användare i systemet. Klassen innehåller det som behövs för att kunna logga in på systemet, så som användarnamn och lösenord och även generella personuppgifter för att kunna kontakta en person. Instanser av `Person` kommer att ha en koppling till en instans av antingen `Customer` eller `Employee`, beroende på vilken sorts användare det är.

Instanser av modellen `Car` representerar en fysisk bil i systemet. Dessa identifieras med sitt *Vehicle Identification Number*, `vin`, ett unikt nummer som varje bil får vid tillverkning. För varje instans lagras ett antal attribut som syftar till att kunna ge en översikt av hur bilens nuvarande tillstånd är.

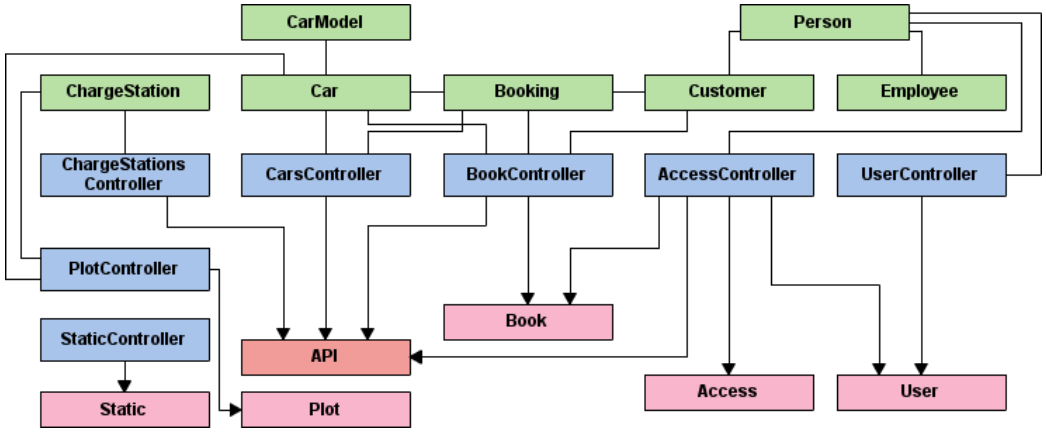
Databas Railsapplikationens modeller önskas finnas i ett persistent tillstånd. För detta har valet gjorts att inkrementellt bygga en databas med hjälp av vad Rails kallar för *migrations*. En migration är en fil som innehåller information om hur databasen förändras mellan versioner av applikationen för att enkelt kunna byta mellan olika databasversioner. Elbipos databas har implementerats helt och hållet med hjälp av migrations och således finns det spår av migrations som reflekterar de förändringar i databasen som skett under utvecklingens gång. Dessa migrations har använts för att bygga upp en MySQL-databas som Railsapplikationen är ansluten till.

`ActiveRecord` lägger även till viss information till varje modellinstans, vilket nyttjas av webbapplikationen. `created_at` är ett attribut som varje instans av klasserna i figur 4 tilldelas när objektet skapas. Detta automatiskt genererade attribut används exempelvis av `Booking` för att avgöra när en bokning gjordes.

4.3.2 Klasstruktur

Det här avsnittet ämnar förklara implementationen utav webbapplikationen på en djupare teknisk nivå. Figur 5 visar hur en grov klassarkitektur för applikationen ser ut, med markeringar för de olika delarna enligt MVC.

`AccessController` hanterar autentisering av kunder och begränsar tillgången till vyer utifrån de restriktioner som finns för ej inloggade användare. För att verifiera e-postadress och lösenord används modellen `Person` för att hämta lagrad data och matcha mot indatan. Även `UserController` använder sig av `Person`, för att hämta information om kunden som sedan kan presenteras i vyn `User`.



Figur 5: Grovarkitekturen hos webbapplikationen. Gröna rutor markerar modeller, blåa kontroller och rosa vyer.

`PlotController` är enbart till för hemsidevyerna och har som uppgift att ta emot en adress eller koordinat för att sedan leta upp de lediga bilar eller laddstationer som befinner sig närmast den angivna positionen. Den använder sig utav information tillhandahållen av modellerna `ChargeStation` och `Car` för att slutligen presentera resultaten på en karta i vyn `Plot`. När kunden valt bil är det `BookController` som tar hand om logiken för att göra bokningen. Den kollar med hjälp av modellerna `Car`, `Booking` och `Customer` så att kunden inte har någon bokad bil och så att bilen verkligen är ledig. Resultatet av bokningen presenteras sedan i vyn `Book`. Det är även den vyn som hanterar visning av tidigare bokningar med hjälp av `BookController`. Det är samma logik i `BookController` som används för att möjliggöra bokningar via API:et.

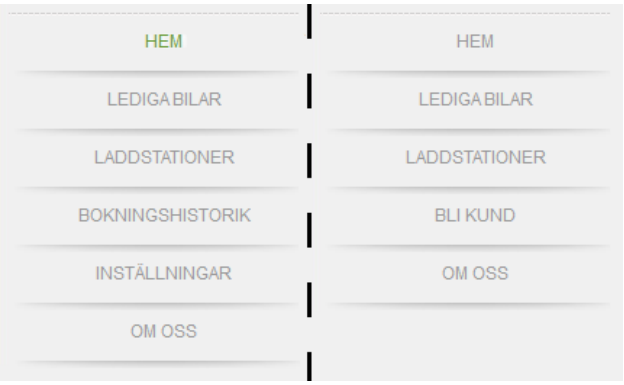
`CarsController` och `ChargeStationsController` har som uppgift att förse API:et med logik för att söka laddstationer och bilar utifrån adress eller koordinat. Resultatet från sökning på laddstationer ger de laddstationer som ligger närmast, samt antalet lediga platser. När bilar sökes ska information om bland annat registreringsnummer, modell, tillverkare, batteristatus och position returneras. `CarsController` är även den kontroller som det privata API:et använder sig av. Där tar den emot och uppdaterar bilars status samt start och avslut av en bokningssession. Vid en statusuppdatering använder den sig av modellen `Car` för att uppdatera databasen. När en ny session startas använder den sig av modellen `Booking` för att verifiera att en bokning finns. Om det inte finns någon bokning och bilen är ledig kommer en ny bokning att göras. Vid sessionsavslut används samma modell för att ändra bokningens status till avslutad och bilen blir därefter ledig.

`StaticController` används endast för vyerna som hör till hemsidan, dess uppgift är bara att hantera förfrågningar till de statiska sidor som finns.

4.3.3 Hemsida

Hemsidan är den del av webbapplikationen som kunden kommer att se och använda. Eftersom kunder som kommer använda systemet har väldigt varierande IT-kunskaper krävs att det ska vara väldigt lättanvänt och attraktivt för dem alla. För att lösa detta har stor vikt lagts på att den ska vara lättnavigerad, lättanvänd och ha en stilren design. Den design som används kommer från Free Website Templates (2012) och valdes då den ger ett seriöst intryck och har en avskalad design.

För att göra sidan lättnavigerad arbetades två huvudmenyer fram, en för inloggade och en för ej inloggade användare, från vilka användare och kunder ska kunna nå de, för dem, vitala delarna av hemsidan.



Figur 6: De två huvudmenyer som visas för användare på hemsidan (inloggad kund till vänster).

Gemensamt för de båda menyerna är alternativen `Hem` - som leder till välkomstsidan, `Om oss` - sidan som kommer presentera systemet, `Laddstationer` - den sida som kommer att låta användare söka laddstationer och `Lediga bilar` - den sida som kommer att låta användare söka lediga bilar. Att låta `Lediga bilar` finnas för ej inloggade användare är för att demonstrera enkelheten i sökningsmomentet. Alternativet `Bli kund` kommer enbart finnas tillgängligt för ej inloggade användare, här ska information om hur de går till väga för att bli kunder finnas.

Inloggade kunder ska genom alternativet `Inställningar` enkelt kunna komma åt sina kontaktuppgifter och ändra dem. Alternativet `Lediga bilar` ska för inloggade kunder göra det möjligt att boka bilar via kartan. Ett sista alternativ som endast kommer att vara tillgängligt för inloggade kunder är `Bokningshistorik`, där kunden ska kunna se aktuella och tidigare bokningar, samt ta bort nyligen lagd bokning.

Sökning och bokning av bilar är de viktigaste och mest komplexa delarna av hemsidan för en kund. Därför låg störst fokus på att göra de funktionerna enkla. Med hjälp av Google Maps API kan en karta ritas upp över sökt område tillsammans med markörer för de närmsta lediga bilarna. Genom att hålla muspekaren över en bil kommer information visas om modell och tillverkare. När kunden valt vilken bil som ska bokas räcker det med ett klick på markören för att boka den. Samma kartsystem implementerades för laddstationer, med skillnaden att den i stället visar information om antalet platser hos laddstationen när muspekaren hålls över den.

4.3.4 API

För Elbipos centrala server övervägdes att implementera en specialiserad mjukvara för att sköta all databashantering. Tanken var då att låta bilar, laddstationer, Androidapplikationer och även webbsida skicka databasrelevant information genom den. När Rails sedan valdes som ramverk för hemsidan var den idén ej längre relevant. Då Rails har egen databashantering och ger möjlighet att skapa en *webbservice* som brukar samma logik och modeller som webbplatsen var det ett klart bättre val.

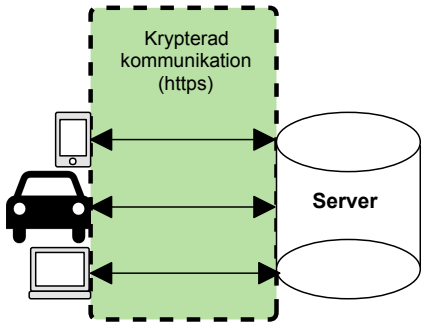
Då en webbservice valdes kommer alla förfrågningar ske genom HTTP-anrop och följa de restriktioner som *REST* specificerar. REST står för *Representational State Transfer* och är en arkitektur för distribuerade system och en webbservice som följer alla restriktioner kallas ofta RESTful. Mer läsning om REST finns i Roy Fieldings dissertation (Fielding R. T., 2000). API:ets funktioner kommer delas in i två grupper, öppna och privata, där öppna API är tillgängliga för allmänheten. Att vissa funktioner görs tillgängliga för allmänheten tillåter tredjepartsutvecklare bidra med egen mjukvara vilket potentiellt kan ge en bättre användarupplevelse. Det privata API:et används uteslutande för rapportering från diskreta enheter, till exempel bilar och laddstationer, till den centrala servern och det vore en potentiell säkerhetsrisk att lämna dessa öppna för allmänheten.

En RESTful webbservice skickar normalt sett data i antingen JSON-, XML- eller YAML-formatering. Av dessa tre alternativ kommer JSON användas eftersom både Rails och Android har inbyggt stöd för JSON-objekt. JSON är också skapat med läsbarhet i åtanke vilket främjar tredjepartsutveckling.

4.3.5 Säkerhet

Hantering av användardata En fundamental tanke inom datasäkerhet är att viss data skall vara skyddad även om en angripare kommer över den. Eftersom systemet Elbipo kommer att innehålla kritiska användaruppgifter, exempelvis lösenord, är det viktigt att det finns adekvat säkerhet vid lagring av dem. Att användare återanvänder lösenord på många olika platser är i dag väldigt vanligt och om de lösenord som används till Elbipo-systemet blir övertagna av en angripare skulle användarnas konton på andra platser också kunna bli det. En dålig lösning till det här problemet är att låta lösenorden lagras i klartext och vid intrång skulle således en angripare ha lösenorden tillgängliga. Därför undersöktes vad som är god standard för lagring av lösenord och Ulrich J. (2011) beskriver hur problem löses vid lagring av användardata för inloggning på ett system. I korta drag handlar det om en algoritm som skapar en representation av lösenordet, en hash, som gör att det inte enkelt eller snabbt går få ut lösenordet i klartext. Denna representation görs ytterligare tidskrävande att knäcka genom att använda en kort textsträng, ett salt, tillsammans med lösenordet innan det hashas.

Krypterad kommunikation De olika komponenterna i Elbipo kommer att kommunicera med varandra över Internet, vilket kan innebära en säkerhetsrisk på grund av dess öppna struktur. Eftersom systemet kommer att ha vetskap om olika bilars, och därmed användares, positioner i realtid är det viktigt att användarna kan garanteras säker dataöverföring. För att vara säker på att den mest säkerhetskritiska data som går mellan klienter och servern inte kan avlyssnas kommer Elbipo att använda sig av protokollet HTTPS. Detta säkra protokoll används för överföring av användaruppgifter, uppgifter om bokningar och överföring av bilarnas positioner, se figur 7 för illustration av detta.



Figur 7: Denna figur illustrerar var HTTPS används.

4.4 Androidapplikation

I det här avsnittet presenteras utvecklandet av den Androidapplikation som finns i systemet, hur den har byggts upp och hur tankarna bakom viss implementation gått till.

4.4.1 Gränssnitt

Syftet med applikationen är att kunder på ett enkelt sätt ska kunna söka och boka bilar samt söka laddstationer, var de än befinner sig. Precis som för hemsidan (avsnitt 4.3.3) ska applikationen vara attraktiv och kunna användas av alla kunder, oberoende av IT-kunskapsnivå. Därför gällde samma sak här; att stor vikt skulle läggas på att få applikationen lättnavigerad, lättanvänd och ha en stilren design.

Designen blev Androids standarddesign för grafiska komponenter, exempelvis knappar och textfält. Valet att använda standarddesignen grundades på att de flesta som äger en Androidtelefon redan är familjära vid det utseendet på applikationer, samt att funktionsimplementering prioriterades högre än designutformande.

Att få applikationen lättnavigerad löstes genom att, efter en kund loggat in, presentera en huvudmeny som kan navigera till de olika delarna kunden behöver komma åt. En skiss för hur menyn skulle se ut presenteras i figur 8.

Figur 8: Skiss på huvudmeny som kund möts av efter att ha loggat in.

Precis som med hemsidan var det sökning och bokning av bilar som var den stora utmaningen i applikationen när det kom till att göra den enkel att använda. Eftersom Google Maps API, som används för hemsidan, även

finns för Android föll sig valet att använda det självklart. För att ytterligare förenkla sökningen av bilar ska valet att söka via den inbyggda GPS:en finnas tillgängligt. En funktion för adress-sökning ska också finnas tillgänglig, som kan användas när GPS:en inte kan fastställa position eller när sökning önskas göra på annan position. Efter att positionen är fastställd ska kunden få upp en karta med resultat i stil med figur 9 där den blå punkten markerar sökt position och de gröna lediga bilar.

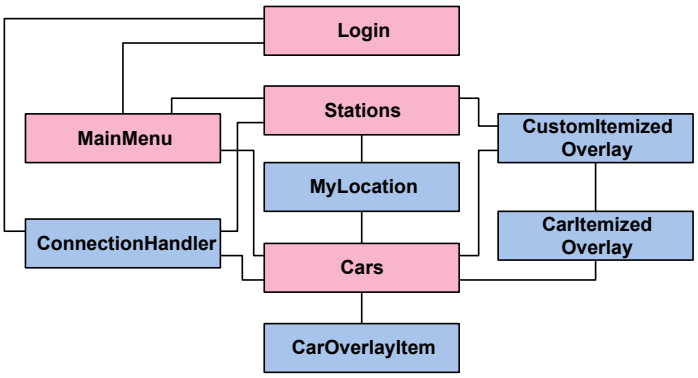
Figur 9: Bilsökning. Markörer som visar bilar och nuvarande position.

För enkelhetens skull gjordes tillvägagångssättet för att boka bilar så likt det på hemsidan som möjligt. Kunden ska genom att klicka på en ledig bil få upp information om märke, modell, årsmodell, batterinivå och registreringsnummer samt en knapp som kunden genom att klicka på ska kunna boka bilen, se figur 10. Att söka efter laddstationer kommer ske på liknande sätt som bilsökningen, med skillnaden att det är information om antalet lediga platser vid vald laddstation som syns när kunden trycker på den.

Figur 10: Information som visas efter att ha klickat på en bilmarkör på kartan.

4.4.2 Klasstruktur

Androidapplikationen kommer enbart att använda sig utav activitieskomponenter. De fyra activitieskomponenterna som kommer användas är `Login`, `MainMenu`, `Cars` och `Stations`. I klassdiagramet som presenteras i figur 11 är deras tillhörande klasser markerade rosa.



Figur 11: Klassarkitektur i applikationen.

`Login` kommer att vara den komponent användaren möts av först. Den hanterar autentisering för att få använda applikationen. När användaren väljer att logga in kommer data skickas till servern för autentisering. För att hålla koll på kundens session i telefonen kommer information om e-postadress, det hashade lösenordet och om användaren vill spara sina uppgifter sparas de i applikationens `SharedPreferences` vid lyckad inloggning. `SharedPreferences` är en klass som används när lättare applikationsdata persistent ska lagras, så som inställningar och användaruppgifter.

När kunden väl loggat in kommer en huvudmeny att presenteras med tre alternativ; `Sök lediga bilar`, `Sök laddstationer` och `Logga ut`. `Logga ut`-knappen kommer att nollställa den information som sparats i `SharedPreferences` och skicka användaren till `Login` igen. `Sök lediga bilar`-knappen kommer att ta användaren till `Cars`. Här kommer en karta, implementerad med hjälp av Google Maps API, och två alternativ i form av knappar att presenteras för användaren. Alternativen är `GPS` och `Adress` och är de val användaren kommer att ha för att söka bilar. Vid GPS-sökning kommer klassen `MyLocation` att, med hjälp av den inbyggda GPS:en försöka fastställa kundens position och hämta dess koordinater. Skulle i stället adress-sökning användas kommer den sökta adressens koordinater hämtas. När positionen fastställts görs ett anrop till servern för att hämta information om de närmsta bilarna som sedan, tillsammans med kundens position markeras ut på kar-

tan. Trycker kunden på en markör för en bil kommer en dialogruta att visas, innehållandes knappar för att boka bilen och stänga dialogrutan. Väljer kunden att boka vald bil kommer ett anrop till servern att göras för att försöka boka den. Resultatet av bokningen kommer vid lyckat försök presenteras i form av bokningsnumret och vid misslyckat försök visas ett felmeddelande.

För att söka laddstationer kommer samma teknik med GPS- och adressökning användas som i sökningen av bilar. Skillnaden är informationen som finns i dialogrutan och knapparna den har. Här efterfrågas i stället information om antalet lediga platser vid de laddstationer som finns närmast den sökta platsen och endast ett alternativ om att stänga rutan.

Varje gång applikationen behöver kommunicera med servern används `ConnectionHandler`. Den ansluter till webbapplikationens öppna API för att skicka och ta emot data i enlighet med de säkerhetskriterier som tagits upp i avsnitt 4.3.5. Valet att ha en egen klass för serverkommunikation var för att få applikationen mer modulär. Skulle API:et, adressen till servern eller dylikt ändras skulle det bara vara den här filen som behöver ändras för att få allt att fungera igen.

`CustomItemizedOverlay` och `OverlayItem` används för att markera ut punkter på kartan. `OverlayItem` innehåller punktens koordinater och den textsträng som önskas visas vid markering. `CustomItemizedOverlay` innehåller alla `OverlayItem` och hanterar vad som ska hända när någon av dem klickas på, samt håller koll på med vilken bild punkterna ska markeras på kartan.

Klasserna `CarItemizedOverlay` och `CarOverlayItem` utökar `CustomItemizedOverlay` respektive `OverlayItem` och används enbart av `Car`. De implementerades av den orsaken att extra information behövde skickas till den `Overlay` som hanterar vad som händer vid markering. Den extra information som behövde skickas med var den markerade bilens registreringsnummer, som behövs när bilen ska bokas.

4.4.3 Rättigheter

För att applikationen skulle kunna använda de tänka funktionerna behövdes utökade rättigheter. De extra systemfunktioner applikationen behöver är tillgång till GPS:en och Internet. GPS:en behövs för att kunna bestämma användarens position vid lokalisering av bilar. Internet behövs för att kunna skicka och ta emot data från servern och för kartorna som presenteras vid sökning av bilar och laddstationer.

4.5 Kvalitetsutvärdering

För att skapa en uppfattning av produktens kvalité genomfördes en kritisk utvärdering av mjukvaran. Utvärderingen grundas i ISO 9126-1:2001 med de egenskapsbeskrivningar som finns angivna i avsnitt 3.4 och syftar till att ge ett underlag för diskussion kring projektet. Det finns inom gruppen inte någon tidigare erfarenhet inom kvalitetsäkring av mjukvarusystem vilket gör att utvärderingen utförs informellt. Utvärderingen genomfördes göras genom att gruppmedlemmarna värderar hur väl mjukvaran följer de givna egenskaperna och på så vis ge ett mått på kvalité.

När mjukvara utvärderas blir resultatet lätt partiskt om utvecklare och utvärderare är samma personer, även om en fördefinerad metod finns. För att få ett så opartiskt resultat som möjligt kommer individuella värderingar diskuteras för att sedan slås samman och ge mjukvaran styrkor, svagheter och möjliga förbättringar för varje egenskap.

För några av kvalitetsegenskaperna finns vissa problem för en grupp mjukvarutvecklare att testa och kommer därför enbart att diskuteras eller i vissa fall lämnas helt. På grund av begränsad möjlighet att påverka uppsättningen hårdvara kommer alla egenskaper rörande det lämnas till enbart diskussion. Juridiska spörsmål lämnas då ingen större erfarenhet finns inom gruppen.

5 Resultat

I det här kapitlet presenteras systemet i sin helhet. Kapitlet inleds med en beskrivning av webbapplikationen, därefter presenteras Androidapplikationen och slutligen en kvalitetsutvärdering av mjukvaran.

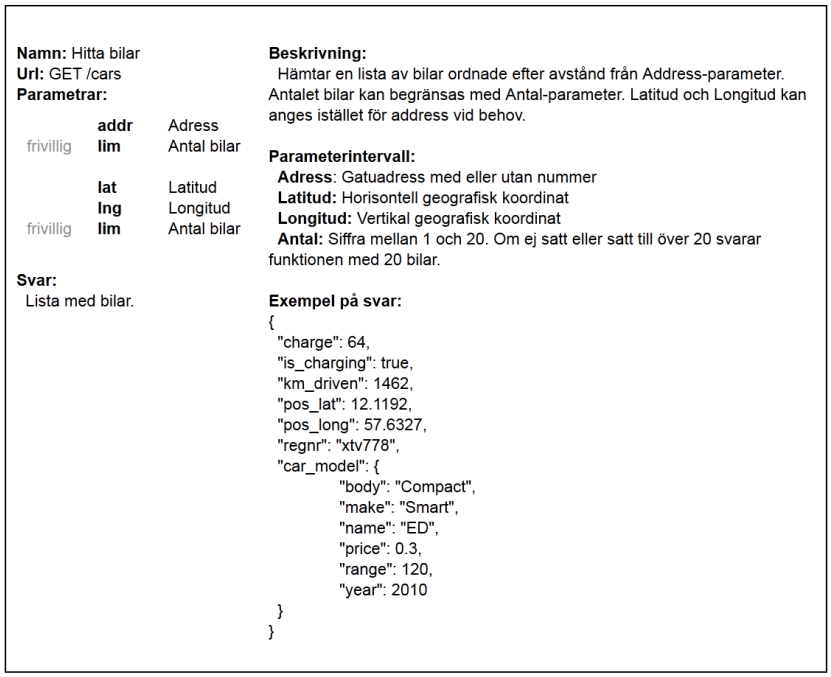
5.1 Webbapplikation

I det här avsnittet presenteras webbapplikationens två uppgifter, API och hemsida.

5.1.1 API

Elbiposystemet har skapats med både öppna och privata API:et för kommunikation av data. Med HTTP-anrop till servern går det att hämta data rörande flera delar av systemet, exempelvis var det finns lediga elbilar eller laddstolpar. Se appendix B för detaljer om API:et.
API:et är uppbyggt med en RESTful-struktur och data levereras i JSON-format. För de funktioner där datamängden kan bli ett problem finns det möjlighet att med parametrar begränsa antalet träffar. För känsliga tjänster, exempelvis bokningar och förändringar i användardata, föreslås att HTTPS används, dock är det inget krav.

Öppet API Vissa delar av Elbipos API är tillgängliga för allmänheten. De funktioner API:et uträttar är hämta information om bilar och laddstationer, samt att skapa nya bokningar. API:et ger möjlighet för tredjepartsutvecklare att utveckla egen mjukvara som interagerar med Elbipos system. För att det öppna API:et ska vara åtkomligt och enkelt att använda finns detaljerade beskrivningar av API:ets funktioner. De innehåller exempelanrop och vad som kan stå i svarsmeddelandet. I figur 12 visas formatet på denna beskrivning.



Namn: Hitta bilar
Url: GET /cars
Parametrar:

	addr	Adress
frivillig	**lim**	Antal bilar
	lat	Latitud
	lng	Longitud
frivillig	**lim**	Antal bilar

Svar:
Lista med bilar.

Beskrivning:
Hämtar en lista av bilar ordnade efter avstånd från Address-parameter. Antalet bilar kan begränsas med Antal-parameter. Latitud och Longitud kan anges istället för address vid behov.

Parameterintervall:
Adress: Gatuadress med eller utan nummer
Latitud: Horisontell geografisk koordinat
Longitud: Vertikal geografisk koordinat
Antal: Siffra mellan 1 och 20. Om ej satt eller satt till över 20 svarar funktionen med 20 bilar.

Exempel på svar:

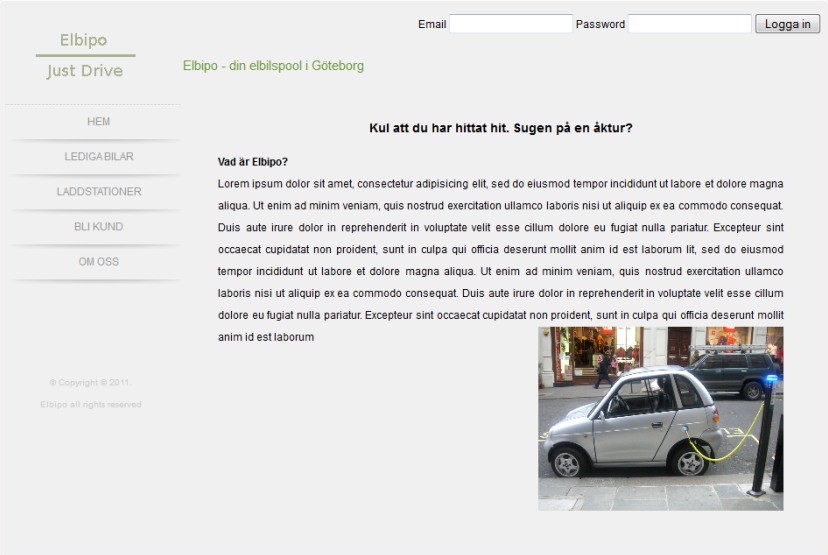
```
{
 "charge": 64,
 "is_charging": true,
 "km_driven": 1462,
 "pos_lat": 12.1192,
 "pos_long": 57.6327,
 "regnr": "xtv778",
 "car_model": {
          "body": "Compact",
          "make": "Smart",
          "name": "ED",
          "price": 0.3,
          "range": 120,
          "year": 2010
 }
}
```

Figur 12: Utdrag ur API-dokumentationen för hur information hämtas om de närmsta lediga elbilarna och samt hur svaret ser ut.

Privat API Det privata API:et, som inte är ämnat för allmänheten utan för rapportering inom systemet, är det gränssnitt som en modul i elbilarna ska kunna kontakta för att rapportera sin status. Även laddstationers rapportering skickas genom det privata API:et. Det har, liksom det öppna, också detaljerade beskrivningar för hur det skall användas, inklusive exempel på anrop och svar.

5.1.2 Hemsida

Elbipos webbplats har en minimalistisk design. I nuvarande alfastadie har ingen tid lagts på att skriva beskrivande texter, varför användare presenteras med fyllnadstext. Hemsidan presenterar olika menyer beroende på om användaren är inloggad eller ej. Till vänster på hemsidan i figur 13 syns menyn för ej inloggade användare.

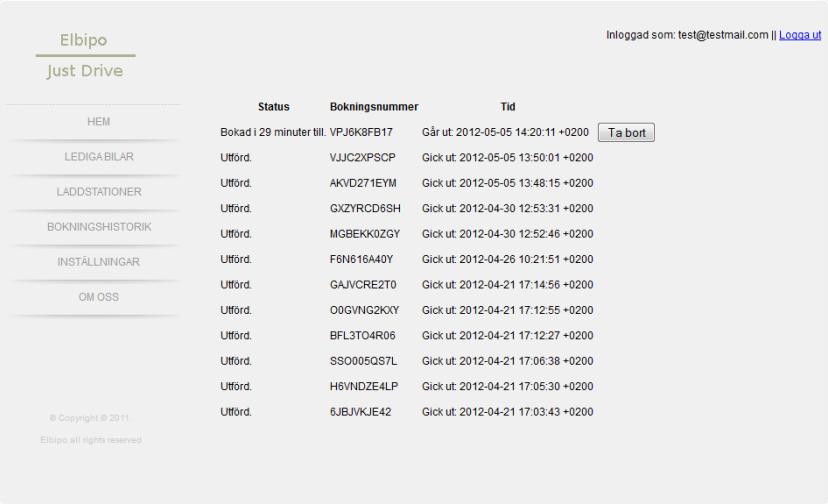


Figur 13: Förstasidan på hemsidan för ej inloggad användare.

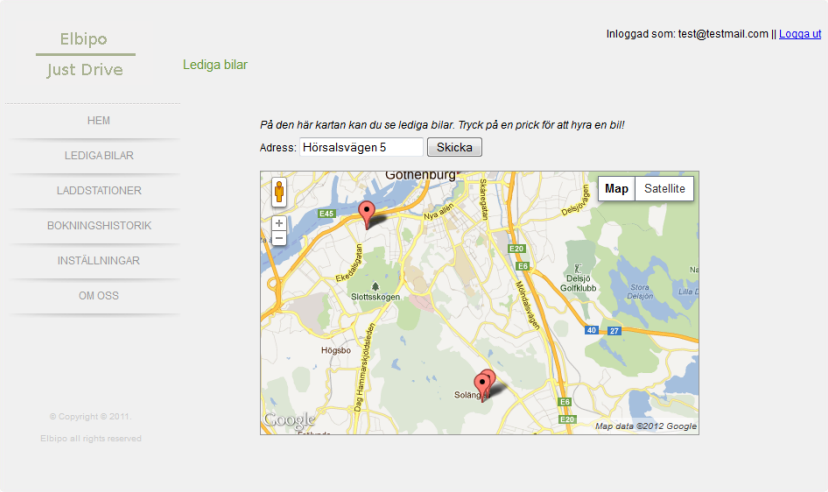
Då användaren väljer `Lediga bilar` presenteras en karta och ett sökfält för adress. Lediga bilar presenteras som markörer på kartan centrerade runt vald adress. Information om en bil presenteras när muspekaren hålls över bilens markör. Då användaren ej är inloggad går det endast att söka bilar, ej boka dem.

Samma vy ges för laddstationer, markörer ger nu lediga och totala antal parkeringsplatser för given laddstation. Alternativet `Bli kund` beskriver hur användaren går till väga för att registrera sig som kund i systemet. Längst upp på sidan finns två textfält där en kunds e-postadress adress och lösenord fylls i för att logga in.

Som inloggad kund ges tillgång till två nya menyval, `Bokningshistorik` och `Inställningar`. En bokning av bil går nu att genomföra och görs genom att en bilmarkör väljs med ett musklick. `Bokningshistorik` tar kunden till en sida där bokningsnummer och tid när bokning avslutades presenteras. Figur 14 visar bokningshistorik för en användare. Bokningssystemet för en inloggad användare visas i figur 15.



Figur 14: Bokningshistorik för en kund med en aktiv bokning.



Figur 15: Sökning av lediga bilar på hemsidan.

För en bokning som görs via hemsidan sätts en tidsgräns på hur länge bilen är reserverad. Reserveringen går att avbryta innan tidsgränsen passerat. `Inställningar` låter kunden ändra sina användaruppgifter. Hela menyn finns alltid tillgänglig för enkel navigering av webbplatsen.

5.2 Androidapplikation

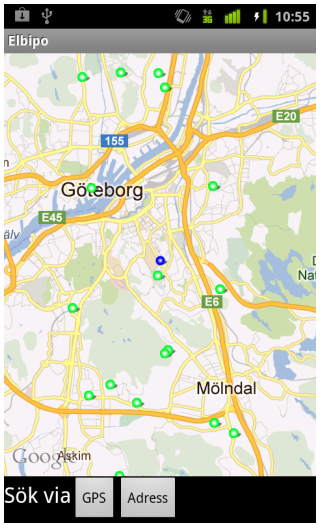
Vid start av applikationen kommer kunden att se en inloggningsruta där verifiering av e-postadress och lösenord krävs för att kunna använda applikationen, se figur 16. Inloggningen, precis som alla annan kommunikation som sker mot webbapplikationen, görs via det API som finns beskrivet i 5.1.1.

Figur 16: Androidapplikationens inloggningsskärm.

När kunden verifierats kommer välkomstskärmen att presenteras, innehållandes tre knappar för att navigera runt i applikationen, se figur 17. Alternativen som finns är `Sök lediga bilar`, `Sök laddstationer` och `Logga ut`.

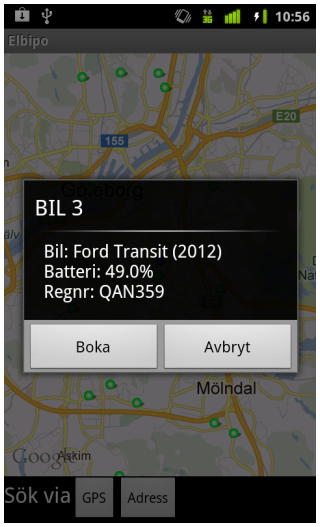
Figur 17: Androidapplikationens huvudmeny.

Väljer kunden att söka efter lediga bilar kommer en karta och två alternativ för hur sökningen ska göras att visas. Väljs alternativet `GPS` kommer telefonens inbyggda GPS att användas för att hämta information om vart kunden befinner sig, som sedan används för att hitta de närmsta lediga bilarna. Väljer kunden i stället alternativet `Adress` kommer en ruta upp där kunden själv kan fylla i adress vilken används som central punkt för att lokalisera de närmsta lediga bilarna runt. När sökningen väl gjorts kommer resultatet att se ut som i figur 18, där den blå punkten markerar den sökta positionen och de gröna lediga bilar. Om sökning via adress gjorts kan kunden genom att trycka på den blå punkten få upp en dialogruta som visar adressen. Har däremot sökning via GPS gjorts kommer dialogrutan att visa information om exaktheten i GPS:ens sökning.



Figur 18: Karta med utmarkerade bilar i Androidapplikationen.

Genom att trycka på någon av de gröna punkterna kommer användaren att presenteras information om bilen i en dialogruta, tillsammans med en knapp för att boka bilen och en knapp för att stänga dialogrutan, exempel kan ses i figur 19. Väljer användaren `Boka bil` kommer en ny ruta upp som visar bokningsnumret om bokningen lyckades, eller vad som blev fel om bokningen inte lyckades.



Figur 19: Dialogruta vid markerad bil.

Om kunden i huvudmenyn väljer alternativet `Sök laddstationer`, kommer en skärm liknande den för sökning av bilar att visas. När kunden gjort

en sökning här kommer ett resultat i stil med resultatet för bilsökningen att komma fram. Skillnaden här är att om kunden trycker på någon av de gröna markeringarna kommer en dialogruta med information om antalet lediga platser att visas.

5.3 Kvalitetsutvärdering

Här sammanställs kvalitetsutvärderingen av Elbiposystemet, gjord enligt tillvägagångssättet beskrivet i avnitt 4.5. Det fulla resultatet för utvärderingen finns att läsa i appendix C.

I utvärderingen ansågs att både hemsidan och Androidapplikationen har en tydligt och enkel design som är enkel att använda och navigera. Navigationen sker via den meny som alltid finns till vänster på hemsidan vilket, enligt mjukvarutestarna, ger en lättnavigerad miljö. Menyvalen har entydiga namn och designmönstret som brukas är väl beprövat, vilket minimerar inlärningstid. Uppfattningen kring bokningsprocessen är att den är snabb och enkel vilket följer den givna problemformulering. Det saknas dock konfirmationsrutor vid bokning av bil vilket leder till tvetydighet huruvida bil faktiskt blivit bokad, det går heller ej att smidigt ångra en bokningen. Vid val av bil ges också bristfällig information, bara tillverkare och karossmodell presenteras i text. Kostnadsinformation saknas helt.

Androidapplikationen följer samma designmönster och bokningsprocessen är identisk med den på hemsidan. Det räcker i allmänhet att användare lär sig hemsidan eftersom applikationen är en nerbantad version, ingen ytterligare inlärning krävs och har användaren börjat med att lära sig appen är det ytterst få nya koncept på hemsidan.

Att systemet är uppbyggt enligt DRY och MVC bidrar till att det blir lätt att underhålla och utveckla. Få eller simpla relationer mellan klasser ger en stark modularitet som gör det enkelt att byta ut hur delar av systemet hanteras utan att påverka övriga system. Bara vissa av modellerna i webbapplikationen har automatiska tester.

6 Diskussion

I det här kapitlet kommer produkten, metoden och vidareutveckling att diskuteras. Därefter presenteras en kort slutsats.

6.1 Produktkvalité

Syftet med den utvecklade produkten var att på ett enkelt, bekvämt och attraktivt sätt möjliggöra användning av ett system för en fritt flytande elbilspool, med vilket menas exempelvis bokning av elbilar. Utifrån resultatet av den kvalitetsutvärdering som presenterats i avsnitt 5.3 anses detta vara delvis uppnått, av ett par olika anledningar.

Mjukvaran är på många sätt mycket enkel och bekväm att använda, då en användare exempelvis genomför en bokning på hemsidan eller i Androidapplikationen. Däremot får användarna för lite information vid interaktion med systemet, om exempelvis priser, vilket gör att det är mindre attraktivt att använda det på grund av osäkerhet. Att systemet går att använda både på en hemsida och i en Androidapplikation gör det mycket bekvämt att använda och mycket attraktivare än om det bara hade funnits en hemsida.

I kvalitetsutvärderingen framkom att systemet är mycket enkelt att underhålla och utveckla vidare, då det finns hög modularitet i systemet. Detta är någonting som nästan kommer automatiskt då programvara utvecklas i Rails, på grund av flera principer som finns definierade (se avsnitt 3.2 för vidare förklaring). Dock finns det ej mycket teknisk dokumentation i form av exempelvis formella kravspecifikationer eller klassbeskrivningar. Det finns även mycket lite automatisk testning. För att ytterligare tillgodose syftet att systemet ska vara enkelt att vidareutveckla bör mer teknisk dokumentation finnas än vad som i nuläget finns att tillgå. Det borde också skapas mer automatisk testning.

6.2 Metodkvalité

Syftet med utvecklingsprocessen var att utreda hur ett IT-system enligt problemformuleringen kunde implementeras. Det har emellertid funnits brister hos produkten vilka kan härledas tillbaka till metoden.

Många konventionella utvecklingsprocesser använder sig av kravspecifikationer, vilket gör att detta projekts avsaknad av kravspecifikation känns som en viktig punkt att diskutera. I stället för en kravspecifikation utgick gruppen ifrån användningsfall vid implementering. Detta beslut togs då det i början av projektet var viktigt att snabbt komma i gång med utvecklingen och gruppen fastslog att användningsfall gav oss tillräcklig vägledning vid implementation. Trots att det gick bra under implementeringsfasen uppkom problem då systemet skulle utvärderas. Hade systemet beställts av en extern kund hade det varit svårt att verifiera systemets funktionalitet på ett konventionellt sätt. Det beror på att kravspecifikationen ofta används för att reflektera kundens behov på ett mer detaljerat sätt än vad användningsfall gör. Således har avsaknaden av kravspecifikation både givit oss för- och nackdelar, emellertid anses nackdelarna väga upp fördelarna. I efterhand hade således en kravspecifikation varit att föredra.

Den mjukvaruutvärdering som utfördes har sin grund i ISO 9126-1:2001 vilket leder till en brist på metriker och alla resultat som fås fram är baserade på åsikter. Eftersom det är gruppens egna medlemmar som gjort utvärderingen är åsikterna partiska, vilket är just vad ISO 9126 ämnar motverka. Att ta fram testfall är en lösning för att avlägsna bias genom att låta externa testare provköra systemet, men ej heller det är gjort. De tre restrerande delarna av ISO 9126 hade bidragit med metrik för att mäta kvaliteten på mjukvaran. All kvalitetskoll av källkod är således gjord enbart med gruppens gemensamma kunskap om vad som är korrekt pragmatik, vilket även det leder till bias. Däremot hjälpte de tester som gjordes i gruppen att hitta en del brister, vilket ses som en framgång. Således var det inte en helt umbärlig aktivitet.

Då webbapplikationen utvecklades med programmeringsspråk och ramverk som ingen i gruppen hade tidigare erfarenhet av lades mycket tid på utbildning vilket har tagit tid ifrån den slutliga produkten. Att utveckla systemet i ett helt nytt språk kan därför ses som en nackdel, men det har till slut givit gruppens medlemmar en bredare teknisk bas att stå på i framtiden och Ruby on Rails har fungerat utmärkt för det ändamål för vilket det har använts. Således är valet av ett nytt programmeringsspråk ändå lyckat då det resulterat

i mer kunskap hos gruppens medlemmar.

6.3 Vidare arbete

För vidare utveckling av systemet Elbipo finns det ett par områden som bör fokuseras på, både vad gäller systemet och själva mjukvarusystemet.

Systemet, den fritt flytande elbilspoolen, som utvecklats är skapat utan stark förankring till vad som är ekonomiskt eller ur lagens perspektiv genomförbart, vilket kan vara problematiskt. Systemet skulle kunna utökas till att ha en högre ekonomisk medvetenhet, genom att kunna avgöra vilka av kundernas interaktioner med systemet som borde debiteras. Exempelvis har frågan huruvida en avbokning ska debiteras ej undersökts noggrannt i detta projekt, vilket kan vara en potentiell ekonomisk risk vid driftsättning. Även legala aspekter hos systemet bör tas i åtanke, för exempelvis registrering av nya kunder. Då varje kund kommer att kunna använda verksamhetens fordon föreligger viss juridisk problematik i form av exempelvis framtagande av avtal. Även detta bör undersökas för att komplettera systemet.

För fortsatt utveckling av mjukvaran finns ett par förslag till fortsättning. Primärt kan det vara fördelaktigt att fastställa en formell utvecklingsprocess då det finns flera problem med nuvarande process. Förslagsvis kan en formell kravspecifikation skrivas för att förenkla testning och validering av systemets funktioner. Användartester borde göras för att testa systemets användarvänlighet då åsikter av användare effektivt kan visa förbättringsområden.

6.4 Slutsatser

Till grund för detta kapitel ligger till stor del den kvalitetsutvärdering som gjordes. Då risken är stor att den är påverkad av mänskligt bias implicerar det vidare att även detta diskussionskapitel är påverkat av det. För att kunna bekräfta eller refusera de slutsatser som dras bör ytterligare, mindre subjektiva användartester utföras för att kunna bedöma huruvida systemet är enkelt, bekvämt och attraktivt att använda.

Ingen i kandidatgruppen avser att fortsätta utvecklingen av Elbipo. Ej heller är mjukvaran färdig längre än till alfastadie. Trots det visar detta kandidatarbete på hur ett IT-system kan skapas för att ge oss kraftfulla verktyg för att reducera den negativa påverkan dagens samhälle har på miljön. Vi tror också att en vidareutveckling av Elbipo skulle vara högst användbart för ett företag med mål att hyra ut miljösmarta bilar.

Referenser

Abramowski L. och Holmström A. (2007) *RES 2005–2006 Den nationella resvaneundersökningen.* Östersund: SIKA.

Android Developers. (2012a) *Platform Versions.* http://developer.android.com/resources/dashboard/platform-versions.html. (2012-05-03).

Android Developers. (2012b) *Application Fundamentals.* http://developer.android.com/guide/topics/fundamentals.html. (2012-05-03).

Android Developers. (2012c) *Android Manifest.* http://developer.android.com/guide/topics/manifest/manifest-intro.html. (2012-05-10).

Android Developers. (2012d) *Android API Levels.* http://developer.android.com/guide/appendix/api-levels.html#fc. (2012-05-10).

Car2go. (2012) *The mobility concept.* Car2go, Washington, DC. http://www.car2go.com/washingtondc/en/concept/. (2012-05-03).

Chester, M., Horvath, A. och Madanat, S. (2010) *Parking infrastructure: energy, emissions, and automobile life-cycle environmental accounting.* Environmental Research Letters, vol. 5, nr. 3.

Davis, A. Y., Pijanowski, B. C., Robinson, K. och Engel, B. (2010) *The environmental and economic costs of sprawling parking lots in the United States.* Land Use Policy, vol. 27, nr. 3, ss. 255-261.

Fielding R. T. (2000) *Architectural Styles and the Design of Network-based Software Architectures.* Irvine: University of California

Free Website Templates. *Pollena Minima.* http://www.freewebsitetemplates.com/preview/minimalisticwebtemplate/. (2012-05-05).

Ruby on Rails Guides. (2012a) *Getting Started with Rails.* http://guides.rubyonrails.org/getting_started.html. (2012-05-03).

Ruby on Rails Guides. (2012b) *Ruby on Rails 2.3 Release Notes.* http://guides.rubyonrails.org/2_3_release_notes.html. (2012-05-03).

Statistiska centralbyrån. (2011) *Fordonsbestånd 2010.* http://www.scb.se. (2011-01-28).

Sunfleet. (2012) *Bil bara när du vill. Vi förenklar din vardag. - Sunfleet.* http://www.sunfleet.com. (2012-05-05).

Swedish Standards Institute. (2001) *Software engineering - Product quality - Part 1: Quality model.* http://www.sis.se/kvalitet/system-och-programvarukvalitet/ss-iso-iec-9126-1. (2012-05-03).

SQA Software Quality Assurance. (2012) *ISO 9126 Software Quality Model.* http://www.sqa.net/iso9126.html. (2012-05-03).

Ulrich, J. (2011) Hashing Passwords. *DShield; Cooperative Network Security Community - Internet Security.* http://www.dshield.org/diary.html?storyid=11110. (2012-05-03).

Zipcar (2012).*Car Sharing, an alternative to car rental and car ownership.* http://www.zipcar.com. (2012-05-05).

A Use Cases

A.1 Prioritering av use cases

Kärnfunktionalitet

- Boka en bil
- Starta hyrning
- Avsluta hyrning
- Söka laddstationer
- Logga in användare
- Logga ut användare
- Ändra användaruppgifter
- Avlägga statusrapport

Mindre viktig funktionalitet

- Avboka bil
- Lägga till bil
- Lägg till användare
- Läff till laddstation
- Ta bort bild
- Ta bort användare
- Ta bort laddstation

A.2 Detaljerade use cases

A.2.1 Boka en bil

Mål Genomföra bokning av en bil
Aktör Kund, Operatör

Main flow

1. Aktören väljer tid och plats för avhämtning av bil.
2. **Antagande** Aktörens val följer reglerna.
3. Systemet letar efter bilar som passar aktörens kriterier.
4. **Antagande** Det finns minst en bil som passar aktörens kriterier.
5. System presenterar de n närmsta bilarna och praktisk information om dem (karossform, märke, storlek m.m).
6. Aktör väljer en av bilarna att boka.
7. **Antagande** Bilen är fortfarande ledig att boka för denna tid.

8. Systemet bokar bilen.
9. System presenterar aktör med utförligare information om den bokade bilen för att underlätta identifiering på plats (registreringsnummer, färg etc).

Alternative flow

2a **Antagande** Aktörens val följer ej reglerna.

1 System informerar aktör om vilken begäran som bröt mot reglerna.
2 Gå till 1.

4a **Antagande** Det finns ingen bil som passar aktörens kriterier.

1 System informerar aktör om att det inte finns några bilar som passar kriterierna.
2 Gå till 1.

7a **Antagande** Bilen blev bokad av någon annan aktör medan nuvarande aktör valde bil.

1 System berättar att bilen ej längre är tillgänglig.
2 Gå till 3.

Post condition IF Bil passar kunds kriterier
Bil bokad för kund
ELSE
Ingen bil bokad

A.2.2 Påbörja användning av bil

Mål Ge tillgång till bilen.
Aktör Bil

Main flow

1. Bil skickar information till system om kundkort som blivit visat
2. **Antagande** Kund finns registrerad i system
3. **Antagande** Bil är bokad av denna kund
4. System informerar bil om att den ska låsa upp dörrar och nyckellås.
5. Bil verifierar öppnandet av dörrar och nyckellås.

Alternative flow

2a **Antagande** Kund ej registrerad

1 System informerar bil om att kund ej är registrerad.

3a **Antagande** Bil är ej bokad av kund, men ledig.

1 Systemet bokar bilen för denna kund.

2 Gå till 4.

3a **Antagande** Bil är bokad av annan kund

1 System informerar bil om att begärande kund inte kan hyra bilen vid detta tillfälle.

Post condition Påbörja användning av bokad bil

A.2.3 Avsluta användning av bil

Mål Avsluta hyrning och lås dörrarna.
Aktör Bil

Main flow

1. Bil meddelar systemet om att kund vill avsluta hyrning.
2. **Antagande** Nyckel är fastlåst och bilen står på tillåten plats.
3. Systemet informerar bilen om att den ska låsa dörrarna.
4. Bil låser dörrarna.
5. Systemet uppdaterar bilens status

Alternative flow

2a **Antagande** Nyckel är ej inlåst i box.

1 System informerar bilen om att nyckeln ej är inlåst.

Post condition IF Bil står på tillåten plats AND Nyckel är fastlåst
THEN Hyrning av bil avslutad
ELSE System meddelar bil om vad som gått snett.

A.2.4 Söka laddstationer

Mål Leta upp närliggande laddstationer
Aktör Kund, Operatör

Main flow

1. Aktör söker efter laddstationer på given plats
2. System kontrollerar närliggande laddstationer
3. Aktör får information om närliggande laddstationer

Post condition Returnera närliggande laddstationer

A.2.5 Logga in användare

Mål Logga in användare
Aktör Kund, Operatör, Administratör

Main flow

1. Aktör fyller i användarnamn och lösenord
2. **Antagande** Användarnamn och lösenord finns registrerade.
3. Systemet startar session för användare och presenterar aktören med användargränssnittet
4. Aktör är nu inloggad i systemet.

Alternative Flow

2a **Antagande** Användarnamn och lösenord finns ej registrerade.

1 System meddelar användare om felaktiga inloggningsuppgifter.

2 Gå till 1.

Post condition Aktör inloggad

A.2.6 Logga ut användare

Mål Logga ut användare
Aktör Kund, Operatör, Administratör

Main flow

1. Aktör meddelar systemet att han/hon vill logga ut.
2. **Antagande** Aktör inloggad
3. Systemet avslutar sessionen för aktör.
4. System notifierar aktör om lyckad utloggning.

Alternative Flow

2a **Antagande** Aktör inte inloggad

1 System meddelar användare om att det inte fanns någon aktiv session

2 Aktör skickas vidare till första sidan

Post condition Aktör utloggad

A.2.7 Ändra användaruppgifter

Mål Förändra befintliga användaruppgifter hos en användare
Aktör Kund, Operatör

Main flow

1. Aktör väljer att editera en användare
2. System skickar data relaterat till användare
3. Aktör ändrar
4. System ändrar data i databasen
5. Aktör får meddelande om att data har ändrats

Post condition Valda data ändrade i databasen

A.2.8 Avlägga statusrapport

Mål Status rapportering från extern hårdvara till system
Aktör Bil, Laddstation

Main flow

1. Aktör kontaktar system med all information den har tillgänglig för sitt nuvarande tillstånd.
2. System verifierar data och sparar undan.
3. System skickar godkännande av data till aktör.

Post condition Information om aktör registrerat i databasen

A.2.9 Avboka bil

Mål Ta bort bokning av bil
Aktör Kund, Operatör

Main flow

1. Aktör väljer att avboka en bil
2. System hämtar alla bokningar som aktör får modifiera.
3. System presenterar bokningarna för aktör.
4. Aktör väljer en bokning i listan.
5. System låter aktör verifiera borttagning.
6. **Antagande** Aktör verifierar avbokning.
7. System tar bort kopplingen mellan bokningen och användare från databasen
8. Aktör får meddelande som bekräftar borttagning.

Alternative flow

5a **Antagande** Aktör verifierar ej bokning.

1 Avbokning avbryts.

Post condition Koppling av bokning till kund borttagen

A.2.10 Lägg till bil

Mål Lägga till en bil i databasen
Aktör Administratör

Main flow

1. Aktör väljer "lägg till bil" under administratörsgränssnitet
2. Aktör fyller i information om den nya bilen
3. **Antagande** Systemet verifierar korrekt indata
4. Aktören presenteras med en konfirmationsdialog
5. Systemet lägger till bilen i databasen

Alternative flow

3a **Antagande** Indata är ej korrekt

1 Aktören blir informerad om var felet ligger
2 Aktören ändrar indata till vad det skulle varit
3 Återgå till 2

Post condition Bil tillagd i databasen

A.2.11 Lägg till användare

Mål Användare läggs till i databasen
Aktör Operatör, Administratör

Main flow

1. Aktör väljer "lägg till användare" under operatörsgränssnitet
2. Aktör fyller i information om ny användare
3. **Antagande** Systemet verifierar korrekt indata
4. Aktören presenteras med en konfirmationsdialog
5. Systemet lägger till användare i databasen

Alternative flow

3a **Antagande** Indata är ej korrekt

1 Aktören blir informerad om var felet ligger

2 Aktören ändrar indata till vad det skulle varit

3 Återgå till Main Flow

Post condition Användare tillagd i databasen

A.2.12 Lägg till laddstation

Mål Lägga till laddstationer i databasen
Aktör Administratör

Main flow

1. Aktör väljer "lägg till laddstationer" under administratörsgränssnitet
2. Aktör fyller i information om ny laddstation
3. **Antagande** Systemet verifierar korrekt indata
4. Aktören presenteras med en konfirmationsdialog
5. Systemet lägger till laddstation i databasen

Alternative flow

3a **Antagande** Indata är ej korrekt

1 Aktören blir informerad om var felet ligger

2 Aktören ändrar indata till vad det skulle varit

3 Återgå till Main Flow

Post condition Laddstation tillagd i databasen

A.2.13 Ta bort bil

Mål Ta bort en bil från databasen
Aktör Administratör

Main flow

1. Aktör väljer “ta bort bil” i administratörsgränssnitet
2. Aktör presenteras med nuvarande bilar i systemet
3. Aktör väljer bil att ta bort
4. Aktör presenteras med konfirmations dialog
5. Systemet tar bort bil ur databas

Post condition Bil borttagen från databas

A.2.14 Ta bort användare

Mål Ta bort användare från databasen
Aktör Operatör, Administratör

Main flow

1. Aktör väljer “ta bort användarel” i operatörsgränssnitet
2. Aktör presenteras med nuvarande användare i systemet
3. Aktör väljer användare att ta bort
4. Aktör presenteras med konfirmations dialog
5. Systemet tar bort användare ur databas

Post condition Användare borttagen från databas

A.2.15 Ta bort laddstation

Mål Ta bort laddstationer från databasen
Aktör Administratör

Main flow

1. Aktör väljer “ta bort laddstation” i administratörsgränssnitet
2. Aktör presenteras med nuvarande laddstationeri systemet
3. Aktör väljer laddstation att ta bort
4. Aktör presenteras med konfirmations dialog
5. Systemet tar bort laddstation ur databas

Post condition Laddstation borttagen från databas

B API

Detta dokument innehåller information om det API som elbilspoolssystemet Elbipo tillhandahåller. Vissa delar är öppna och går att anropas hur som helst, för detta API kommer specifikationer att släppas. Den stängda delen är avsedd för system i bilar och i laddstolpar, och är ämnade för intern kommunikation endast. Formatet JSON kommer att användas till samtliga svar, och för vissa förfrågningar. De förfrågningar som inte använder sig av JSON är GET-förfrågningar, sådana som bara hämtar data utan att modifiera den.

Detta API är uppbyggt enligt RESTful-arkitektur. Detta innebär att alla förfrågningar och manipuleringar som skickas till API:t skickas genom HTTP-protokollet.

B.1 Öppet API

Namn: Hitta bilar
Url: GET /cars
Parametrar:

	addr	Address
frivillig	**lim**	Antal bilar
	lat	Latitud
	lng	Longitud
frivillig	**lim**	Antal bilar

Svar:
Lista med bilar.

Beskrivning:
Hämtar en lista av bilar ordnade efter avstånd från Address-parameter. Antalet bilar kan begränsas med Antal-parameter. Latitud och Longitud kan anges istället för address vid behov.

Parameterintervall:
Address: Gatuadress med eller utan nummer
Latitud: Horisontell geografisk koordinat
Longitud: Vertikal geografisk koordinat
Antal: Siffra mellan 1 och 20. Om ej satt eller satt till över 20 svarar funktionen med 20 bilar.

Exempel på svar:


```
{
 "charge": 64,
 "is_charging": true,
 "km_driven": 1462,
 "pos_lat": 12.1192,
 "pos_long": 57.6327,
 "regnr": "xtv778",
 "car_model": {
        "body": "Compact",
        "make": "Smart",
        "name": "ED",
        "price": 0.3,
        "range": 120,
        "year": 2010
 }
}
```

Namn: Hitta Laddstationer
Url: GET /chargestations
Parametrar:

	addr	Address
frivillig	**lim**	Antal stationer
	lat	Latitud
	lng	Longitud
frivillig	**lim**	Antal stationer

Svar:
Lista med laddstationer.

Beskrivning:
Hämtar en lista av laddstationer ordnade efter avstånd från Adress-parameter. Antalet laddstationer kan begränsas med Antal-parameter. Latitud och Longitud kan anges istället för address vid behov.

Parameterintervall:
Address: Gatuadress med eller utan nummer
Latitud: Horisontell geografisk koordinat
Longitud: Vertikal geografisk koordinat
Antal: Siffra mellan 1 och 20. Om ej satt eller satt till över 20 svarar funktionen med 20 stationer.

Exempel på svar:

```
{
 "id": 8,
 "nr_free": 1,
 "nr_spot": 1,
 "pos_lat": 12.1712,
 "pos_long": 57.6583
}
```

Namn: Verifiera Användare
Url: GET /access/remote_login
Parametrar:

	cust	Användare
	pw	Lösenord

Svar:
Hashat lösenord.

Beskrivning:
Hämtar ett hashat lösenord för användare. Används vid bokning av bil från mobil applikation.

Parameterintervall:
Användare: Användarens email
Lösenord: Lösenord för användare i klartext

Exempel på svar:

```
{
"success": true,
 "password":  4f646d2c80fd3abc0a25a69ec601538e7cc291a1
}
```

Namn: Lägg till en bokning
Url: POST /book/add_remote
Parametrar:

cust	E-postadress.
hashpw	Hashat lösen.
car	Registreringsnr.

Svar:
Resultat på bokningsförsöket.

Beskrivning:
Försöker boka en bil för användaren specificerad i parametern. Kunden autentiseras med hjälp av hashat lösen han/hon fått genom Verifiera Användare-metoden. Om bilen redan är bokad eller kunden redan har en bokning returneras felmeddelande och ingen bokning görs.

Parameterintervall:
E-postadress: Användarens e-postadress.
Hashat lösen: Det hashade lösenord som fåtts genom Verifiera användare-metoden.
Registeringsnummer: Registreringsnummer hos en elbil.

Exempel på svar:

```
{
  "success": true,
  "booking_nr": "FA2DS456LV"
}
```

B.2 Internt API

Namn: Uppdatera bils status
Url: PUT /cars/status
Parametrar:

car	Bil
lat	Latitud
lng	Longitud
charge	Laddning
charging	Laddar
km	Sträcka

Svar:
Bekräftelse

Beskrivning:
Uppdaterar status för en bil. En bils status innehåller data som var den befinner sig, laddstatus, huruvida den laddas nu och hur långt den kört.

Parameterintervall:
Bil: Bilens *Vehicle identification number*
Latitud: Horisontell geografisk koordinat
Longitud: Vertikal geografisk koordinat
Laddning: Procentuell batteristatus
Laddar: Boolean
Km: Bilens totala körsträcka

Exempel på svar:

```
{
"success": true
}
```

Namn: Uppdatera Laddstationstatus
Url: PUT /chargestations/status
Parametrar:

id	Identifikation
nr_free	Antal

Svar:
Bekräftelse

Beskrivning:
Uppdaterar status för en laddstation. En status i det här fallet är bara hur många lediga platser som finns för tillfället.

Parameterintervall:
Identifikation: identifikationsnummer för laddstation
Antal: Antal lediga platser

Exempel på svar:

```
{
"success": true
}
```

Namn: Starta hyrsession
Url: PUT /cars/start_session
Parametrar:

car	Bil
cust	Användare

Svar:
Bekräftelse

Beskrivning:
Anropas när kund aktiverar en bil. Är bilen bokad av kund aktiveras bokning. Om kunden har en annan bokning stängs den och den aktuella bilen aktiveras.
Parameterintervall:
Bil: Bilens *Vehicle Identification Number*
Användare: Användarens email

Exempel på svar:

```
{
"success": true
}
```

Namn: Avsluta hyrsession
Url: PUT /cars/end_session
Parametrar:

car	Bil
cust	Användare
lat	Latitud
lng	Longitud
charge	Laddning
charging	Laddas
km	Sträcka

Svar:
Bekräftelse

Beskrivning:
Anropas när kund avaktiverar en bil. Uppdaterar även bilens status.

Parameterintervall:
Bil: Bilens *Vehicle Identification Number*
Användare: Användarens email
Latitud: Horisontell geografisk koordinat
Longitud: Vertikal geogragisk koordinat
Laddning: Procentuell batteristatus
Laddas: Boolean
Sträcka: Resans sträcka

Exempel på svar:

```
{
"success": true
}
```

C Kvalitetsutvärdering

C.1 Functionality

Styrkor

- De användningsfall som var högt prioriterade är implementerade.
- Mjukvaran samarbetar väl med MySQL och GoogleMaps.
- Systemet är uppbyggt med hjälp av API:er vilket ger möjlighet för utveckling av tredjepartsapplikationer.
- Kan användas med krypterad HTTPS-anslutning.
- Lösenord är hashade och saltade.

Svagheter

- Felformaterad indata på hemsidan kan generera fel.
- Finns inget registrerat SSL-certifikat för att verifiera Elbiposervern.
- Adress utanför Göteborg ger ej väntad utdata utan returnerar Centralstationen.

Förbättringar

- Skriva tester till alla funktioner.
- Administratörsläge skulle behöva implementeras.
- Lagring av personliga sök-kriterier.

C.2 Reliability

Styrkor

- Vid normalt genomförande av användningsfall uppstår inga fel.

Svagheter

- Vid de fel som upptäcktes vid medvetet försök att orsaka fel återhämtar systemet sig inte, en felsida visas.
- Vid hårdvarufel eller strömavbrott kommer datan om aktiva bokningar bli felaktig och nya bokningar kommer ej kunna göras.

Förbättringar

- Bättre och mer konsekvent felhantering.

C.3 Useability

Styrkor

- Tydlig och enkel design både på hemsidan och i Androidapplikationen.
- Lätt att navigera på hemsidan då den följer ett välanvänt mönster med alltid tillgänglig meny.
- Länkar har tydliga namn på hemsidan.
- Bokningsprocessen är snabb och enkel.
- Systemet är väldigt enkelt och lätt att lära sig.

Svagheter

- Ingen explicit förklaring till att inloggningsrutorna faktiskt är inloggningsrutor på hemsidan.
- Ingen verifiering för bokning av bil på hemsidan.
- Bristfällig information om individuella bilar.
- Kundregistrering kan tyckas omständig.
- Finns inga prisuppgifter.
- Bokningshistorik visar inte så mycket information.

Förbättringar

- Mer explicit information genom använding av verifieringsinformation och förklarande text.
- Mer nyttig data för användaren.
- Färgkodning på bilar på kartan.

C.4 Efficiency

Styrkor

- Hemsidan är såpass snabb att man ej tänker på laddningstider. Väntetiden för laddning av ny sida går väldigt fort.

Svagheter

- Många databastransaktioner för att hämta lediga bilar.
- Onödigt mycket data hämtas ut vid databasanrop.

Förbättringar

- För ett verkligt projekt skulle Webrick-servern behöva bytas ut då den ej är gjord för att användas i produktion.
- Bättre testningar för laddningstider av sidor, CPU-användning och användning av minne borde utföras.
- Borde implementera smartare databasqueries.

C.5 Maintainability

Styrkor

- Fördelaktigt att systemet är modulärt vid förändring.
- För att göra ändringar i systemet krävs ändringar i ett begränsat antal moduler.
- Projektet följer för det mesta DRY (Don't Repeat Yourself) vilket gör att när ett fel hittas och rättas till är det tämligen säkert att det åtgärdas överallt.
- Att leta upp källan till felet är relativt lätt på grund av kombinationen av DRY och MVC.

Svagheter

- Det finns få automatiska tester.
- Lite dokumentation.

Förbättringar

- Kontrollera att alla delar i projektet slaviskt följer MVC och DRY.
- Fler automatiska tester.

C.6 Portability

Styrkor

- Finns mycket automatik vid konfiguration av databas.
- Då databasen är utvecklad med hjälp av migrations är det enkelt att utveckla den vidare.

Svagheter

- Servern måste installera Ruby on Rails och MySQL.
- Använder två externa Rubygems som inte följer med Rails.
- MySQL måste installeras separat.
- Behöver konfigurera databasen.
- Svårt att installera på annat än unix-baserade operativsystem.

Förbättringar

- Borde göra webbapplikationen mer enkel att driftsätta med Rubygems, exempelvis Capistrano.