HitchHike.se - utveckling av en webbtjänst för samåkning

Magnus Larsson

21 juni 2009
Master's Thesis in Computing Science, 30 ECTS credits
Supervisor at CS-UmU: Jerry Eriksson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Sammanfattning

Antalet bilar på Sveriges vägar och koldioxidutsläppen från dessa är en fråga som hett diskuteras i dagens samhälle, ur miljösynpunkt. En lösning för att minska biltrafiken och dess miljökonsekvenser är genom att samåka i de bilar som kör på vägarna. Denna rapport beskriver hur en webbtjänst som skall främja samåkning har utvecklats.

Målet med arbetet var att dels identifiera vad som krävs för att göra en lyckad webbtjänst för samåkning, dels att utveckla denna webbtjänst genom att skapa bra och stabil grund som är lätt att arbeta vidare på i framtiden. Tanken med projektet var att ett företag skulle skapas, HitchHike.se, som efter arbetet skulle ta över den fortsatta utvecklingen av webbtjänsten och sköta marknadsföringen.

Resultatet av arbetet blev en fungerande webbtjänst som fungerar som ett annonsforum där användare från hela Sverige kan lägga upp annonser där de söker eller erbjuder samåkning. Hela sajten är uppbyggd runt *Google Maps* för att kunna söka på adresser, visa färdvägar och markera resor. *Model View Controller*-modellen har använts när den underliggande strukturen har implementerats. Detta har gjort att webbsidan är mycket flexibel och enkel att hantera i framtiden.

HitchHike.se - developing a website for carpooling

Abstract

The high amount of vehicles on the roads in Sweden and the carbon dioxide polutions from them are a topic of many discussions today. A solution for reducing the traffic and also the consequences on the environment is to start carpooling, more then it is done today. This report describes the development of a website that will promote carpooling.

The aim of the work was first of all to identify what it takes to make a succesful website for carpooling. The second objective was to develop a website from scratch, with a solid and flexible foundation that will be easy to work with in the future. Another plan with the project was to start a company that will continue the development and marketing of the website after this project is completed.

A functional website that acts as an advertising marketplace is the result of the project. The users may add advertisements where they are looking for, or offering, a place for carpooling. The website is built around *Google Maps* which makes it possible for the user to search for addresses, view routes and mark trips on the map. To make the website flexible and easy to update, the Model View Controller model has been used to build the underlying structure.

Innehåll

Figurer

Tabeller

Kapitel 1

Introduktion

Denna rapport är skriven som en del i ett examensarbete utfört vid Institutionen för datavetenskap på Umeå Universitet. Syftet med examensarbetet är att utveckla en webbtjänst som skall göra det möjligt att på ett enkelt sätt hitta personer att samåka med vid enstaka längre resor eller resor till olika evenemang runt om i landet.

Bakgrunden till idén är de heta diskussionerna som förs angående fordons koldioxidutsläpp och att de är farliga för miljön. Genom att samåka och försöka att aldrig köra en bil tom (med bara förare) kan alla hjälpa till att minska dessa koldioxidutsläpp, och samtidigt spara pengar genom att dela kostnader för bensin, bilslitage med mera. Dessutom medför färre antal fordon på vägen, färre antal olyckor samt minskat vägslitage. Det finns ingen uppdragsgivare till detta projekt, men företaget HitchHike.se kommer att skapas för att kunna utveckla sidan vidare i framtiden.

1.1 Bakgrund

Detta projekt startade genom att en bekant sedan tidigare kontaktade mig och hade en bra företagsidé som skulle hjälpa till att förbättra miljön. Han behövde en utvecklare till projektet och vi bestämde att vi skulle skapa företaget gemensamt och att jag skulle börja med utvecklingen av systemet under våren 2009, som en del av mitt examensarbete. Grundtanken med företaget var att arbeta för en bättre miljö, genom att försöka minska mängden koldioxidutsläpp i Sverige, framförallt från personbilar.

Planen för att åstadkomma detta var att utveckla en webbtjänst för samåkning och pendling. En användare av sidan skulle kunna hitta personer att samåka med genom att lägga ut en kontaktannons. Både personer som har bil och de som inte har bil skulle kunna lägga ut annonser där de antingen söker någon att resa med, eller söker personer som behöver någon att åka med. Företagets motto är ”Ingen bil skall gå tom”.

Projektet i sig hade en enkel grundtanke. Svårigheten var och förblir att få människor i Sverige att använda tjänsten. Ingen vetenskaplig undersökning som undersöker människors inställning till samåkning utfördes i anslutning till projektet, men de flesta personerna som informellt tillfrågades sade sig vara positiva till samåkning, och ansåg att hela konceptet var en bra idé. När det väl gäller verkade den allmänna uppfattningen dock vara att det känns besvärligt att samåka, oftast på grund av att viss planering och extra tid krävs. Den tanken vill vi få bort från människor och i stället få dem att tänka på alla positiva effekter samåkning ger:

- Förbättrar miljön genom att minska framförallt koldioxidutsläppen, men även andra utsläpp som bilarna avger.
- Minskar slitaget på vägarna samt troligtvis antalet trafikolyckor eftersom det blir mindre trafik på vägarna.
- Man får allt som oftast trevligt sällskap under resan och det är roligare än att åka själv. Det kan även vara en möjlighet att utöka sitt kontaktnät.
- Alla sparar pengar på samåkning, genom att dela på kostnaderna för resan.
- De som inte har någon egen bil sparar tid jämfört med att åka med andra kollektiva transportmedel, exempelvis buss, eftersom man slipper ta sig till och från bussen samt att bussen kör långsammare och stannar på flera ställen. Observera att målet inte är att konkurrera ut kollektivtrafiken, samåkning skall ses som ett bra komplement till kollektivtrafiken.

Enligt vägverkets undersökningar[1] och egna erfarenheter kan människor vara lite rädda för samåkning, dels eftersom man eventuellt skall åka med helt okända människor, dels för att det kan kännas svårt och besvärligt att ta kontakt med folk. Dessa tankar har beaktats under hela projektet och åtgärder har vidtagits för att försöka minska problemen. Webbtjänsten konstruerades på ett sådant sätt att den skulle vara enkel att använda, lättöverskådlig och enhetlig. För många funktioner, flashiga animeringar och för mycket information på sidan tenderar att förvirra mer än att hjälpa. För att varje enskild användare skall känna sig säkrare på personerna de samåker med bestämdes det att följande åtgärder skulle vidtas för sajten:

- Man måste vara registrerad användare för att kunna anmäla sitt intresse på en resa eftersom det då är svårare att utge sig för att vara någon annan än den man egentligen är.
- När man anmäler sitt intresse på en resa visas den egna kontaktinformationen för den som har lagt ut annonsen och de båda parterna måste därefter ta kontakt med varandra innan det bestäms om personerna skall samåka.
- När resan är genomförd skall ett omdöme om respektive person lämnas. Andra personer kan sedan kolla på omdömen och se om personen i fråga har skött sig på tidigare resor.

1.2 Mål och syfte

Det övergripande målet med hela projektet, *HitchHike.se*, var att skapa en fullt fungerande webbtjänst samt ett företag för att förbättra miljön genom att att öka samåkningen i Sverige. Sett till det examensarbete denna rapport behandlar var målet och syftet att skapa en enkel och lättanvänd webbtjänst för samåkning med en bra och stabil grund att bygga vidare på i framtiden. Fokus har först och främst lagts på att designa den underliggande strukturen på sajten på ett väl fungerande sätt. Alla önskade funktioner (se framtida arbete sektion 6.3 på sidan 42) har kommit i andra hand, eftersom de är enklare att lägga till i framtiden, jämfört med om hela sajtens grundstruktur måste byggas om.

[1] http://www.vv.se/Startsida-foretag/Trafiken/Hallbart-resande/Stod-i-arbetet/Samakning/, 2009-05-31

Innan projektets detaljplaneringen startade definierades några stora funktionsmässiga mål med sajten. Hur de skulle implementeras och exakta detaljer beslutades det om senare. De viktigaste funktionsmässiga målen för projektet var att det skulle vara möjligt att:

- Registrera en användare.
- Lägg upp en annons att man söker någon att samåka med.
- Söka bland de befintliga annonserna.
- Registrera sitt intresse på annonser.

1.3 Tillvägagångssätt

Arbetet har utförts med en iterativ utvecklingsprocess, vilket har gjort det möjligt att gå tillbaka till varje distinkt steg i processen och göra förändringar. Under vissa perioder har arbete skett ett steg åt gången, medan det i vissa perioder har förekommit arbete i flera steg samtidigt. Arbetet har passerat följande moment:

- Kravinsamling – Alla funktionella och icke-funktionella krav på sidan arbetas fram.
- Teknisk planering – Innefattar den underliggande system-designen samt den utseendemässiga designen.
- Implementation – Webbtjänstens utveckling utifrån den plan som arbetats fram.
- Testning – Sajten funktionstestas för att minimera antalet fel och buggar.
- Dokumentation – Denna rapport skrivs, även en blogg om arbetet finns tillgänglig[2].

1.4 Relaterat arbete

Det finns ett antal webbtjänster i Sverige som på något eller några sätt hanterar samåkning, men inga som är uppenbart välkända i Norrland. De befintliga tjänsterna har analyserats för att hitta fördelar och nackdelar med dem. Nackdelarna har noterats och lärdom har tagits av dem. Fördelarna, tillsammans med egna idéer, har använts för att kunna planera de funktionsmässiga kraven på *HitchHike.se* för att försöka få en optimal webbtjänst. Nedan listas de befintliga tjänsterna som har analyserats under projektets gång:

- http://www.samaka.se/
- http://www.samakning.se/
- http://www.bilplats.se/
- http://www.pendlarservice.se/
- http://www.samakningstjanst.se/php/welcome.php

[2]http://magnus-larsson.blogspot.com/

1.5 Rapportens disposition

Nedan återfinns en beskrivning av rapportens disposition.

Kapitel 2 beskriver i kronologisk ordning hur arbetet har utförts.

Kapitel 3 ger en teoretisk beskrivning av hur sidan är uppbyggd och hur de tekniska lösningarna fungerar.

Kapitel 4 redogör för projektets fördjupningsstudie som behandlar webbsäkerhet.

Kapitel 5 visar resultaten av projektet.

Kapitel 6 diskuterar resultaten och även problem som har uppkommit samt hur framtiden för webbtjänsten ser ut.

Kapitel 2

Arbetets gång

Detta kapitel beskriver hur hela projektet har genomförts, från planering till färdig produkt. En iterativ utvecklingsprocess har använts eftersom det i början var svårt att planera för hela arbetet och veta säkert att misstag inte gjorts i planeringen. Med den iterativa processen blev det möjligt att utföra allting i mindre steg och modifiera planeringen vid behov.

2.1 Kravinsamling

Grundtanken med sidan var tydlig från början, den skulle göra det enklare att hitta någon att samåka med, men exakt hur det skulle genomföras var långt ifrån bestämt. Det första som gjordes var att diskutera och spåna på idéer för sidan. Som inspirationen användes dels samåkningssajterna nämnda i sektion 1.4 på sidan 3, och dels några stora annons-sajter som *Blocket.se* och *Tradera.se*. Traditionella annons-sajter användes eftersom HitchHike.se i grunden skulle vara en webbtjänst där kontakter förmedlas, det vill säga i samma syfte som *Blocket.se* och *Tradera.se*.

En uppsättning funktionella och icke-funktionella krav identifierades, vilka definierar funktionerna som skulle finnas tillgängliga i tjänsten och hur de skulle uppföra sig. Ett funktionellt krav beskriver en funktion hos systemet, till exempel att det skall gå söka bland annonserna. Ett icke-funktionellt krav beskriver hur systemet skall uppföra sig, till exempel att webbtjänsten måste fungera i både *Internet Explorer* och *Firefox*. Resterande del av denna sektion beskriver de funktionella och icke funktionella kraven som skulle implementeras.

2.1.1 Aktörer

Aktörer i systemet kallas för användare. En användare av sidan kan befinna sig i två tillstånd, inloggad eller inte inloggad. Om användaren är inloggad innebär det att hon har registrerat sig som en användare på sidan, samt autentiserat sig genom att skriva in sitt lösenord och email adress. Det är endast inloggade användare som skall kunna nyttja systemet fullt ut. Detta beror för det första på säkerhetsaspekterna som nämndes i sektion 1.1 på sidan 2. För det andra beror det på att det är tekniskt svårt att hantera bland annat resor om det inte är känt i systemet vem som skapat resor och så vidare. Om en användare benämns som en ”annonsör” innebär det att en inloggad användare har skapat en annons och är ansvarig för den annonsen.

2.1.2 Funktionella krav

De funktionella kraven har delats upp i olika sektioner för att strukturera upp alla funktioner. Om speciella attribut är kopplade till en funktion visas de i samband med funktionen.

Användare

1. En användare på sidan skall kunna registrera sig för att få tillgång till fler funktioner. Följande attribut skall anges vid registreringen för att möjliggöra kontakt med användaren.
   (a) För- och efternamn
   (b) Email adress
   (c) Lösenord
   (d) Mobiltelefonnummer
   (e) Födelsedag
2. Användaren måste bekräfta sin registrering innan den börjar gälla.
3. När användaren registrerar sig skall ett bekräftelse-mail skickas till henne, vilket skall används för bekräftelse av registreringen.

Annonser

1. En användare som är inloggad skall kunna skapa en annons där hon söker eller erbjuder samåkning.
2. Det skall finnas två typer av annonser (resor)
   (a) Samåkningsresa – den ”vanliga” formen av samåkning för lite längre resor, exempelvis hem till föräldrar och så vidare.
   (b) Evenemangsresa – när personen vill samåka till ett specifikt evenemang som skall ske. Exempel på evenemang är idrottsevenemang, konserter eller dylikt.
3. En samåkningsresa har följande attribut:
   (a) Vart resan startar
   (b) Vart resan slutar
   (c) Vilka orter resan går genom. Måste anges om resan inte går genom den av systemet föreslagna färdvägen.
   (d) Vilket datum avresan sker
   (e) Vilken tid avresan sker
   (f) Om restiden är flexibel
   (g) Vilket fordon som används vid resan om användaren erbjuder samåkning
   (h) Antal passagerare som ryms i bilen om annonsören erbjuder samåkning eller antal passagerare som vill följa med om annonsören söker samåkning.
   (i) Övrig information om resan i textform
4. En evenemangsresa har följande attribut:
   (a) Vart resan startar
   (b) Tid och datum för avresa

   (c) Om restiden är flexibel
   (d) Tid och datum för hemresa
   (e) Vilket fordon som används vid resan om användaren erbjuder samåkning
   (f) Antal passagerare som ryms i bilen om annonsören erbjuder samåkning eller antal passagerare som vill följa med om annonsören söker samåkning.
   (g) Övrig information om resan i textform
   (h) Vilket evenemang resan sker till. Ett evenemang har följande attribut:
      i. Titel på evenemanget
      ii. Startdatum och tid
      iii. Slutdatum och tid
      iv. Vart evenemanget äger rum
      v. Typ av evenemang
      vi. Beskrivning av evenemanget

5. Ett fordon i en annons har följande attribut:
   (a) Vem som förfogar över fordonet
   (b) Registreringsnumret
   (c) Om fordonet är allergivänligt med avseende på päljsdjur eller dylikt
   (d) Om fordonet är rökfritt
   (e) Vilken typ av fordon det är. Endast personbil finns tillgängligt i början men fler typer är planerade (taxi, lastbil med mera)
   (f) Övrig information om fordonet i textform
6. Om en användare är intresserad av en annons skall hon kunna registrera sitt intresse på annonsen.
7. När en användare registrerar sitt intresse på en annons skall hon få tillgång till annonsörens kontaktuppgifter. Ett mail skall även skickas till annonsören med information om att det finns en intresserad person, tillsammans med den personens kontaktuppgifter.
8. En användare skall bara kunna registrera sitt intresse på en resa om hon är inloggad.
9. Annonsören skall själv godkänna de personer som skall åka med henne/honom.
10. Annonsören skall kunna avboka en intresserad persons plats och en person som har anmält sitt intresse skall även själv kunna avboka sin plats.

Sökning

1. Det skall vara möjligt att söka bland alla annonser, utan att användaren är inloggad.
2. För en samkåningsresa skall användaren kunna söka resor på följande parametrar:
   (a) Om användaren söker resor där annonsören har bil, där annonsören söker någon att åka med eller både och.
   (b) Mellan vilka två datum användaren kan tänka sig åka, tidigast avfärd och senast avfärd.
   (c) Vilken ort eller län användaren vill åka från.

   (d) Vilken ort eller län användaren vill åka till.
   (e) Om fordonet måste vara allergivänligt.
   (f) Om fordonet måste vara rökfritt.

3. För en evenemangsresa skall användaren först välja vilket evenemang hon skall åka till och sedan söka bland de resor som går till det evenemanget.
4. När ett evenemang skall hittas görs sökning på följande attribut:
   (a) Vilken ort eller län evenemanget hålls på.
   (b) Vilken typ av evenemang resan går till.
   (c) Fritextsökning på evenemangets namn och beskrivning.
5. När en specifik annons för ett evenemang skall väljas skall följande attribut anges:
   (a) Om användaren söker resor där annonsören har bil, där annonsören söker någon att åka med eller både och.
   (b) Mellan vilka två datum användaren kan tänka sig åka, tidigast avfärd och senast avfärd.
   (c) Vilken ort eller län användaren vill åka från.
   (d) Om fordonet måste vara allergivänligt.
   (e) Om fordonet måste vara rökfritt.

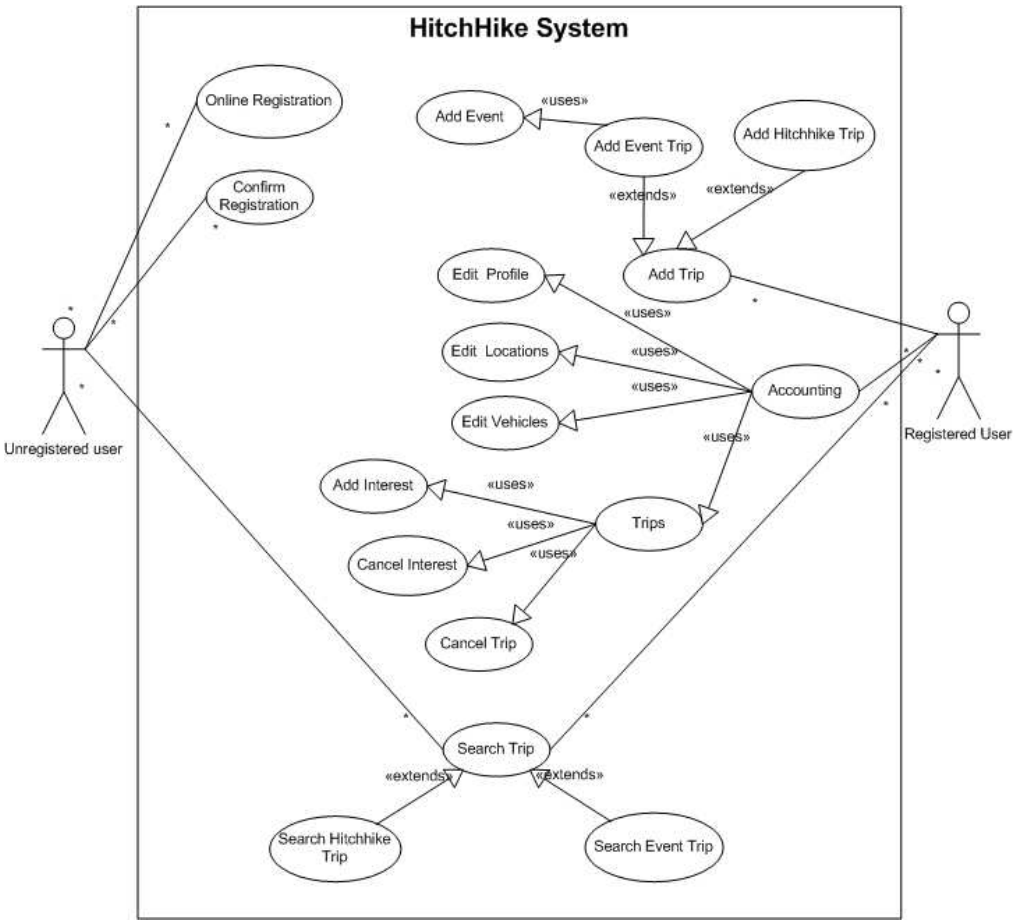
2.1.3 Icke-funktionella krav

De icke-funktionella kraven är färre, men dock väldigt viktiga. De listas här nedan:

1. Den underliggande strukturen på webbtjänsten skall vara väl genomtänkt och enkel att arbeta med och uppdatera i framtiden.
2. Webbtjänsten skall vara lättanvänd och inte ha för mycket information som syns hela tiden så att användarna blir förvirrade.
3. Webbtjänsten skall se bra ut, i den mening att den skall vara behaglig att kolla på och använda samt att alla sidor skall vara konsistenta i sitt utseende.
4. Sajten måste passa ”alla” befintliga skärmupplösningar för stationära och bärbara datorer. Handdatorer, mobiltelefoner och liknande enheter är undantagna.
5. Sidan måste fungera och visas korrekt i åtminstone de två största webbläsarna *Internet Explorer* och *Firefox*.
6. Sidan skall ta hänsyn till de relevanta säkerhetsaspekterna som diskuteras i fördjupningsstudien i kapitel 4 på sidan 21.

2.2 Planering och modellering

Med hjälp av de funktionella och icke-funktionella kraven arbetades det fram en underliggande modell av sajten som förklarar hur den skulle vara uppbyggd. Denna modell visualiseras och beskrivs i kapitel 3. Vidare konstruerades det en användningsfallmodell som beskriver vilka funktioner som skulle finnas i systemet och vilka roller en användare skulle kunna anta. Användningsfallmodellen illustreras i figur 2.1 och redogör för vad en aktör/användare i systemet kan göra beroende på vilket tillstånd (inloggad eller ej) hon är i.



Figur 2.1: Användningsfall för hela systemet, byggd utifrån de funktionella kraven i sektion 2.1.2

Målet med projektet var, som det nämndes i sektion 1.2 på sidan 2, att skapa en bra och stabil grund för sajten som är lätt att vidareutveckla. För att åstadkomma detta genomfördes en del efterforskningar om hur webbutveckling brukar gå till. Model View Controller (MVC) modellen [9] var ett bra alternativ som utvecklingsmodell eftersom den separerar design, modeller och kontrollenheter från varandra, vilket gör att varje del kan bytas ut utan att påverka resterande delar av systemet. Förutom att följa MVC-modellen har även den objektorienterade tanken utnyttjats. Klasser har använts till största möjliga mån för att kunna återanvända kod.

2.2.1 Kartor och färdvägar

Redan från projektets början var tanken att geografiska kartor, där resor är utmarkerade, skulle vara en del av sajten. Detta eftersom kartor (bilder) snabbt och tydligt kan visa mycket information, det vill säga om många resor skall visas samtidigt. Det var även logiskt att inkorporera en karta på sidan eftersom samåkning är styrt av landets geografi.

Tankarna kretsade först kring att göra en egen implementation av en karta. Det konstaterades dock snabbt att det skulle bli alldeles för tidskrävande, och förmodligen inte lika korrekt eller snyggt som redan befintliga lösningar. I stället styrdes fokus till karttjänster som redan finns på marknaden, och valet föll på *Google Maps*[1]. Andra alternativ som undersöktes var *Yahoo Maps* och *Window Live Maps*.

Några av huvudargumenten till att använda *Google Maps* var bland annat att det är gratis, det är en stor och välkänd tjänst som fungerar bra och det har de önskade funktionerna för detta projekt. Funktioner som framförallt efterfrågades var möjligheten att få vägbeskrivningar mellan orter, att hitta adresser på kartan samt att smidigt och enkelt kunna bädda in kartan i webbtjänsten. Under utvecklingen av sajten visade det sig att *Google Maps* har allt som behövdes och mer därtill.

2.2.2 Tekniska lösningar

En mängd övriga tekniker skulle användas i projektet. Det var viktigt att alla tekniker skulle vara gratis att använda, samt relativt populära och väldokumenterade så att det skulle finnas lösningar till eventuella komplikationer. Det första valet föll på *Javascript*[2], av den anledningen att *Google Maps* använder sig av den tekniken. I och med *Javascript* blev det även aktuellt att använda *AJAX*[3]. Detta eftersom *Google Maps* använder sig av det internt, men även för på grund av att viss funktionalitet, exempelvis automatisk komplettering (auto completion), går att åstadkomma med hjälp av *AJAX*.

Nästa fundering låg kring vilket serverside-språk som skulle användas för själva HTML-uvecklingen. *PHP* och *ASP* var de främsta kandidaterna, men eftersom *PHP* var mer bekant för mig som utvecklare föll valet på det. *PHP* är ett mycket väl utvecklat språk som uppdateras kontinuerligt och dessutom finns det mycket information om *PHP* på nätet.

All HTML-kod som skrevs skulle följa *XHTML 1.0 Strict*-standarden, formateringen på koden skulle alltså vara strikt och helt korrekt ur *XML* synpunkt med slutna taggar och dylikt. Vid presentationen av sidorna skulle *CSS*[4] användas, eftersom *XHTML 1.0 Strict* inte inkluderar någon funktionalitet för presentation, utan räknar med att *CSS* används. Den största fördelen med *CSS* är att all formatering av sidan styrs av separata filer och är det något som behöver ändras görs det i *CSS*-filen, inte i varje HTML-fil.

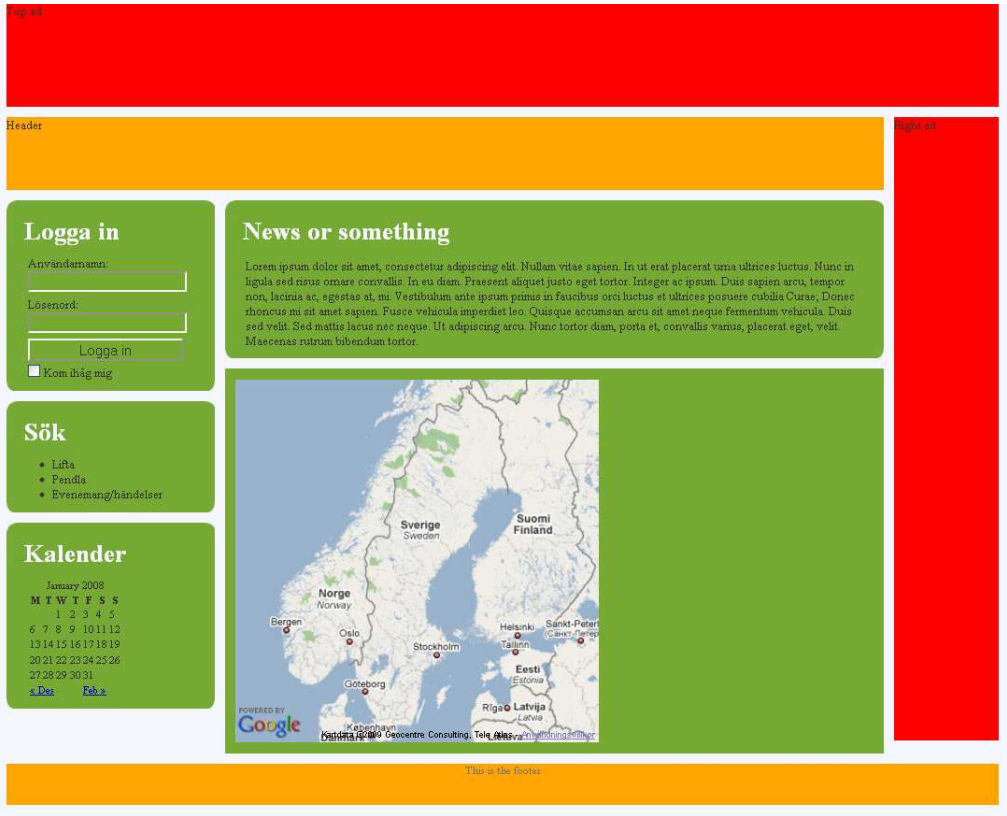
Då sajten skulle komma att använda användardefinierad information krävdes det en databas att lagra informationen i. *MySQL*[5] valdes för denna uppgift eftersom det är en välkänd databas som många webbhotell använder sig av. *GIMP*[6] användes som bildredigeringsprogram eftersom det är gratis och kraftfullt. Slutligen utnyttjades *WAMP*[7] som lokal webbserver under utvecklingens gång eftersom det är enkelt att installera den, och den använder de tekniker som skulle användas för i projektet (*MySQL* och *PHP*).

[1] `http://code.google.com/intl/sv-SE/apis/maps/`, 2009-05-31
[2] `http://en.wikipedia.org/wiki/Javascript`, 2009-05-31
[3] `http://en.wikipedia.org/wiki/Ajax_(programming)`, 2009-05-31
[4] `http://en.wikipedia.org/wiki/Css`, 2009-05-31
[5] `http://en.wikipedia.org/wiki/MySQL`, 2009-05-31
[6] `http://en.wikipedia.org/wiki/GIMP`, 2009-05-31
[7] `http://en.wikipedia.org/wiki/Wamp`, 2009-05-31

2.2.3 Design

Det stora målet med utseendet på sajten var att den skulle vara enkel att använda för användaren. Vissa andra sajter med liknande tema hade tryckt in så mycket information på varje sida (framförallt startsidan)att det blev svårt att få en överblick över vilka funktioner som fanns. Detta skulle till största möjliga grad undvikas på *HitchHike.se*, målsättningen var enkelhet och tydlighet.

Designarbetet började med att surfa runt på diverse sidor för att försöka identifiera enkla men ändå snygga och designupplägg att ta inspiration från. Samtidigt noterades det vad som fungerade bra och mindre bra, vilket resulterade i en första preliminär design på papper. Det skapades även en digital version av denna design som illustreras i figur 2.2. Därefter har många modifikationer utförts för att nå det slutgiltiga målet.



Figur 2.2: En av de tidigare design-versionerna av webbtjänstens index-sida.

Färgerna som används i den slutliga versionen av sajten (se alla figurer i kapitel 5 på sidan 31) har av flera anledningar. Det gröna ska få användarna att tänka på miljön medan det gröna och gråa tillsammans har en lugnande effekt. Färgerna på sajten passar även bra ihop med färgerna i *Google Maps*, vilket bland annat illustreras i figur 5.3 på sidan 35. Alla boxar på sidan är tonade eftersom det ger ett lugnare intryck, det blir annars lätt för intensiva färger.

Flera guider i användbarhet (usability) har använts för att försöka få designen

användarvänlig, bland annat har en guide från *Webmaster Tips* använts [13]. Tabell 2.1 på motstående sida visar alla riktlinjer från *Webmaster Tips* samt hur väl de uppfylls på *HitchHike.se*. 16 av 20 riktlinjer har uppfyllts.

2.3 Implementering

Implementationen av systemet utgick från planeringen och innefattade databaskonstruktion samt programmering av funktioner och designen av sajten. Relativt mycket tid spenderades på planering (cirka 2 veckor), vilket medförde att få stora förändringar genomfördes i implementationsfasen. Sajten byggdes upp ordentligt från grunden med ett objektorienterat tänk, vilket innebar att det efteråt blev lätt att underhålla den samt att göra uppdateringar. Implementationen är den del av arbetet där mest tid har spenderats, detta steg beskrivs därför i ett eget kapitel (kapitel 3 på sidan 15).

2.4 Utvärdering

Webbtjänsten har funktionstestats kontinuerligt under utvecklingens gång för att minimera antalet buggar innan de hinner sprida sig i systemet. I slutet av projektet utfördes det även ett större test som utgick från de funktionella kraven. Där kontrollerades det att alla funktionella och icke-funktionella krav uppfylldes och att alla funktioner fungerar korrekt. Kapitel 5 på sidan 31 beskriver de resultat som har framkommit av projektet och kapitel 6 på sidan 41 diskuterar resultaten och utvärderar systemet och arbetet.

2.5 Dokumentering

Den huvudsakliga dokumentationen har gjorts i denna rapport samt i källkoden. Arbetet har även dokumenterats med hjälp av *Google Docs* samt i en Blogg[8]. Dokumentationen i källkoden är inte tillgänglig för alla eftersom den är skyddad. Det som finns skrivet i *Google Docs* är inte heller allmänt tillgängligt eftersom det i stort sett bara är anteckningar och idéer som har diskuterats. Denna rapport innehåller alla viktiga delar från allt som har dokumenterats.

I dagsläget har ingen användarhandledning producerats, men den kommer att göras snarast och läggas upp på sidan så att användarna av sajten får nytta av den.

[8] `http://magnus-larsson.blogspot.com/`

Guideline	Efterlevs
1. **Design a clear and simple navigation system.**	**6/8**
1.1 Keep it consistent.	Ja
1.2 Use appropriate text inside links.	Ja
1.3 Use CSS to emphasize text links.	Ja
1.4 Always include text links.	Ja (enbart text)
1.5 Add a text-based site map.	Nej (skall göras)
1.6 Include a home page link inside your main navigation system.	Ja
1.7 Site logo links to home page.	Ja
1.8 Include a site search box.	Nej
2. **Keep the content clear and simple.**	**5/5**
2.1 Don't save the best for last.	Ja
2.2 Make page content easy to scan.	Ja
2.3 Avoid using text inside images. whenever possible	Ja
2.4 Add ALT and TITLE attributes to all images.	Ja
2.5 Contrast, contrast, contrast.	Ja
3. **Support your brand.**	**3/4**
3.1 Keep colors and typefaces consistent.	Ja
3.2 Keep page layout consistent.	Ja
3.3 Custom error page.	Nej
3.4 Create a good tagline and use it on every page.	Ja ("Ingen bil skall gå tom")
4. **Provide for visitor feedback.**	**2/3**
4.1 Keep feedback forms short and clearly note which information is required to successfully submit the form.	Ja
4.2 Remember your international users and don't require information they may not have - like area codes or ZIP codes.	Nej (sidan anpassas för Sverige)
4.3 Present complete contact information including your business phone number and postal address.	Ja (kommer att göra när sådan information finns)
5. **Test the site on real users.**	Ej genomfört ännu

Tabell 2.1: Tabell som visar ett antal riktlinjer om användbarhet[13] och om de uppfylls på sajten

Kapitel 3

Implementation

Detta kapitel förklarar på vilket sätt MVC-konceptet har använts i utvecklingen av webbtjänsten samt hur den underliggande strukturen i systemet ser ut. Informationen som ges i detta kapitel är dock begränsad på grund av säkerhetsskäl, eftersom rapporten är en offentlig handling och vem som helst kan läsa den. Vi vill alltså inte att vem som helst skall kunna ta del av exakt hur sajten är uppbyggd.

3.1 MVC - Model View Controller

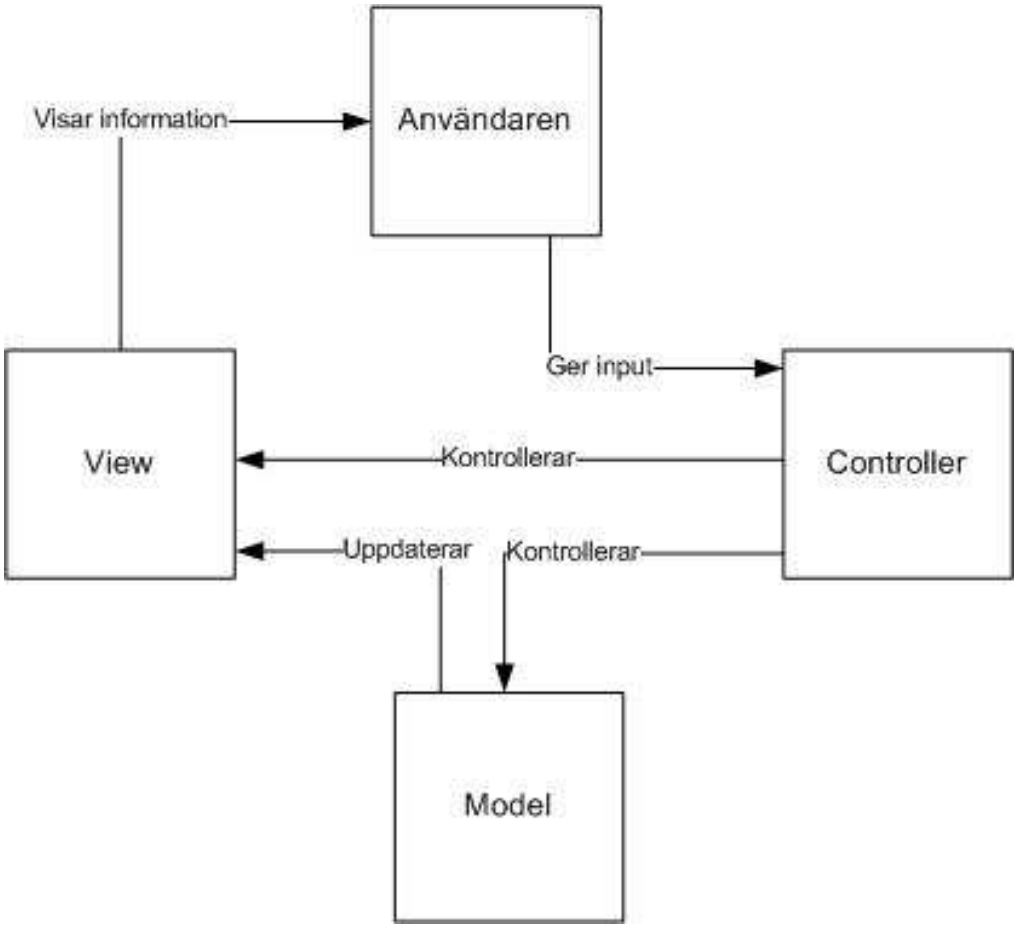
Model View Controller [9] (MVC) är ett designmönster som ofta används när större datasystem skall utvecklas, för att separera användargränssnittet från datastrukturerna. En schematisk skiss av MVC-modellen visas i figur 3.1 på följande sida. De tre delarna (fyra med användaren) har följande funktioner:

- Datastrukturer (Model) som representerar modeller och sköter kommunikation med databasen.
- Användargränssnitt (View) som ger användaren möjlighet att interagera med systemet.
- Kontrollenheten (Controller) som hanterar input från användaren, kontrollerar händelseflödet samt kontrollerar användargränssnittet och datastrukturerna.

Fördelarna med att dela upp systemet på detta vis är dels att det ofta blir enkelt att återanvända kod, dels att systemet blir enklare att underhålla och dels att hela systemet blir mer överskådligt och lättutvecklat. Den här uppdelningen gör även att det blir naturligt att använda objekt och klasser, åtminstone för datastruktur-delen.

3.1.1 Implementation av MVC

Till många av de mest utbredda programmeringsspråken existerar det färdiga implementationer av ramverk (frameworks) som följer MVC-modellen. Några exempel på sådana är *Ruby on Rails*, *PHPonTrax* och *FUSE*. I detta projekt har dock inget ramverk använts, i stället har en egen tolkning av MVC-modellen implementerats. Katalogstrukturen är uppbyggd efter MVC-modellen, det vill säga det finns speciella kataloger för modeller, användargränssnitt och kontrollenheter.



Figur 3.1: Förklarar hur MVC-modellen fungerar.

Användargränssnittet är uppdelat i ett antal olika kataloger eftersom det finns flera distinkta delar av gränssnittet. Exempelvis finns det en katalog för formulär-klasser, en för visningar av olika objekt samt en mapp för huvudsidorna[1].

Modellerna i systemet finns i en speciell katalog och består endast av klasser som representerar de olika objekten i systemet. Objekten byggs upp utifrån data som har lagrats i databasen och det är modellerna som sköter all databaskommunikation. Det betyder att modellerna är de enda filerna i systemet som är beroende av databasen och dess implementation, vilket gör det lätt att uppdatera systemet om någon del i databasen behöver förändras.

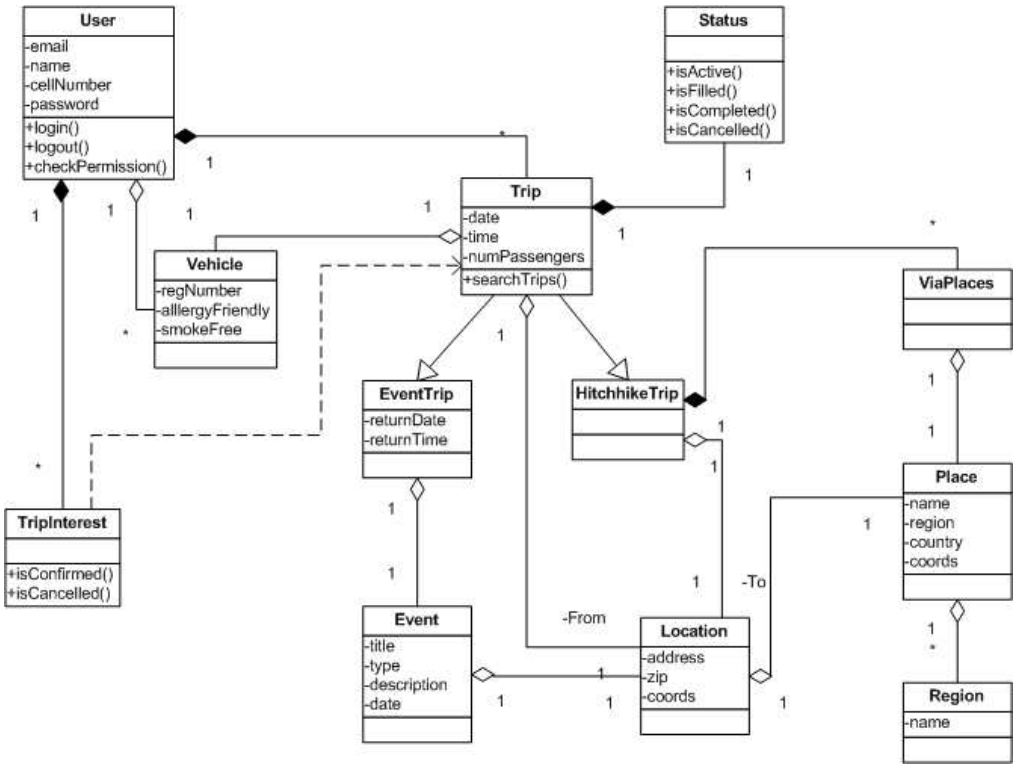
Kontrollenheterna finns även de samlade på ett enda ställe, och har till uppgift att parsa input från användaren, bestämma vad som skall göras på sidan och göra information tillgänglig för visning. Till exempel, om en resa skall skapas är det en kontrollenhets uppgift att kontrollera informationen som användaren ger via formulär, sedan skapa en modell av resan och slutligen säga åt modellen att spara informationen

[1] Sidorna som användaren ser, exempelvis index.php

till databasen.

3.2 Systembeskrivning

Systemet består i grunden av en databas vars uppgift är att lagra all information användarna matar in. Som det nämndes i föregående sektion är det utifrån databasen alla modeller byggs upp. Den fullständiga databasstrukturen kommer ej att visas i denna rapport på grund av säkerhetsmässiga skäl. Dock visas ett förenklat klassdiagram[2] över systemet i figur 3.2. Detta klassdiagram representerar alla modeller i systemet och illustrerar hur allt hänger ihop. Klassdiagrammet är ej fullständigt, bara de viktigaste attributen och metoderna visas.



Figur 3.2: Förenklat klassdiagram över hela systemet.

Alla klasser i systemet har get- och set-metoder för variablerna. Det finns också flera andra metoder i varje klass som har olika uppgifter. Ett urval av dessa metoder samt sambanden mellan klasserna diskuteras i följande sektioner. Alla resonemang utgår från klassdiagrammet.

[2]Observera att alla klass- och attributnamn inte stämmer överens exakt med verkligheten på grund av säkerhetsmässiga skäl

3.2.1 Användare och fordon

När en användare av systemet registrerar sig på sajten skapas ett objekt av typen *User*. Klassen ansvarar för att logga in och logga ut användaren samt för att autentisera användaren när speciella rättigheter krävs. Användarklassen innehåller all information om användaren som namn, telefonnummer, mailadress och lösenord. Lösenordet sparas ej i klartext utan lagras krypterat ifall obehöriga skulle komma åt innehållet i databasen.

En användare kan förfoga över ett antal fordon i systemet. Dessa fordon kan användaren välja mellan när hon skapar en resa där hon erbjuder samåkning. Det är användaren själv som ansvarar för att lägga till fordon som hon använder för samåkning. Ett fordon representeras av klassen *Vehicle* och har till uppgift att hålla reda på information om fordonet, till exempel registreringsnummer, om fordonet är rök- och allergifritt med mera. Samma fordon kan finnas registrerat flera gånger men för olika användare, exempelvis om ett par har en bil som båda använder kan båda två lägga till fordonet i sin användare på sidan.

3.2.2 Geografiska positioner

Eftersom systemet bygger på att resa innefattar stora delar av systemet att hålla reda på geografiska positioner. Detta görs med klasserna *Location* (ställe), *Place* (ort) och *Region* (län). All geografisk data som lagras i databasen hämtas från *Google Maps*, via adressökning. Det innebär att data som lagras i databasen alltid är korrekt, det vill säga att alla lagrade adresser och positioner verkligen existerar och är rätt stavade[3], vilket underlättar när frågor skall utföras på databasen.

Region är motsvarigheten till Sveriges län. En region är kopplad till ett län och ett land. Landet är lagrat för framtida bruk ifall webbtjänsten senare utökas internationellt. Regionindelning är till för att kunna göra smidigare sökningar på större områden. Till exempel är det möjligt att söka resor som går till Västerbotten, i stället för att göra sökningar på flera olika städer inom Västerbotten.

En ort i Sverige representeras av klassen *Place*. Med ort avses allt från stora städer till små byar. Det finns inga dubbletter av orter i systemet vilket gör det möjligt att effektivt filtrera resor per ort om så önskas. En ort har ett namn, land, region samt longitude och latitude. En ort kan unikt identifieras på dess namn eller dess interna id-nummer. Orterna i systemet läggs inte in manuellt i förväg, dessa läggs till dynamiskt när användare skapar resor. På så sätt finns alla relevanta orter med i databasen och det mödosamma jobbet att att lagra alla orter manuellt uteblir.

Location används för specifika ställen där resor startar från eller går till. Ett ställe har en specifik adress och måste alltid vara ihopkopplat med en ort. Ett ställe behöver inte vara precist specificerat, det kan vara allt ifrån bara en ort, till ett postnummer till en exakt adress. Eftersom Google Maps används när adresser och geografiska positioner matchas ihop är det möjligt att se hur noggrann en given adress är med hjälp av givna konstanter. Den stora skillnaden mellan en ort och ett ställe är att en ort alltid bara pekar på en ort, medan ett ställe pekar på en precis position och håller reda på hur noggrann positionen är given. Ett ställe är även kopplat till en specifik användare medan en ort inte är det.

[3]Det är korrekt så länge *Google Maps* har rätt

3.2.3 Resor

De lagrade resorna i databasen utgör en av de viktigaste grundstenarna i systemet. Ett reseobjekt (*Trip*) representerar och håller reda på all information om en resa. I systemet finns det två olika typer av resor, samåkningsresor(*HitchhikeTrip*) och evenemangsresor (*EventTrip*). Samåkningsresor är resor som sker på oregelbundna tider och till i princip vart som helst inom Sverige, exempelvis när någon skall resa hem till sina föräldrar. Evenemangsresor är den typ av resor som sker till evenemang i Sverige. Evenemanget kan vara av vilken typ som helst, sporthändelser, konserter med mera. Tanken är att det skall finnas flera resor från olika orter som går till samma evenemang.

De två typerna av resor har vissa attribut gemensamt, vart resan startar, tid och datum för avfärd, vilket fordon som används för resan om samåkning erbjuds, antal passagerare samt om användaren är flexibel i avresetid. Specifikt för samåkningsresor är att de håller reda på vart resan slutar. Evenemangsresor håller i stället reda på vilket evenemang (*Event*) resan går till samt vilken tid och datum hemresan sker. Ett evenemang består av en titel, startdatum och tid, slutdatum och tid, vart evenemanget äger rum samt vilken typ av evenemang det är.

Ytterligare en klass som har med resorna att göra är *TripInterest*. Den klassen ansvarar för att hålla reda på att en användare har anmält sitt intresse på en given resa. Den vet när användaren anmälde sitt intresse, om och när annonsören godkände anmälan samt om och när någon av de två parterna har avanmält intresset att samåka.

3.3 Google Maps

Google Maps har använts flitigt i utvecklingen, dels för att visa positioner på en karta, dels för att hitta ställen utifrån sökningar på adresser, och dels för att beräkna färdvägar utifrån ett antal givna orter som resan måste passera. Google Maps har ett väl utvecklat API[4] som fungerar med JavaScript. De klasser som framförallt har använts är:

- GMap2 - klassen som visar kartan, finns på varje sida som Google Maps används.
- GClientGeocoder - hittar positioner på kartan utifrån en given adress.
- GDirections - används för att hitta färdväg mellan orter.

Förutom dessa tre huvudklasser har några mindre hjälpklasser använts i arbetet, bland annat *GLatLng* (geografiska koordinater), *GInfoWindow* (informationsfönster som visas), *GMarker* (markera objekt på kartan) och *GPolyline* (dra räta linjer mellan punkter på kartan).

Som ett komplement till APIt har framförallt sidan `http://maps.forum.nu/` använts för inspiration och tekniska lösningar. Den sidan har många bra exempel på hur *Google Maps* kan användas och hur tekniska problem kan lösas. Vidare har *Googles* verktyg *AJAX APIs Playground*[5] använts för att experimentera med vissa av kartfunktionerna.

[4] `http://code.google.com/intl/sv-SE/apis/maps/documentation/reference.html`
[5] `http://code.google.com/apis/ajax/playground/`

3.4 HTML

Webbsajten är implementerad med *XHTML 1.0 Strict*[6] och varje sida är validerad via `http://validator.w3.org/`, detta för att åstadkomma bästa kompatibilitet med olika webbläsare. *XHTML 1.0 Strict* är i princip en uppstramning samt förenkling av tidigare *HTML*-standarder och använder sig av strikt *XML* för att strukturera sidorna. Standarden kräver sedan att CSS (sektion 3.6) nyttjas för design och layout. *HTML*-koden följer till största möjliga mån de riktlinjer som finns vid design med *XHTML 1.0 Strict*, dock kommer detta ej att diskuteras utförligare i denna rapport.

3.5 Javascript

Ett antal *JavaScript*-filer har skapats under arbetets gång. Dessa har dels i uppgift att styra och visa kartorna, dels att göra sidan mer användarvänlig genom att med hjälp av *AJAX* göra sidorna mer dynamiska och levande.

JavaScript-filerna har utvecklats med grundtanken att de skall vara körbara på så många webbläsare som möjligt. *JavaScript* är dock väldigt svårt i detta avseende men allting har testats i de aktuella versionerna av de två största webbläsarna, *Firefox* och *Internet Explorer*. Document Object Model[7] (DOM) har till stor del använts eftersom det åtminstone förbättrar möjligheterna att det skall fungera i alla webbläsare.

Det brukar vara rekommenderat att en webbtjänst skall fungera även om användaren har avaktiverat *JavaScript* i sin webbläsare[8]. Detta efterlevs dock ej för detta system eftersom *Google Maps* används, vilket kräver att *JavaScript* är aktiverat. Några av de funktioner som har implementerats i *JavaScript* för detta system är:

- Göra sökningar på adresser med hjälp av *Google Maps*.
- Möjliggöra automatisk komplettering (auto completion) av ortnamn vid sökningar med hjälp av *AJAX*.
- Kontrollera beteenden och funktioner hos vissa knappar och tabeller.
- Kontrollera utseendet av kartor samt lägga till speciella kontroller på kartorna.

3.6 CSS

Stilmallar i form av Cascading Style Sheets[9] (CSS) har genomgående nyttjas för designen av hela sajten. CSS möjliggör att på ett enkelt sätt kunna ändra design och färger på sidan genom att ändra i några få filer. CSS ökar alltså separationen av delarna i systemet ytterligare.

CSS-filerna har delats upp i tre olika användningsområden för att det skall vara lättare att hitta det som skall redigeras. *Layout* styr hur elementen på sidan skall placers, *style* styr det grundläggande utseendet på sidan i form av färger, fonter och så vidare och *form* styr layout av fält och etiketter på formulär.

[6] `http://www.w3.org/TR/xhtml1/`
[7] `http://en.wikipedia.org/wiki/Document_Object_Model`, 2009-05-31
[8] Enligt bland andra
`http://www.klientsidan.se/artiklar/20080501/Javascript_och_AJAX_pa_ratt_satt/`
[9] `http://en.wikipedia.org/wiki/Css`

Kapitel 4

Webbsäkerhet

Detta kapitel behandlar det, för examensarbetet, obligatoriska fördjupningsarbetet. Fördjupningsarbetet tar upp området webbsäkerhet, dels ur den utvecklade webbtjänstens perspektiv, dels ur ett större mer generellt perspektiv. De mest förekommande säkerhetsbristerna samt specifika brister för den utvecklade sajten identifieras och förslag ges på hur dessa brister kan åtgärdas utifrån publicerade lösningar.

Kapitlet börjar med en inledning om webbsäkerhet och identifierar vilka speciella säkerhetsmål som finns med *HitchHike.se*. Sedan följer ett antal sektioner med diskussioner kring olika vanligt förekommande säkerhetshål och hur dessa kan avhjälpas. Kapitlet avslutas med en diskussion kring webbsäkerhet.

4.1 Bakgrund till problemet

Ett stort problemen med dagens webbsajter är att de blir mer och mer utsatta för säkerhetsattacker. Detta beror till stor del av att sidorna blir mer användarstyrda, vilket medför att det finns fler ingångar till sidorna. Detta i sin tur innebär att personer med dåliga avsikter också får ökade möjligheter att på olovlig väg ta sig in på sidan och orsaka skada. Som utvecklare måste man vara väl medveten om vilka säkerhetsrisker som finns med alla tekniker man använder samt vilka metoder som finns för att motverka attacker.

Beroende på webbsajtens syfte och vilken information den tillhandahåller är risken för attacker samt vilken skada en attack kan åstadkomma olika stor. Det går att definiera sidor som högrisksidor eller lågrisksidor, med en bred skala däremellan. En högrisksida är till exempel en sida som hanterar pengar, känsliga personuppgifter, företagsinformation med mera. Det kan få ödestigna konsekvenser om obehöriga personer får tillträde till en sådan sida, därför måste kraftiga åtgärder vidtas för att upprätthålla säkerheten. Lågrisksidor är motsatsen till högrisksidor, de hanterar mindre viktig information och det är ingen katastrof om obehöriga på något sätt skulle lyckas få tag i information från databasen eller på annat sätt sabotera sidan. Det är alltid viktigt att vidta åtgärder för att göra systemen så säkra som möjligt, men säkerhetsarbetet måste stå i proportion till sidans innehåll och sajt-ägarnas vilja att skydda sig.

4.1.1 Säkerhetskrav på HitchHike.se

HitchHike.se kan i dagsläget anses vara lågrisksida. Inga pengar hanteras och endast begränsade personuppgifter lagras i databasen. Detta innebär dock inte att säkerheten

helt skall ignoreras. Målet med sajten är att den skall utvecklas och det finns vissa framtidsplaner om betaltjänster med mera vilket bör tas med i beaktande. Om sajten växer sig stor kommer det förmodligen att finnas mer känslig information eftersom den utvecklas, det är då bra om vissa säkerhetsbrister har blivit åtgärdade. Nedan listas en rad säkerhetskrav som har identifierats för *HitchHike.se*. Detta är inte en uttömmande säkerhetsanalys av systemet, bara de mest tydliga säkerhetskraven behandlas.

1. Ingen obehörig skall kunna ta sig in på någon annans konto.
2. Lösenord skall vara krypterade i databasen. Detta för att det ej skall vara möjligt att se lösenorden i klartext om någon olovligen får tillgång till databasen.
3. Det skall vara så svårt som möjligt för obehöriga att komma åt information från databasen, med tanke på att mailadresser och personuppgifter finns lagrade där.
4. Det skall inte vara möjligt för någon obehörig att ta bort eller redigera information på sajten.
5. Det skall inte vara möjligt att stjäla information om en användare, i form av cookies och så vidare, när användaren är online.
6. Sajten skall vara skyddad mot överbelastnings-attacker (DOS-attacker) och brute-force attacker.

4.2 Säkerhetshot

Det finns ett stort antal olika säkerhetshot mot webbsajter. Denna sektion tar upp ett antal av de mest relevanta hoten baserat på hur vanligt förekommande de är och hur stor risk det är att de används för att attackera *HitchHike.se*, utifrån riskerna som nämndes i sektion 4.1.1. För varje hot förklaras det vad det åstadkommer, vilka effekter det får för sidan/användarna samt hur det är möjligt att minimera riskerna att bli drabbad av det specifika hotet. I slutet av denna sektion nämns även en mängd andra säkerhetshot som inte får lika stort fokus i denna rapport, men ändå har tillräckligt stor betydelse för att nämnas.

4.2.1 SQL Injektioner

SQL-injektioner (SQL injections) är en form av attack som sker vid tillfällen där användarna skickar data via input-fält som på något sätt används för att skapa en SQL-fråga. Genom att skriva in smart konstruerade SQL-kommandon i input-fälten, i stället för den data som efterfrågas, kan en ondsint användare orsaka skada på olika sätt. Det kan bland annat vara möjligt att logga in som en annan användare utan det korrekta lösenordet, eller få tillgång till tabell-strukturer med mera genom att generera felmeddelanden. Det kan till och med vara möjligt att förstöra delar av databasen. Dessa brister kan därmed även innebära att data från databasen kan hamna i fel händer, vilket är mycket dåligt ifall det finns känslig data lagrad [5, 1].

Enligt Halfond et. al[4] är det inte bara via input-fält som SQL-injektioner kan utföras. All data som används för att skapa SQL-frågor kan potentiellt utnyttjas. Exempelvis kan data komma från cookies eller från server-variabler. Om en ondsint användare lyckas ändra värden på variabler i exempelvis cookies kan även detta

användas för att skapa SQL-injektioner, vilket är något som många utvecklare inte uppmärksammar.

Några åtgärder som är vanliga att utföra vid arbete med databaser och är en del av lösningen mot SQL-injektioner är att nyttja så kallad defensiv programmering och input-validering [5, 4]. Några exempel på sådana metoder är följande:

- **Escapa strängar**. Genom att sätta in ett escape-tecken, "\", före vissa speciella tecken i strängar, bland annat före apostrofer och citationstecken, går det att undvika många fallgropar med inputsträngar. Tecken som tidigare tolkades som en del av SQL-syntaxen tolkas i stället som ett vanligt tecken i en sträng och kan inte längre orsaka skada i databasen.

- **Lösenordskryptering**. Att kryptera lösenord före de sparas i databasen medför dels att användarnas lösenord är skyddade, de syns aldrig i klartext. Dels medför det att när användaren skriver in sitt lösenord i ett fält måste det krypteras för att få samma form som det lagrade lösenordet, och eftersom hela strängen krypteras innebär det att eventuella "farliga" tecken inte kommer göra någon åverkan på databasen.

- **Kontrollera antal träffar**. Många gånger före en SQL-fråga skickas till databasen är det känt hur många träffar frågan skall ge. Exempelvis när en användare skall logga in skall det bara bli en träff på mailadress och lösenord. Då bör det kontrolleras att antalet träffar blev precis en, varken mer eller mindre. Detta förhindrar att ondsinta användare kan komma använda en känd mailadress och göra en SQL-injektion så att lösenordet ignoreras i frågan.

- **Undvik felutskrifter**. Tillgång till information om hur databasen är uppbyggd, dess struktur, kan vara en bra källa för att kunna åstadkomma skada i databasen. Därför är det viktigt att inte skriva ut felutskrifter som blir om frågan till databasen är inkorrekt. Om utskrifter tillåts är det möjligt, genom att sätta in olika SQL-kommandon som genererar fel i SQL-frågan, att få tillgång till detaljerade information om databasen. Information som i fel händer kan vara dåligt ur säkerhetssynpunkt.

- **Kontrollera input-strängar**. Genom att kontrollera formatet på input-strängar undviks många fallgropar angående SQL-injektioner. Ofta är det känt precis hur input-strängen skall se ut och det går därför med hjälp av reguljära uttryck att kontrollera en sträng så att den uppfyller de krav som finns på strängen. Om det finns "konstiga" tecken i strängen går valideringen ej genom.

Förutom ovan föreslagna åtgärder finns det mer avancerade förslag på hur SQL-injektioner kan undvikas. W. Halfond et. al har två förslag på lösningar, *AMNESIA* [5] och *WASP* [4]. *AMNESIA* använder ett modell-baserat angreppssätt med en statisk och en dynamisk del. Den statiska delen bygger modeller av tillåtna frågor medan den dynamiska delen jämför den av användaren dynamiskt genererade frågan mot de statiska modellerna för att se så att den givna frågan är giltig.

Halfonds andra lösning med *WASP* använder teknikerna *Positive Tainting* och *Syntax-Aware Evaluation*, fritt översatt positiv färgning och syntax-medveten utvärdering. Halfond själv förklarar tekniken på följande sätt:

> Our approach consists of 1) identifying trusted data sources and marking data coming from these sources as trusted, 2) using dynamic tainting to

> track trusted data at runtime, and 3) allowing only trusted data to form the semantically relevant parts of queries such as SQL keywords and operators

Vidare har Buehrer et. al gjort en lösning med *Parse Tree Validation* [1]. Den liknar Halfonds första lösning då den bygger upp ett parse-träd av den givna SQL-frågan och kontrollerar om den är korrekt, jämfört med ett giltigt parse-träd.

4.2.2 Spoofing och Phising

Spoofing innebär att en ondsint användare utger sig för att vara/representera en känd organisation/företag i syfte att få den vanliga användarens förtroende för att sedan kunna utnyttja detta på diverse sätt. Med andra ord är det en typ av identitetsstöld. En vanlig metod för att åstadkomma detta är genom att skicka förfalskade email som innehåller länkar till en förfalskad hemsida som ser ut precis som den "riktiga" hemsidan.

Spoofing utnyttjas, bland annat, för *Phising*, det vill säga för att lura användaren att ge ut hemlig information (lösenord, användarnamn, pin-koder, kontonummer med mera) i webbformulär. Denna information används sedan för att komma åt hemlig information eller att komma över pengar på något sätt [10, 6, 3].

Spoofing-attacker har utförts mot väldigt många kända organistationer. Paypal, Ebay och Citibank är några av offren som Dinev [3] nämner. Dinev hävdar också att det är mycket svårt att skydda sig mot *spoofing*-attacker eftersom de personer som använder sig av *spoofing* hela tiden förbättrar sina tekniker när de äldre metoderna har kunnat stoppas. Enligt Deniv är det bästa sättet för användaren att skydda sig mot dessa attacker genom att vara medvetna om att *spoofing* existerar och vad *spoofing* gör.

Tabell 4.1 på motstående sida är härledd från Denivs artikel och beskriver några egenskaper för mail och hemsidor som har blivit spoofade. Tabellen skall som ses en guide på hur det är möjligt att identifiera *spoofing*- och *phising*-attacker, den är dock inte helt uttömmande.

Förutom att ta ett personligt ansvar för att undvika *spoofing*- och *phising*-attacker, finns det tekniska lösningar som till viss del hanterar dessa problem. Herzberg och Jamara utvecklar i deras artikel [6] en lösning, kallad *TrustBar*. Deras lösning skall hjälpa användaren att upptäcka när *spoofing* eller *phising* sker genom att

> ...enhance browsers with mandatory, improved security and identification indicators. In our implementation, the indicator presents (by default) identification from the certificate, and allows user-customization.

4.2.3 DOS-attacker

En Denial of Service-attack (DOS-attack) har som syfte att göra en tjänst otillgänglig för de vanliga användarna. En DOS-attack kan ske på två sätt. Det första sättet är genom att skicka ett paket som är specialgjort för att skada systemet genom att utnyttja ett känt säkerhetshål. Det andra sättet är genom att skicka ett mycket stort antal paket som orsakar överbelastning och får systemet att krascha. Det finns även DDOS-attacker, Distributed Denial Of Service-attack. Dessa attacker är om möjligt ännu svårare att upptäcka och förhindra. En DDOS-attack går till så att en stor mängd datorer genomför DOS-attacker mot samma mål samtidigt. Datorerna i attacken behöver nödvändigtvis inte vara medvetna om attacken, det kan vara så att de olovligt har blivit injicerade med ett litet program som antingen på kommando eller på ett visst klockslag utför attacken [12].

Beskrivning	Avslöjande faktorer
Övertygande email. Åstadkoms genom att göra så att det ser ut som att mailet kommer från en giltig avsändare (ex. service@paypal.com) och har ett bra ämne. Innehållet i mailet är sedan kodat med HTML och *JavaScript* vilket gör det möjligt att länken till sidan ser korrekt ut, men i verkligheten pekar till en förfalskad sida.	Om hälsningsfrasen i mailet är opersonlig, exempelvis inte innehåller ens riktiga namn utan bara "Dear PayPal user" eller liknande.
Övertygande logos och layout. Allt ser äkta ut på sidan och länkar fungerar.	**Existerar endast en kort tid**. Sidan finns tillräckligt länge för att hinna få information från ett antal användare, och skall helst försvinna spårlöst efter det.
Input från användaren. Sidan vill att användaren skall ge personlig information om sig själv på grund av diverse anledningar. Ibland kan det även vara frågan om att göra en ny registrering och ange all information om sig själv.	Kan inte hantera felaktiga login-försök, alla försök godkänns.
Övertygande HTML. Genom att kopiera innehållet på den "riktiga" sidan kan den spoofade sidan se precis ut som originalet. Sedan stoppas exekverbart innehåll in i sidorna, via bland annat *JavaScript*. *JavaScript* används även för att kontrollera länkar på sidan.	När användaren skickar in formulär eller klickar på länkar syns aldrig den riktiga URLen, eftersom länkar, bilder och händelser vid forumlär styrs av *JavaScript*.
Övertygande länkar. Det är möjligt att ersätta interface-komponenter i webbläsaren, vilket medför att det är möjligt att sätta dit ett nytt adressfält och indikera att SSL används med mera.	– Den verkliga adressen till sidan visas om användaren sparar sidan som en favorit. – Genom att kotrollera *cookies* för sidan visas den riktiga adressen till sidan.

Tabell 4.1: Beskriver egenskaper hos mail och hemsidor som blivit utsatta för *spoofing* och *phising*. Tabellen är härledd från [3]

Om en service (webbsajt) blir otillgänglig (Denial of Service) behöver det nödvändigtvis inte vara en attack av något slag. Till exempel, om onormalt många människor går in på en sida samtidigt och på så sätt gör att servern inte klarar av belastningen, kan problem uppstå och det kan se ut att vara en attack. Detta fenomen kan undvikas genom att dimensionera servrarna på rätt sätt, eventuellt med dynamiska resurser. Även nätverket kan krascha utan uppsåt, detta innebär också att tjänsten kan bli otillgänglig utan att någon medvetet försöker sabotera sidan.

Enligt Kargl et. al [7] finns det en mängd delar som kan vara mål för en DOS-attack. Dessa delar är hårdvaran (servrar, routrar med mera), operativsystemet som hårdvaran kör på samt TCP-stacken. Det finns ett flertal tekniker som har utvecklats för att försöka stoppa attacker mot dessa områden, bland annat Peng et. al [12] och Kargl et. al [7] har utvecklat några förslag.

Peng föreslår en fyra stegs metod mot DOS-attacker. Han talar om *prevention* - att förhindra attackerna redan före de göra någon skada. *Detection* - att upptäcka attacken. *Attack Source Identification* - att identifiera vem det är som utför attacken. *Reaction* - vilka åtgärder som skall vidtas vid en attack. Detaljer om Pengs metoder finns att läsa i [12].

Kargls metod [7] går ut på att använda *class based routing* och *active traffic monitoring* och med hjälp av dessa verktyg begränsa bandbredden för attackerande värdar (eng. hosts). Trafiken flyter då på som vanligt för de vanliga användarna, medan de som attackerar får kraftigt begränsad badbredd eller blir helt avstängda. Kargl nämner i artikeln att denna metod är bra för att motverka bandbredds-attacker, det vill säga attacker som går ut på att överbelasta systemet. Det innebär alltså att denna metod ej fungerar mot smartare attacker där specialgjorda paket skickas för att sabbotera, som det nämndes i första stycket på denna sektion.

En åtgärd som kan vidtas från implementationssidan av en webbsajt att införa begränsningar på sökningar [12]. Genom att införa exempelvis att samma IP-nummer endast får göra ett visst antal sökningar per minut minskas risken att någon skall utnyttja det som ett sätt att sabotera sidan. Begränsningar liknande detta kan genomföras för alla funktioner som är beräkningskrävande eller som gör stora sökningar i databasen. Detta eftersom webbservern snabbt kan bli överbelastad om en DOS-attack genomförs genom att exempelvis göra ett mycket stort antal sökningar under en kort tidsperiod.

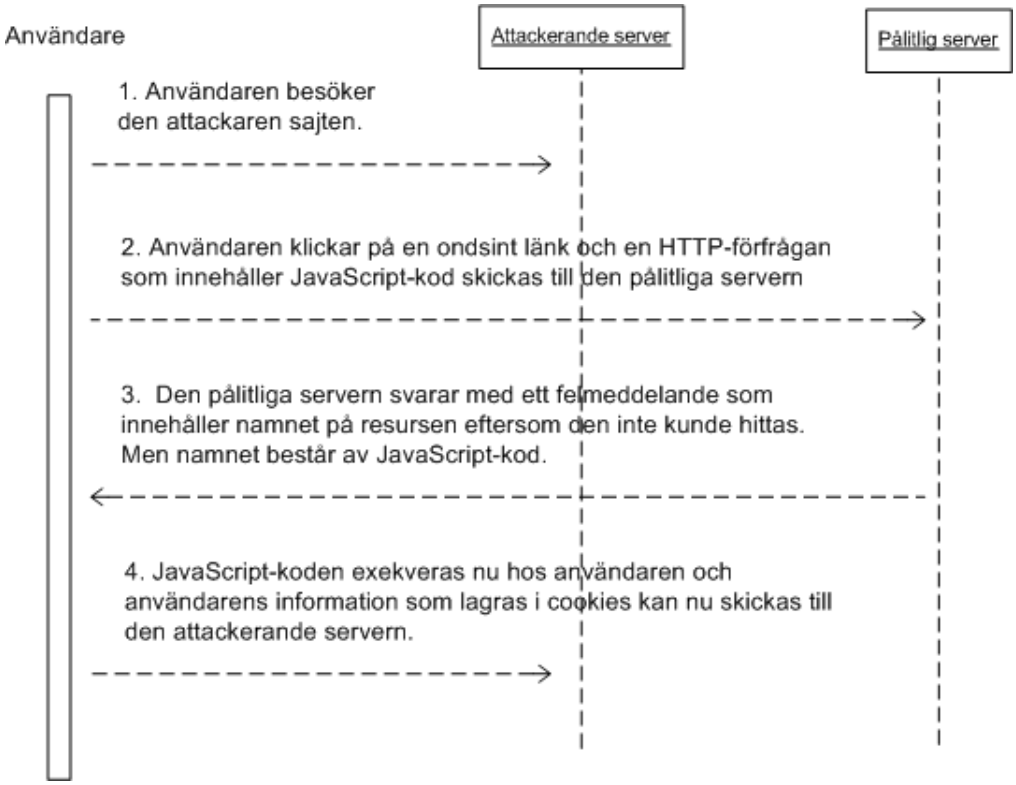
4.2.4 Cross site scripting (XSS)

Cross-site scripting (XSS) är en säkerhetsattack där den attackerande parten kör sin egen *JavaScript*-kod på någon annan sajt. En XSS-attack ger den ondsinta användaren möjlighet att modifiera sidan eller att stjäla information från användarens cookies.

JavaScript har inbygda säkerhetsåtgärder för detta problem, så kallad *sandboxing* och *same origin policy*. Dessa två metoder skall se till att kod som exekveras från en viss sida verkligen kommer från den sidan. De båda metoderna skapar en säker zon där endast kod från samma ursprungssida får exekveras, men det går dock att lura dessa mekanismer. Dels går det, om programmeraren inte har tagit hand om problemet, att infoga *JavaScript*-kod via formulär. Dels går det att konstruera länkar på den attackerande partens webbsida med skadlig *JavaScript*-kod inbäddad, och när användaren klickar på dessa länkar kan en attack ske mot en annan sajt [8, 11].

Figur 4.1 på nästa sida (härledd från figur 1 i [8]) visar hur en vanligt förekommande XSS-attack går till. Det börjar med att användaren klickar på en länk på den

attackerande partens sida. Då skickas en förfrågan till servern som skall attackeras, förfrågan innehåller *JavaScript*-kod. Den pålitliga servern tar emot förfrågan och behandlar den, men genererar fel eftersom förfrågan som har skickats ej är giltig. Då returnerar den pålitliga servern ett felmeddelande, som innehåller namnet på resursen som genererade felet, och då även *JavaScript*-koden. När användaren tar emot denna kod är det nu som att det är den pålitliga servern som har skickat *JavaScript*-koden, och därför får koden exekveras tillsammans med den pålitliga sidans övriga kod. *Sandboxing*-regeln är därmed bruten.



Figur 4.1: Visar flödet hos en XSS-attack. Figur härledd från [8]

Oda et. al [11] har utvecklat *SOMA - Same Origin Mutual Approval*, vilket är en teknik som skall kontrollera flöden av information och förhindra säkerhetsbrister som möjliggör bland annat XSS. Deras metod går ut på att förstärka policyn att all data skall ha samma ursprung, *same origin policy*. Detta åstadkoms i två steg. Det första steget är att programmeraren skapar en fil som namnger de sidor som är tillåtna. I nästa steg skall webbläsaren, via ett plugin-program, kontrollera att ursprunget på inkluderade filer matchar någon av de tillåtna sidorna.

Ett annat förslag till lösning på problemet har utvecklats av Kirda et. al [8]. Deras lösning bygger på en typ av proxy/brandvägg som kontrollerar varje anslutning som görs mot sidor. Användaren måste sedan godkänna de anslutningsförsök som görs mot sidor som inte kommer från huvud-sidan. För att förbättra denna lösning och göra det mindre besvärligt för användaren har de även implementerat en ananlysdel i programmet som

kontrollerar vilka länkar som är statiska, det vill säga de länkar som servern själv infogar på sidan. Dessa statiska länkar godkänns automatiskt, vilket medför att användaren endast behöver godkänna de dynamiskt skapade länkarna, vilka är de som kan vara skadliga.

4.2.5 Övriga hot

De ovan nämnda säkerhetshoten är bara en del av alla de säkerhetshot som existerar. Denna sektion beskriver kortfattat några av de övriga säkerhetshoten utifrån ett dokument som *Web Application Security Consortium* har skrivit [2].

En *Buffer Overflow*-attack utförs i syfte att antingen ändra funktionaliteten hos ett program eller att göra så att det kraschar. Detta åstadkoms genom att skriva data som överskrider den mängd minne som är allokerat. Databufferten blir därmed överfull (overflow) och kan därför börja påverka andra delar av programmet. Dessa attacker är dock vanligast mot program skrivna C och C++. Vanligaste fallet för *Buffer Overflow*-attacker för webbsidor blir därmed om exempelvis ett PHP-script använder sig av ett C/C++ program i bakgrunden.

Informationsstöld och *läckage av känslig information* är säkerhetshål som kan utnyttjas av ondsinta användare. Automatiskt katalogindexering kan exempelvis ge information om hur sidan är uppbyggd. Ett annat exempel på läckage av information är om det finns kvar kommentarer och felutskrifter i koden som visas i källkoden på sidan. Dessa säkerhetsbrister kan vara svåra att använda för direkta attacker, men ju mer information en ondsint användare har om sajten desto lättare blir det att planera och genomföra attacker.

Med en *brute force*-attack försöker den attackerande parten att gissa sig fram till användarnamn och lösenord, krypteringsnycklar med mera för att kunna få olovlig tillgång till information. Detta kan motverkas genom att bara tillåta ett begränsat antal försök för exempelvis inloggning.

Om sajten lider av *Insufficient Anti-automation* har den en säkerhetsbrist som innebär att det är möjligt att automatisera vissa processer på sajten, som egentligen skall utföras manuellt. Exempel på processer som ej skall gå att utföra automatiskt är att registrera nya användare, eller att posta nya meddelande på ett forum. Dessa brister kan leda till sämre prestanda i systemet eller i värsta fall att systemet kraschar.

4.3 Webbsäkerhet - diskussion

Den här fördjupningsstudien har visat några av, enligt mig, de viktigaste säkerhetshoten som finns vid webbutveckling. Även lösningar på problemen har diskuterats till viss del. Dels relativt enkla lösningar som programmeraren kan påverka, dels mer avancerade lösningar som involverar servrar och TCP-stacken. Området webbsäkerhet är ett
gigantiskt område och det är omöjligt att ta upp alla säkerhetshot som kan upptså, ännu svårare är att säga precis hur alla problemen skall lösas. Den här fördjupningsstudien skall alltså inte ses som någon fullständig guide på hur man undviker alla säkherhetshot.

Övergripande med alla metoder för att täppa till säkerhetshål måste man se till kostnader och resursåtgång. Är det värt att implementera eller går det att ta smällen vid en attack i stället.

Sett till HithcHike.se som är en lågrisksajt, vilket nämndes tidigare, blir det för mycket jobb att implementera alla de föreslagna åtgärderna. De åtgärder som kan

genomföras, utan att blanda in speciallösningar från diverse artiklar, är till stor del implementerade för att göra systemet så säkert som möjlig. I framtiden, om webbsajten utvecklas och blir större, måste en ny säkerhetsanalys genomföras för att se om det fortfarande finns några stora säkerhetsbrister och vilka nya åtgärder som måste tas.

Den avslutande delsektionen i detta kapitel tar upp de säkerhetskrav som nämndes i sektion 4.1.1 på sidan 22 och diskuterar om dessa krav kan uppfyllas med metoderna som har diskuterats i denna artikel.

4.3.1 HitchHike.se

Varje säkerhetskrav citeras och därefter diskuteras vilka säkerhetshot som kan påverka kravet, samt vilka metoder som kan användas för att lösa problemet. Inga av de mer avancerade metoderna som olika författare själva har utvecklat har använts eftersom det skulle ta för mycket tid att implementera och testa.

> Ingen obehörig skall kunna ta sig in på någon annans konto.

Det är framförallt SQL-injeketioner som kan orsaka denna typen av säkerhetsluckor. Alla åtgärder som nämns i listan i sektion 4.2.1 på sidan 22 har vidtagits helt eller delvis. Även *phising* och *spoofing* samt XSS-attacker skulle möjligtvis kunna ge tillgång till en användares konto genom att användaren antingen blir lurad att ge ut sina användaruppgifter eller att någon stjäl information från cookies via en XSS-attack. Inga speciella åtgärder har vidtagits för *phising* och *spoofing*. För XSS-attacker finns det visst skydd i form av input-validering.

> Lösenord skall vara krypterade i databasen. Detta för att det ej skall vara möjligt att se lösenorden i klartext om någon olovligen får tillgång till databasen.

Detta åstadkoms i och med att lösenordet krypteras före det sparas i databasen. Fastän någon kommer över det krypterade lösenordet för en specifik användare går det inte att logga in på sajten med det, eftersom lösenordet skall skickas i klartext till servern som sedan krypterar det. Det uppstår dock en ny säkerhetsbrist i och med att lösenordet skickas i klartext till servern, någon som lyssnar av trafiken från användarens dator kan se lösenordet. Detta kan avhjälpas med *SSL* men har inte implementerats då tid inte har funnits för detta.

> Det skall vara så svårt som möjligt för obehöriga att komma åt information från databasen, med tanke på att mailadresser och personuppgifter finns lagrade där.

SQL-injektioner är den stora biten här, detta diskuterades tidigare i denna sektion. Uppgiften att skydda informationen ligger även mycket i händerna på den organisation som har hand om databasen och webbservern. Kommer man åt filerna den vägen är det lätt att komma åt alla information som finns i databasen.

> Det skall inte vara möjligt för någon obehörig att ta bort eller redigera information på sajten.

En användare autentiseras på varje sida som kräver att det är känt vilken användare som utför operationer. Så länge en ondsint användare inte på något sätt har lyckats logga in på någon annans konto bör det inte vara möjligt att redigera information på sidan.

> Det skall inte vara möjligt att stjäla information om en användare, i form av cookies och så vidare, när användaren är online.

Detta har främst med XSS-attacker att göra. Som det nämndes ovan appliceras input-validering för att försöka förhindra XSS-attacker (och SQL-injektioner). Dock måste användaren själv ta ansvar när hon surfar runt på andra sidor.

> Sajten skall vara skyddad mot överbelastnings-attacker (DOS-attacker) och brute-force attacker.

Tyvärr har det inte funnits tid att vidta åtgärder mot dessa typer av attacker. Meningen är dock att det för inloggning skall göras ett system där man har exempelvis fem försök på sig att logga in. Misslyckas alla försök måste man aktivera sitt konto igen via ett mail som skickas till mailadressen som använts för inloggningsförsöken. Vidare är det meningen att antalet sökningar på resor skall begränsas på något sätt, en sökning per 5 sekunder eller liknande är planen.

Kapitel 5

Resultat

Det här kapitlet visar resultatet av projektet och diskuterar om målet uppfyllts. Målet var att skapa stabil grund för en hemsida som möjliggör samåkning inom Sverige. Funktionsmässigt skulle alla funktionella och icke-funktionella krav uppfyllas. Resultatet delas in i tre delar, design, underliggande struktur och funktioner.

5.1 Design

De funktionella kraven styrde ej speciellt mycket över sajtens design. De icke-funktionella kraven 2, 3, 4 och 5 inverkade desto mer över designen. Om dessa krav har uppfyllts beror till stor del på subjektiva bedömningar, men motiveringarna nedan bör övertyga läsaren om att dessa krav är uppfyllda. Först citeras varje icke-funktionellt krav, sedan visas motiveringen till varför det är uppfyllt.

> Icke funktionellt krav 2: Webbtjänsten skall vara lättanvänd och inte ha för mycket information som syns hela tiden så att användarna blir förvirrade.
>
> Icke funktionellt krav 3: Webbtjänsten skall se bra ut, i den mening att den skall vara behaglig att kolla på och använda samt att alla sidor skall vara konsistenta i sitt utseende.

Designen på hela sajten är rund och mjuk vilket medför att den upplevs som behaglig att kolla på. Sidan är inte överarbetad och plottrig vilket gör att den blir lättanvänd och användare skräms inte iväg av oreda. Färgerna är lugna och diskreta, det gröna för tankarna till miljön. Det finns väl avgränsade delar (boxar) som strukturerar upp innehållet på ett bra sätt. Se alla figurer i detta kapitel för bilder på sajtens design.

Vidare illustrerar tabell 2.1 att många användbarhetsprinciper är implementerade vilket också förbättrar användarens helhetsupplevelse. Dessutom har alla funktioner och formulär utvecklats så att det skall krävas minimalt med arbete av användaren för att färdigställa en process, bara de allra viktigaste parametrarna måste anges. I stället finns det på många ställen fält för text där användaren kan skriva in övrig information om hon så önskar. Alla dessa punkter förbättrar upplevelsen för användaren när hon använder systemet.

> Icke funktionellt krav 4: Sajten måste passa ”alla” befintliga skärmupplösningar för stationära och bärbara datorer. Handdatorer, mobiltelefoner och liknande enheter är undantagna.

Webbsajten har testats på flertalet olika datorer med olika skärmar och upplösningar och den fungerar på alla dessa. Bredden på varje sida är konstant 955 pixlar vilket ansågs vara en bra bredd då den allra största majoriteten av skärmar idag minst har en upplösning på 1024x768 pixlar enligt *W3Schools*[1] och *hobo*[2]. Varför bredden valdes till 955 pixlar och inte 1024 är att det ofta försvinner lite utrymme på sidan till förmån för scroll-lister och eventuella verktygsfält. Höjden på sajten är variabel beroende på vilket innehåll som skall visas. Höjden på en sida spelar inte lika stor roll som bredden eftersom det är enklare och mer accepterat att scrolla i höjdled än i sidled.

> Icke funktionellt krav 5: Sidan måste fungera och visas korrekt i åtminstone de två största webbläsarna *Internet Explorer* och *Firefox*.

Webbsajten har kontinuerligt testats i *Firefox 3.0* samt *Internet Explorer 7*. Vissa mindre skillnader design-mässigt kan noteras vid noga kontroller, men i stort sett ser det identiskt ut och allting fungerar.

5.2 Underliggande struktur

Den underliggande strukturen är det område där mest arbete har lagts ner, men det är även de som är svårast att visa upp och kontrollera. I kapitel 3 på sidan 15 beskrivs det hur implementationen av systemet har gjorts och vilka delar systemet består av. Implementationsdelen visar i och med dess välgenomtänkta struktur och användande av MVC-modellen att första icke funktionella kravet är uppfyllt:

> Den underliggande strukturen på webbtjänsten skall vara väl genomtänkt och enkel att arbeta med och uppdatera i framtiden.

I kapitel 4 på sidan 21 görs en fördjupningsstudie om webbsäkerhet. I sista sektionen på fördjupningsstudien diskuteras det vilka av säkerhetsåtgärderna som är relevanta för den utvecklade webbtjänsten samt om åtgärderna är vidtagna. Detta uppfyller det icke funktionella kravet nummer 6:

> Sidan skall ta hänsyn till de relevanta säkerhetsaspekterna som diskuteras i fördjupningsstudien i kapitel 4 på sidan 21.

5.3 Funktioner

Denna sektion visar att de funktionella kraven har uppfyllts genom att demonstrera och förklara funktionerna i systemet med text och bilder, vilket kan ses som slutprodukten av arbetet. Varje funktionellt krav citeras inte, som det har gjorts med de icke funktionella kraven, men alla funktionella krav är uppfyllda även om inte exakt varje punkt nämns.

5.3.1 Registrera ny användare

Figur 5.1 på nästa sida illustrerar hur en användare kan registrera sig på sajten. När användaren har klickat på ”Registrera” skickas ett bekfräftelsemail till den givna mail-adressen. Mailet innehåller välkomsttext samt en länk. När användaren klickar på länken tas hon till aktiveringssidan och kontot aktiveras.

[1] `http://www.w3schools.com/browsers/browsers_display.asp`
[2] `http://www.hobo-web.co.uk/tips/25.htm`

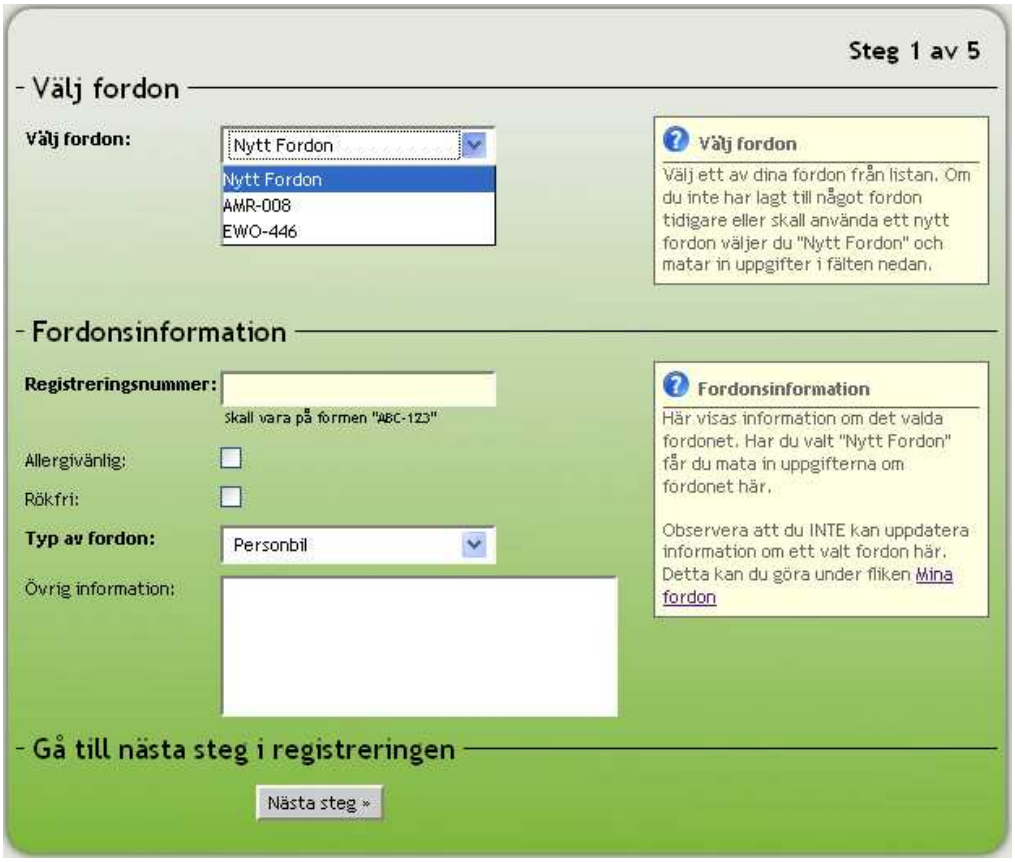
Figur 5.1: Användaren matar in sina personliga uppgifter för att kunna registrera sig på sidan.

5.3.2 Skapa en resa

För att en användare skall kunna lägga till en annons i systemet måste hon vara inloggad. Först väljs vilken typ av resa som skall genomföras, samåkningsresa eller evenemangsresa, samt om hon erbjuder samåkning eller söker samåkning. För en samåkningsresa måste användaren gå igenom följande steg innan annonsen är redo att läggs in:

1. Om användaren erbjuder samåkning väljer hon vilket fordon som skall användas för resan. Detta för att de som är intresserade av att åka med på resan skall veta vilken bil de kommer att åka med. Användaren kan välja att skapa ett nytt fordon eller välja något av sina befintliga fordon. Detta förfarande visas i figur 5.2 på följande sida.

2. Användaren väljer vart hon åker ifrån genom att antingen söka på adressen eller genom att välja bland de ställen som hon tidigare skapat. Om användaren har angivit en giltig adress, flyttas markören på kartan till vald position och kartan förstoras till en förutbestämd nivå, beroende på hur detaljerad den givna adressen är. Användaren kan även, efter att hon sökt på en adress, dra markören på kartan för att mer precis specificera positionen. *Google Maps* försöker hitta en giltig adress där användaren släpper markören, och om en adress som är mer detaljerad än den



Figur 5.2: Användaren väljer vilket fordon som skall användas för en resa där hon erbjuder samåkning.

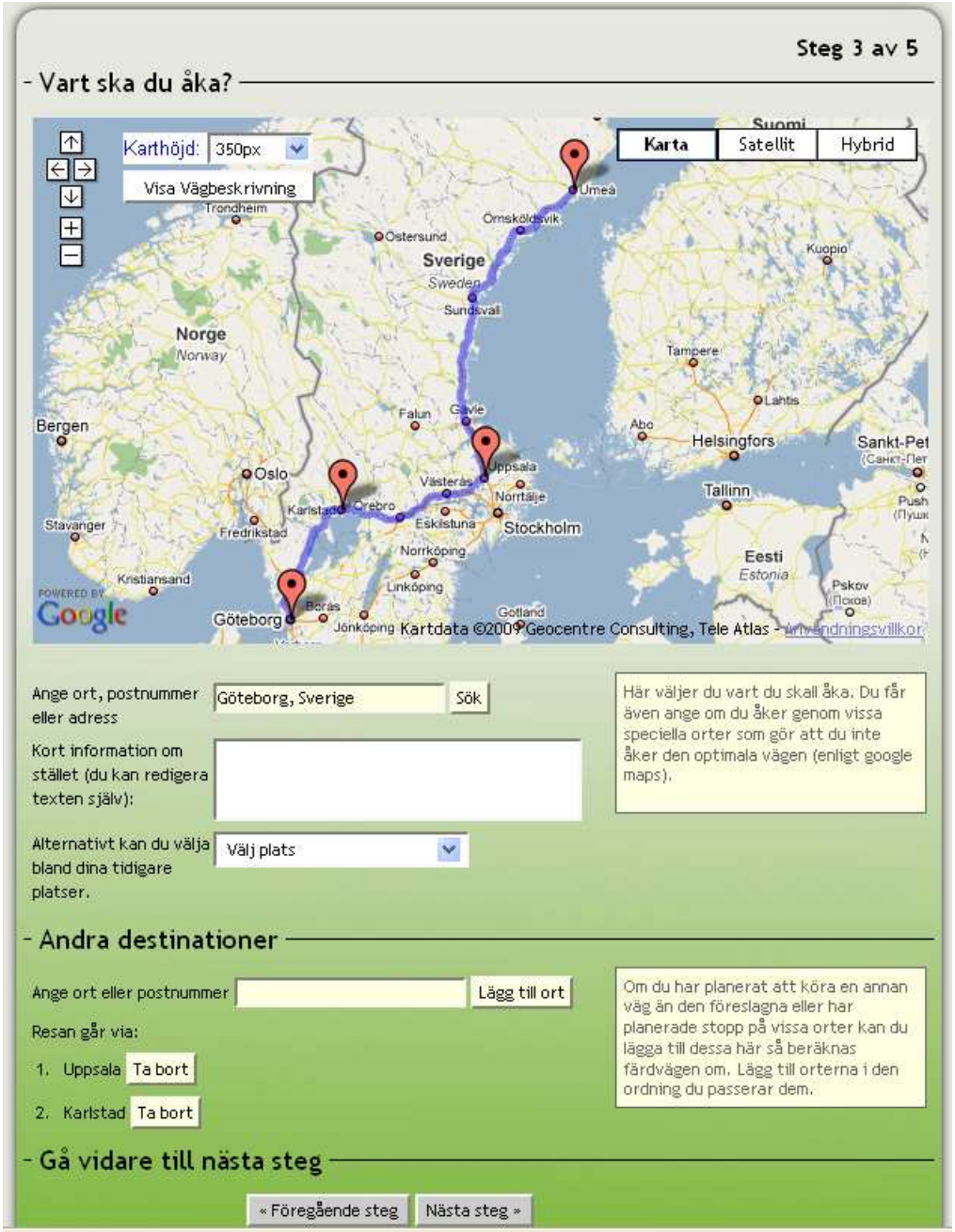
tidigare angivna adressen hittas frågar systemet om användaren vill använda den nya adressen i stället.

3. Figur 5.3 på nästa sida visar hur användaren kan välja vilket ställe hon skall åka till, vilket går till på samma sätt när användaren väljer vart hon skall åka ifrån. Skillnaden i detta steg är att färdvägen visas på kartan i stället för bara stället hon väljer. Vidare kan användaren även välja att lägga till ett antal ”via-orter”, vilka är orter som resan passerar genom. Detta är framförallt användbart om färdvägen som *Google Maps* beräknar, inte stämmer överens med den verkliga färdvägen för resan. ”Via-orterna” används även när sökningar görs på orter, vilket medför att en användare kan hitta resor som passerar stället där hon vill ansluta.

4. Användaren matar in övrig information om resan. Tid, datum, om restiden är flexibel med mera.

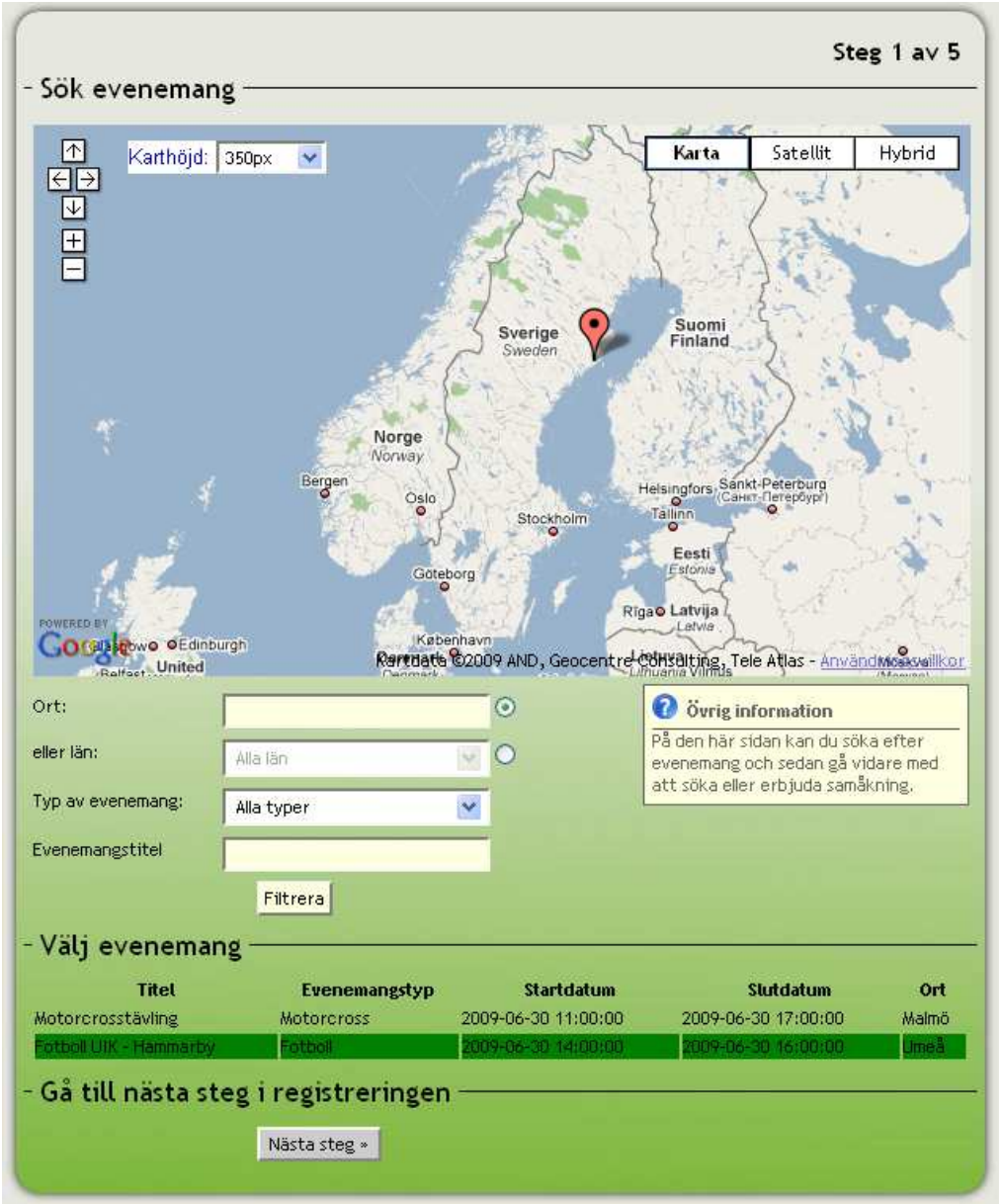
5. En summering av all information visas och resan sparas i systemet när användaren trycker på ”Spara resan”. Ingen information sparas i databasen förrän efter detta steg, detta för att kunna avbryta processen under vilket steg som helst och undvika

att lämna ofullständig data i databasen.



Figur 5.3: Användaren väljer vart resan slutar och färdvägen mellan de två orterna visas på kartan.

Om en evenemangsresa skall skapas är processen i princip densamma som för samåkningsresor. Skillnaden är att steget där användaren får välja vart resan slutar är utbytt mot ett steg där användaren får välja vilket evenemang resan skall gå till. Detta formulär visas i figur 5.4. Användaren kan i det formuläret filtrera evenemang på ort eller län, typ av evenemang och evenemangstitel. På kartan visas orten som användaren har valt eller orten som sökningen skall ske på. Om användaren har valt ett evenemang visas i stället orten där evenemanget sker.



Figur 5.4: Användaren väljer vilket evenemang resan skall gå till.

5.3.3 Söka resor

När användaren skall söka efter resor är det första valet om det skall vara en samåknings-resa eller en evenemangsresa. För samåkningsresor skall följande parametrar används för en samåknings-sökning, vilket även illustreras i figur 5.5 på nästa sida:

- Annonsören erbjuder samåkning, anger att annonsören måste ha tillgång till ett fordon och erbjuda samåkning.
- Annonsören söker samåkning, innebär att den som gör sökningen har ett fordon att tillgå och vill att folk skall åka med.
- Tidigast avfärd, anger tidigast vilket datum resan skall genomföras.
- Senast avfärd, anger senast vilket datum resan skall genomföras.
- Från ort anger att resan måste starta från en viss ort, sidan ger förslag utifrån de orter som finns i systemet. Alternativt går det att välja vilket län resan skall starta från.
- Till ort anger att resan måste gå till en viss ort, sidan ger förslag bland de utifrån som finns i systemet. Alternativt går det att välja vilket län resan skall gå till.
- Allergivänligt fordon, om fordonet måste vara allergivänligt. Om rutan ej är ikryssad innebär det att det inte spelar någon roll, alltså både resor med allergivänliga och ej allergivänliga fordon visas.
- Rökfritt fordon, om fordonet måste vara rökfritt. Om rutan ej är ikryssad innebär det att det inte spelar någon roll, alltså både resor med rökfria och ej rökfria fordon visas.

När användaren har gjort en sökning visas en lista med alla resor som matchade sökningen. Det är möjligt att klicka på varje resa, antingen i listan eller på kartan för att visa mer information om resan. Alla valda resor markeras med en grön linje på kartan, till skillnad från den röda linjen när en resa ej är vald.

Genom att trycka på knappen ”Mer information/Intresseanmälan” visas en ny sida med ännu mer detaljerad information om resan. Härifrån är det möjligt att anmäla sitt intresse på resan genom att trycka på knappen ”Anmäl Intresse”. När en intresseanmälan görs skickas ett mail till annonsören. Mailet innehåller en bekräftelse på anmälan samt kontaktuppgifter till den intresserade personen för att möjliggöra kontakt mellan dem.

Om en sökning bland evenemangsresorna skall utföras måste man först välja vilket evenemang resan skall gå till. Detta görs på samma sätt som i figur 5.4 på motstående sida. När ett evenemang är valt syns alla resor som går till evenemanget. Det är sedan möjligt att filtrera bland dessa resor på liknande sätt som för samåkningsresor.

5.3.4 Annonshantering

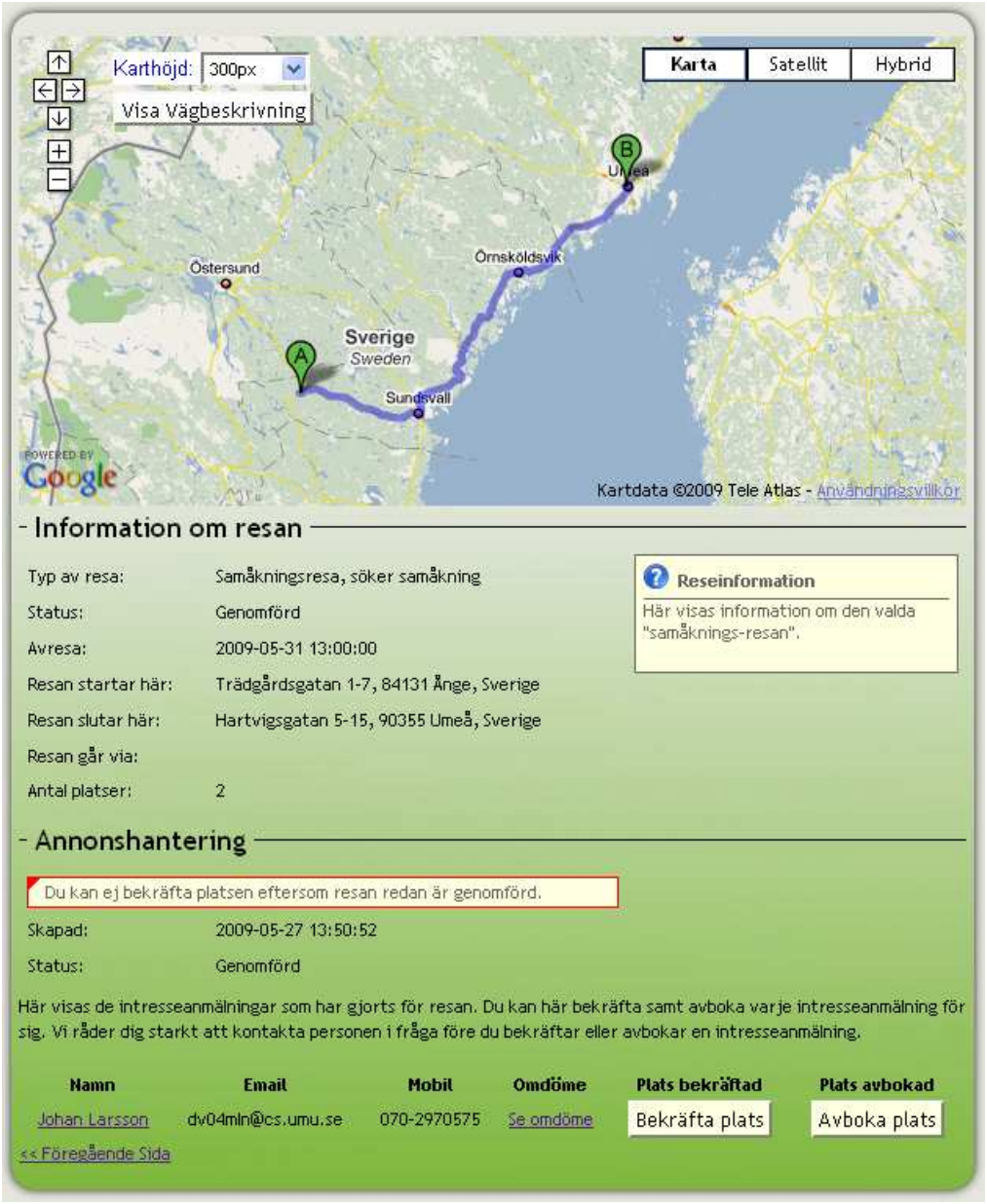
Det finns två viktiga funktioner för att hantera annonser. Under fliken ”Mina annonser” syns alla annonser man själv har lagt upp. De är uppdelade på samåkningsresor och evenemangsresor och både gamla och nya resor visas. Statusfältet visar om resan är genomförd, har avbrutits, är fullsatt eller är aktiv.

När användaren går vidare med en resa därifrån visas en detaljerad vy över den valda resan, figur 5.6 på sidan 39. I denna vy är det möjliga att avbryta annonsen och

Figur 5.5: Söksidan för en samåkningsresa.

markera resan som fullsatt. Man ser även vilka personer som är intresserade av att samåka på resan och det är möjliga att bekräfta samt avboka platser för personerna som är intresserade.

Den andra funktionen som hanterar annonser är ”Mina intresseanmälningar”. Där syns alla de resor som man har anmält sitt intresse på och för varje resa visas status samt om annonsören har bekräftat en plats för samåkning. Sidan för intresseanmälningar visas i figur 5.7 på följande sida.



Figur 5.6: Annonsörens vy för en resa där status för resan kan uppdateras och platser för intresserade användare kan bekräftas och avbokas.

Figur 5.7: Alla resor som användaren har anmält sitt intresse på. Är möjligt att se om platsen är bekräftad samt att avboka platsen.

Kapitel 6

Diskussion

Det här kapitlet diskuterar vilka problem som har uppstått under utvecklingens gång, vilka begränsningar systemet har samt vilka utvecklingsmöjligheter som finns med sajten. Avslutningsvis dras även några sista slutsatser om projektet och vilka planer som finns för företaget i framtiden.

6.1 Problem och svårigheter

Idén till projektet verkade till en början vara väldigt simpel. Men det var som vid många projekt man tar sig an, mer än väntat som behövde göras och allting tar mer tid än man tror. I det stora hela har dock allt flutit på bra, men vissa funktioner som från början var planerade har fått sättas på väntelistan på grund av tidsbrist.

Eftersom den datavetenskapliga utbildningen i Umeå gått har haft fokus på problemlösning och inte på grafisk design och användbarhet var själva utformningen av sajten den del jag oroade mig mest för. Det har varit rätt mycket arbete med att få sajten att se ut som den gör nu, men jag har hela tiden fått tips och idéer från min samarbetspartner Nicklas samt från folk runt omkring mig så det har gått över förväntan med designarbetet. Det största problemet med designen har varit att få sidan att fungera med både Firefox och Internet Explorer.

Sett till den underliggande implementationen har allting gått relativt smärtfritt, mycket beroende på den ordentliga planeringen som utfördes i början av projektet. Tack vare planeringen har det varit enkelt att veta vad som skall göras och på vilket sätt. Problem med implementationen har varit att buggar ibland har smugit sig in, samt att tekniker som var nya för mig (PHP/JavaScript/Google Maps) krävde inlärning och en del testning före de kunde användas. Som alltid med nya tekniker lär man sig nya saker allt eftersom och hittar bättre sätt att lösa problem på. Skulle jag göra om allting igen hade jag förmodligen gjort vissa saker annorlunda med tanke på det jag kan idag.

6.2 Begränsningar

Det kan vara svårt att skilja på vad som är en begränsning och vad som bara är funktioner som inte har hunnit implementeras ännu (framtida arbete). Denna sektion tar upp några av de begränsningar som jag anser finns i systemet.

Sökningarna efter resor är inte helt optimala. I dagsläget går det inte att matcha resor på orter som ligger på färdvägen mellan start- och slutdetinationen, förutom på de orter som har lagts till som ”via-orter”. Jag ser det inte som en lösning att använda *Google Maps* vid varje sökning för att se om en given ort ligger på färdvägen, eftersom det skulle bli en oeffektiv lösning. En lösning som jag däremot skulle kunna tänka mig är om det går att göra någon sorts spindel som traverserar färdvägen när en annons läggs till och sparar alla orter den hitta som ”via-orter”.

Det skulle vara bra om det gick att lägga till en resa där man både söker eller erbjuder samåkning. Detta bör inte vara några större problem att implementera men har inte gjorts.

I dagsläget fungerar tjänsten bara i Sverige och all text på sajten är på svenska. En lösning för språkstöd finns implementerad, men utnyttjas inte då det krävs en del jobb för att få den helt fungerande. För att få tjänsten att fungera i andra länder måste vissa underliggande modifikationer göras eftersom olika länder delas in i regioner på olika sätt och alla har inte postnummer med mera. Det är dock fullt möjligt att göra detta.

Från början av projektet var det meningen att det skulle finnas tre typer av resor, samåkningsresor, evenemangsresor och pendlarressor. Tiden blev dock för knapp för att hinna implementera pendlarresor. En pendlarresa är en resa som utförs regelbundet till exempelvis jobbet.

Sajten tar ingen hänsyn till personer som har *JavaScript* avaktiverat i sin webbläsare. Till viss del skulle det gå att genomföra förbättringar på denna punkt. Egentligen är det enda som verkligen kräver tillgång till kartorna (och därmed *JavaScript*) när annonser skapas. Kartorna används i alla andra fall endast för att göra informationen tydligare och sidorna lättare att använda.

6.3 Framtida arbete

Framtiden för sajten ser mycket ljus ut, i alla fall i utvecklingsarbetet. Det finns en stor mängd funktioner och förbättringar som kommer att göras om arbetet med sajten fortsätter. Den här sektionen listar några av de många idéerna som finns och förklarar vad de har för syfte. Många av funktionerna finns redan implementerade till viss del i systemet, men har inte färdigställts och visas därför inte på sidan i dagsläget.

- **Pendlarresor**. Personer som varje dag/vecka pendlar till jobbet skall kunna hitta varandra för att kunna samåka. Som systemet är just nu finns det inget bra sätt att förmedla samåkning för pendlare, men det är en av de första funktionerna som kommer att läggas till.
- **Sökoptimeringar** måste göras så att sökmotorer hittar webbsajten enklare.
- **CO_2-mätare** som mäter hur mycket utsläpp varje användare minskar samt hela sajten totalt. Exakt hur detta skall lösas är ej bestämt.
- **Grupphantering**. Det skall vara möjligt att skapa användargrupper på sajten och skapa annonser som bara dessa grupper kan se.
- **Tipsa en kompis**. Det skall vara möjligt att tipsa en kompis om tjänsten via mail. Det kanske även går att göra någon sorts ”bonussystem”, där användare som tipsat andra personer som sedan registrerar sig kan få någon slags belöning för det.

- **RSS-feeds**. Publicera nya annonser via RSS och dela upp dem på rimligt sätt så att användare kan prenumerera på RSS-feeden och kontrollera den direkt i mobilen eller dylikt.
- **Utöka sökfunktionen** genom att tillåta mer generella sökningar angående vart ifrån och till var resan skall gå. Exempelvis säga att man vill åka Söderut från Umeå.
- **Företagsdefinierade funktioner** med till exempel biluthyrningsföretag, släpvagnsuthyrningsföretag, landsting med flera. Vissa idéer finns för sådana funktioner, men inget är bestämt ännu.
- **Inkorporera webservices**. Om sajten växer och blir större kommer det att bli aktuellt med att använda webservices till vissa funktioner. Bland annat kommer att att göras en webservice för sökning så att företag och organisationer kan göra egna specialanpassade sökverktyg och använda dessa på deras interna sidor.
- **Lokal reklam**, antingen utifrån färdvägen på resan som man kollar på för tillfället eller utifrån IP-adressen som användaren sitter på. På det viset vet reklamannonsörerna att deras reklamannonser verkligen visas för potentiella kunder, och inte för användare som aldrig kommer att vara i närheten av reklamannonsörens företag.
- **Chatforum** för varje resa. Bara de som anmält intresse skall kunna se diskussionen.
- **Meddelande-funktion**. Användarna skall kunna skicka meddelanden till varandra via sajten. På så vis blir kontakten enklare och det blir lättare att hålla reda på vilka man har pratat med och vad som har sagts.
- **SMS-funktioner** för att få förslag på resor, bekräfta bokningar med mera.

6.4 Slutsatser

Denna rapport visar hur projektet *HitchHike.se* har utvecklats, från planering till implementation och dokumentation. Målet med projektet var att skapa en stabil och väl genomtänkt grund för en webbtjänst för samåkning, som senare skulle kunna vidareutvecklas och förhoppningsvis växa sig stor och stark. Sett till de övergripande målen för projektet är de uppfyllda och webbsajten klarar av alla funktionella krav som identifierades i början av rapporten. Det finns dock, som det visas under framtida arbete i sektion 6.3 på föregående sida, väldigt mycket kvar att göra. Vissa av punkterna som ligger under framtida arbete hade vi hoppats skulle hinnas med under examensarbetet, men det fanns tyvärr inte tid till det.

I övrigt tycker vi att projektet är blev lyckat, det finns nu en fungerande webbsajt som ser bra ut, och den har de viktigaste funktionerna som behövs för att kunna köra igång sajten på riktigt. Nu återstår bara att marknadsföra projektet så att folk börjar hitta till sajten och börjar använda den, vilket i sig är ett helt projekt.

Projektet har varit kul och lärorikt att genomföra. Det har varit intressant att se hur vissa idéer bara har kommit och gått, medan vissa har implementerats och utvecklats gradvis efter diskussioner. Det är även kul att alla som vi har pratat med och berättat om idén har varit mycket positivt inställa, och vi hoppas att folk börjar använda sidan när den väl körs igång.

Kapitel 7

Tack

Jag vill först och främst tacka min samarbetspartner, Nicklas Sandberg. Utan honom hade projektet aldrig kommit igång och han har dessutom varit till stor hjälp med sina åsikter om allting. Jag vill även tacka min handledare Jerry Eriksson som har hjälpt mig med rapporten. Vidare vill jag tacka min älskade sambo, Sofie Johansson. Utan henne hade jag aldrig orkat sitta hemma själv i fem månader och arbeta med detta projekt. Slutligen vill jag tacka alla andra personer som på något sätt har bidragit med idéer och åsikter om projektet.

References

[1] Gregory Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti. Using parse tree validation to prevent sql injection attacks. In *SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware*, pages 106–113, New York, NY, USA, 2005. ACM.

[2] Web Application Security Consortium. Threat classification. `http://www.webappsec.org/projects/threat/v1/WASC-TC-v1_0.pdf`, accessed 2009-05-11, 2004.

[3] Tamara Dinev. Why spoofing is serious internet fraud. *Commun. ACM*, 49(10):76–82, 2006.

[4] W. Halfond, A. Orso, and P. Manolios. WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation. *IEEE Transactions on Software Engineering (TSE)*, 34(1):65–81, 2008.

[5] William G.J. Halfond and Alessandro Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In *Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005)*, Long Beach, CA, USA, Nov 2005. http://www.cc.gatech.edu/ whalfond/publications.html.

[6] Amir Herzberg and Ahmad Jbara. Security and identification indicators for browsers against spoofing and phishing attacks. *ACM Trans. Interet Technol.*, 8(4):1–36, 2008.

[7] Frank Kargl, Joern Maier, and Michael Weber. Protecting web servers from distributed denial of service attacks. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 514–524, New York, NY, USA, 2001. ACM.

[8] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 330–337, New York, NY, USA, 2006. ACM.

[9] Tony Marston. The model-view-controller (mvc) design pattern for php. `http://www.tonymarston.net/php-mysql/model-view-controller.html`, accessed 2009-05-11, 2004.

[10] Tyler Moore and Richard Clayton. Examining the impact of website take-down on phishing. In *eCrime '07: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 1–13, New York, NY, USA, 2007. ACM.

[11] Terri Oda, Glenn Wurster, P. C. van Oorschot, and Anil Somayaji. Soma: mutual approval for included content in web pages. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 89–98, New York, NY, USA, 2008. ACM.

[12] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, 39(1):3, 2007.

[13] Webmaster Tips. Web site usability checklist. `http://www.netmechanic.com/news/vol7/design_no4.htm`, accessed 2009-05-11, 2004.