CHALMERS | 

GÖTEBORGS UNIVERSITET

Ett mjukvarusystem för koordinering och schemaläggning av artisttransporter under musikfestivaler och event

Kandidatarbete inom Datateknik, Datavetenskap och Informationsteknik

Patrick Franz
Oskar Jiang
Simon Nielsen
Andreas Pegelow
Erik Pihl
David Ådvall

Chalmers Tekniska Högskola
Göteborgs Universitet
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, juni 2016

Eciton
A software system for coordinating and scheduling artist transportation during music festivals and events

Patruck Franz
Oskar Jiang
Simon Nielsen
Andreas Pegelow
Erik Pihl
David Ådvall

Supervisor: Moa Johansson
Examiner: Niklas Broberg

[Cover: The logo was designed by Peter Nielsen, © Peter Nielsen 2016.
Commercial use of the logo allowed.]

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

Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Förord

Denna rapport behandlar utvecklingen av systemet Eciton, som ägde rum inom ramarna för ett kandidatarbete på Chalmers Tekniska Högskola och Göteborgs Universitet. Gruppen som har stått bakom utvecklingen skulle vilja tacka Sebastian Malcus och Karl Westgårdh från Elvaett som har bidragit med projektförslaget och bidragit med stöd under projektets gång. Vi vill också tacka vår handledare Moa Johansson för hennes hjälp med genomförandet av projektet. Till sist vill vi även tacka Jens Clausen som testade Androidapplikationen och gav oss nyttig feedback angående funktionaliteten. Utan er hade inte detta projekt varit möjligt att genomföra.

Göteborg, 1 juni 2016

Abstract

This report describes the development of Eciton, a software system for coordination and scheduling of transports during music festivals and events. The system consists of a web application for coordinators, a mobile application for drivers and an automatic scheduler. The result that is generated by the scheduler are usable schedules for shifts and drives that were produced faster than what could have been achieved manually. The solution is brought about with the help of SAT-solvers and self-made algorithms. The system is a prototype that with only a few additions can be used by companies in the industry.

Coordination and scheduling are hard and time-consuming tasks that requires a lot of time and energy and involves a great deal of people. There are similar logistics programs, for example for haulages or taxi but none of these fits the work patterns that are requested by the industry at hand. It is this demand in the market that Eciton seeks to fulfil.

Keywords: scheduling, coordination, SAT, transport, software, web, mobile, application

Sammanfattning

Denna rapport beskriver utvecklingen av Eciton, ett mjukvarusystem för koordinering och schemaläggning av transporter under musikfestivaler och event. Systemet består av en webbapplikation för koordinatorer, en mobilapplikation för chaufförer samt en automatisk schemaläggare. Resultatet av det som genereras av schemaläggaren är användbara scheman för arbetspass och körningar som framställts snabbare än vad som kan åstadkommas manuellt. Lösningen tas fram med hjälp av SAT-lösare och egenskapade algoritmer. Systemet är en prototyp som med endast få tillägg kan användas av företag i branschen.

Koordinering och schemaläggning är svåra uppgifter som kräver mycket tid och energi och involverar många människor. Det finns liknande logistikprogram, till exempel för åkerier eller taxi men inget av dessa passar det arbetssätt som efterfrågas i den aktuella branschen. Det är denna efterfrågan på marknaden som Eciton syftar till att tillfredsställa.

Nyckelord: schemaläggning, koordinering, SAT, transport, mjukvarusystem, webbapplikation, mobilapplikation

Ordlista

Körning En bil och en chaufför förflyttar passagerare från punkt A till punkt B.

Transport En eller flera körningar som tillsammans kan transportera en artist och hela dess följe. Alla körningar som ingår i en transport har samma start- och sluttid och samma geografiska start- och slutposition.

Koordinator En person som är ansvarig för att samordna och schemalägga transporter.

Storyboard Används i utvecklingsmiljön XCode för att beskriva vyn för det tillstånd som applikationen befinner sig i. *T.ex Vid uppstart av appen visas en statisk text*

TCP Kommunikationsprotokoll som används på internet. Det garanterar att data som skickas når mottagaren.

Konjunktiv normalform (förkortas CNF) En form för att beskriva ett logiskt uttryck enbart med operationerna $\wedge$, $\vee$, och $\neg$. Uttrycket skrivs som en samling uttryck åtskiljda av $\wedge$ där varje uttryck innehåller variabler, som kan vara negerade, åtskiljda av $\vee$. Ett exempel på ett uttryck på konjunktiv normalform är $A \wedge (B \vee \neg C) \wedge \neg D \wedge (A \vee B \vee C \vee D)$.

Google Cloud Messaging (förkortas GCM) En tjänst, som tillåter att push-notiser och data kan skickas från en server till en mobiltelefon med Android eller iOS [1].

Innehåll

1

Inledning

Att koordinera hundratals transporter under en festival är en svår uppgift som kräver mycket planeringsarbete, bra struktur och bra kommunikation med alla inblandade. Att manuellt schemalägga dessa kräver ännu mer tid och blir vid stora antal transporter nästan praktiskt omöjligt. För att lösa detta krävs ett system som kan tillfredsställa alla dessa behov och assistera i att utföra uppgifterna.

1.1 Bakgrund

Inom Operationsanalys [2], som är ett forskningsområde för att ta hjälp av bland annat matematisk programmering och algoritmer för beslutstagande, finns ett problem som kallas Nurse Scheduling Problem [3]. Problemet handlar om att schemalägga sjuksköterskor på arbetsskift utifrån regler för hur mycket och hur ofta de får arbeta samt sjuksköterskornas preferenser gällande arbetstider. Arbetsskiften har dessutom krav och önskemål på kompetensen hos sjuksköterskorna. Detta är ett problem som studerats under flera decennier och är ett så kallat "NP-hard" (Non-deterministic Polynomial-time hard) problem som betyder att det saknar en känd tidseffektiv lösning [4].

Problemet att schemalägga sjuksköterskor är inte helt olikt det som inom musikfestivalbranschen kallas för "Ground transport". Ground transport innebär att koordinera de medverkande artisternas alla förflyttningar under en festival, exempelvis mellan flygplats och hotell eller mellan hotell och den festivalscen där de ska uppträda. Artisterna beställer körningar och specificerar krav på vilken sorts bil och chaufför det ska vara. Förarna i sin tur är bara tillgängliga vissa tider, är olika erfarna, har preferenser angående vilka tider på dygnet de vill arbeta, och får inte jobba för mycket. Under en festival som pågår i 5 dagar kan det genomföras uppemot 700 transporter uppdelade på 80 chaufförer. Informationen om när transporterna ska ske kan komma från artisterna både långt i förväg och kort inpå att transporten ska ske.

Ett företag som jobbar med detta i Sverige är Elvaett och deras största kund är

musikfestivalen Way Out West. Idag använder Elvaett kalkylark i Excel för att schemalägga transporterna och SMS för att kommunicera med chaufförer. De har automatiserat en del av sitt arbete i Excel men större delen av arbetet sker manuellt. Bland annat sker informationsutskick till chaufförer genom att manuellt kopiera data från kalkylarken till SMS. Då det är stora mängder data som måste skickas till varje chaufför blir den totala kostnaden för alla SMS hög.

Elvaett har undersökt marknaden för logistikprogram i jakt på ett system som de skulle kunna använda. De har undersökt system som används av åkerier, rederier och taxi. Trafikledningssystemen för taxi liknade det som Elvaett sökte men inget av systemen som finns på marknaden passar deras arbetssätt. För att hitta en lösning sökte sig Elvaett till Chalmers och föreslog konstruktionen av ett trafikledningssystem som ett kandidatarbete vilket den här rapporten är ett resultat av.

1.2 Syfte

Syftet med det projekt som denna rapport beskriver är att skapa en prototyp för ett trafikledningssystem som ska schemalägga och koordinera transporter vid musikfestivaler och andra event. I detta ingår också delmoment såsom att koordinatorer ska kunna kommunicera med chaufförer för att bekräfta att transporten kommer att ske som planerat, att chaufförer ska kunna se sina egna körningar, samt hantering av användarnivåer. Meningen är även att utforska olika sätt att lösa de olika delmomenten på och välja de metoder som är lämpligast för att skapa en produkt som uppfyller syftet och kraftigt effektiviserar problemområdet.

1.3 Avgränsningar

Huvudprioriteten är att skapa ett fungerande system som bara erbjuder grundfunktionaliteter. Grundfunktionaliteterna innefattar att generera och presentera ett schema samt att kommunicera detta schema till chaufförerna. Systemet ska dock skapas på ett sådant sätt att det kan utvidgas med mer avancerade funktionaliteter modulärt. Schemat som genereras ska vara giltigt men behöver inte vara optimalt. Dessa avgränsningar är baserade både på vad som bedöms är realistiskt att hinna med inom projektperioden samt vad Elvaett har önskat.

2

Problemspecifikation

Vad är det övergripande problemet och vilka delar kan problemet delas in i? Förslagsgivarna berättade om sitt arbetssätt idag och vad de skulle vilja ha för produkt. Problemet de vill lösa är att artister behöver vara på olika platser vid olika tidpunkter och de behöver transporteras däremellan på ett smidigt, kostnads-, och resurseffektivt sätt. Något som försvårar situationen är att majoriteten av bokningarna av transporter kommer in med kort varsel och många av dem kommer att ändras senare under festivalen. Mängden körningar som sker på en typisk festival är såpass stor att det är otympligt skapa ett schema manuellt.

För att kunna lösa problemet behövs ett system som automatiserar utförandet av uppgifter som är omfattande och komplexa för en människa att utföra manuellt, samtidigt som det ger användaren goda möjligheter att finjustera och styra operationen manuellt med systemet som verktyg. Därför behöver systemet bestå av flera samverkande komponenter som specificeras i detta kapitel.

2.1 Schemaläggningsprocessen

Inför en festival behöver det skapas en plan. En sådan plan beskriver

- vilka tidsintervall varje chaufför ska arbeta. Varje arbetspass kan vara antingen omkring åtta eller omkring tolv timmar långt. Varje chaufför har en preferens angående vilken arbetspasslängd den föredrar samt om den föredrar att arbeta på dagen eller på natten.
- vilken chaufför som ska bemanna vilken bil vid varje given tidpunkt under festivalen.
- vilket par av chaufför och bil som ska utföra varje körning. En körning har en geografisk start- respektive slutpunkt samt en start- och en sluttid. Den ställer också krav på kompetensnivån hos chauffören och kvalitetsnivån hos bilen.

För att säkerställa att planen blir korrekt och optimal, samt för att effektivisera planeringsarbetet, ska systemet hjälpa koordinatorn genom att generera en plan utifrån vissa förutsättningar som koordinatorn matar in i systemet. Följande förutsättningar

matas in i systemet av koordinatorn för en specifik festival.

$\mathbb{Z}$ En mängd av geografiska zoner inom vilka operationen kommer att utspela sig. För varje zon definierar koordinatorn färdtiden mellan zonen och alla andra zoner i $\mathbb{Z}$.

$\mathbb{P}$ En mängd av geografiska platser vilka kommer att utgöra start- eller slutpunkter för körningarna under operationen. Varje plats tillhör en (och endast en) av zonerna i $\mathbb{Z}$.

$\mathbb{BK}$ En mängd av bilkategorier. För varje bilkategori definierar koordinatorn vilka bilkategorier den är "bättre än". Att en bilkategori bk_1 är bättre än en bilkategori bk_2 innebär att en bil i bilkategorin bk_1 håller lika hög eller högre kvalitet än, och har lika många eller fler platser än, en bil i bilkategorin bk_2. Exempel på bilkategorier är "Personbil standard", "Personbil premium" och "Minibuss premium". I exemplet är det troligt att koordinatorn definierar att både "Personbil premium" och "Minibuss premium" är bättre än "Personbil standard" eftersom den ena sannolikt har lika många platser men högre kvalitet och den andra förmodligen har samma kvalitet men ett större antal platser. Någon relation mellan "Personbil premium" och "Minibuss standard" är sannolikt inte definierad.

$\mathbb{B}$ En mängd av bilar. Varje bil tillhör en bilkategori i $\mathbb{BK}$. För varje bil definieras en tidsperiod under vilken den är tillgänglig och får vara bemannad.

$\mathbb{FK}$ En mängd av förarkategorier. Förarkategorierna rangordnas efter kompetensnivå och det definieras vilken lägsta kompetensnivå som krävs för varje biltyp i $\mathbb{BK}$. Exempel på förarkategorier är "Standard" och "Premium". Premium är sannolikt rankad högst av dem två. En Premium-bil kräver sannolikt en chaufför av minst kategorin Premium medan en Standard-bil förmodligen tillåter både Standard- och Premium-chaufför.

$\mathbb{F}$ En mängd av chaufförer. Varje chaufför tillhör en förarkategori i $\mathbb{FK}$. För varje chaufför definieras om chauffören helst vill ha arbetspass på dagen eller på natten samt om chauffören vill ha långa eller korta arbetspass. För varje chaufför definieras också under vilka tidsintervall chauffören är tillgänglig och får schemaläggas.

$\mathbb{BE}$ En specifikation av resursbehovet vid varje tidpunkt under operationen. Resursbehovet definieras som antalet bemannade bilar som behövs per bilkategori för varje timme mellan operationens start- och sluttid. Resursbehovet en viss timme kan vara betydligt större än antalet i förväg planerade körningar den timmen eftersom beställningar av körningar kan komma in med kort varsel medan själva operationen pågår. Koordinatorn utgår från sin domänkunskap och erfarenheter från tidigare festivaler när denna definierar resursbehovet.

lp_{max} Ett maximalt antal långa pass per bilkategori per dag.

$\mathbb{K}$ En mängd av körningar. En körning är en uppgift som en chaufför, med hjälp av en bil, ska utföra. Varje körning har en start- och en slutpunkt som båda tillhör $\mathbb{P}$, samt en start- och en sluttid. Varje körning har också ett krav på sämsta tillåtna bilkategori hos bilen som ska utföra körningen. Koordinatorn kan ange manuellt att en viss körning ska vara låst alternativt halvlåst till en viss bil. En låst tilldelning får aldrig ändras av schemaläggaren. En halvlåst tilldelning får ändras av schemaläggaren, men det är inte önskvärt.

Utifrån de definierade förutsättningarna ska systemet generera en plan. Planen är giltig om och endast om samtliga av följande krav är uppfyllda.

- Antalet planerade arbetspass som pågår under varje festivaltimme är aldrig mindre än det behov som definierats i $\mathbb{BE}$ för respektive timme.
- Antalet planerade långa arbetspass per dag per bilkategori är mindre än eller lika med lp_{max}.
- Vid varje given tidpunkt är varje bil $b \in \mathbb{B}$ tilldelad max ett arbetspass a. Bilen b måste vara tillgänglig under hela den tidsperiod som a pågår.
- Låt oss kalla mängden av alla genererade arbetspass för $\mathbb{A}$. Varje arbetspass $a \in \mathbb{A}$ är tilldelat en och endast en chaufför f från $\mathbb{F}$. Chauffören f har en kategori $fk \in \mathbb{FK}$ som har samma eller högre kompetensnivå som bilen b kräver. Chauffören f är tillgänglig under hela den tidsperiod som a pågår. Om a är ett långt pass måste chauffören ha egenskapen att den helst vill ha långa pass. Däremot är det inget krav att korta pass måste tilldelas chaufförer som helst vill ha korta pass.
- En chaufför får tilldelas max ett arbetspass per dygn.
- Om koordinatorn manuellt har låst en chaufför till ett arbetspass så ska den och endast den chauffören tilldelas det arbetspasset.
- Varje körning $k \in \mathbb{K}$ har tilldelats en och endast bil $b \in \mathbb{B}$. Den bilkategori som b ingår i är samma som eller bättre än den sämsta tillåtna bilkategori som k kräver. Bilen b är bemannad under hela den tidsperiod som körningen k pågår.
- En bil $b \in \mathbb{B}$ får inte tilldelas både körning k_1 och körning k_2 om den ena av körningarna börjar under den tidsperiod som den andra av dem pågår. Detsamma gäller om tiden det tar att köra från slutdestination för den av körningarna som inträffar först till den andra körningens startposition är större än tiden mellan den första körningens sluttid och den andra körningens starttid.
- Om koordinatorn manuellt har låst en bil till en körning ska den och endast den bilen tilldelas den körningen.

Om det finns en plan som uppfyller alla krav är det sannolikt att det finns flera olika giltiga planer som är olika bra. Det är önskvärt men inget krav att systemet

kan försöka hitta en så bra plan som möjligt inom den mängd av giltiga planer som har genererats utifrån följande önskemål.

- Arbetspassen är skapade på ett sådant vis att antalet bilar som behövs är så litet som möjligt.
- Om flera lösningar har arbetspass skapade på ett sådant vis att de kräver lika många bilar är den lösning av dem som kräver minsta sammanlagda antal arbetstimmar bäst.
- Om flera lösningar har arbetspass skapade på ett sådant vis att de kräver lika många bilar och lika många arbetstimmar är den lösning av dem vars antal tolvtimmarspass per bilkategori per dag är närmast lp_{max}.
- Så många chaufförer som möjligt ska få sitt önskemål om att arbeta på dagen respektive på natten uppfyllt.
- Så många chaufförer som möjligt ska få sitt önskemål om att arbeta långa respektive korta pass uppfyllt.
- Som standard är det önskvärt att körningarna är fördelade över bilarna på ett sådant vis att varje bil har så lång paus som möjligt mellan sina körningar. Det skapar en robusthet som gör att schemat lättare kan hantera förseningar till följd av exempelvis oväntade bilköer. Dessutom är det bra för förarnas arbetsmiljövillkor då det blir fler chanser för förarna att ta en paus för att exempelvis äta eller vila.
- I specialfallet att en körning slutar strax innan en annan körning börjar och slutdestinationen för den första körningen är samma som startpositionen för den andra körningen är det önskvärt att samma bil tilldelas båda körningarna. Detta ökar resurseffektiviteten eftersom det kan innebära exempelvis att en bil som precis har lämnat en artist på flygplatsen väntar på flygplatsen och tar en körning som går därifrån en kvart senare istället för att åka tillbaka till centrum efter den första körningen och låta en annan bil åka till flygplatsen för att ta den andra körningen.
- För varje tilldelning av en bil till en körning är det önskvärt att bilen är så dålig som möjligt givet att den uppfyller körningens krav. Eftersom körningar kan beställas när som helst fram till och under festivalen är det bra om de bilar som är obokade vid varje givet tillfälle är så bra som möjligt eftersom en bil som är bättre än en annan bil är kompatibel med ett större antal okända potentiella körningar än den andra bilen.
- Så få körningar som möjligt som är halvlåsta till en viss bil ska tilldelas en annan bil än den som de är halvlåsta till. En körning k kan vara halvlåst till en bil b exempelvis om det har kommunicerats till chauffören eller till en extern part att det är bil b som kommer att utföra körning k. Att behöva kontakta den part som detta har kommunicerats till en gång till för att meddela att

tilldelningen har förändrats är inte önskvärt.

2.2 Prestanda

För att systemet ska vara användbart krävs både att det producerar ett schema av hög kvalitet och att det inte tar för lång tid för systemet att utföra sin uppgift. Eftersom schemaläggning i allmänhet kan vara en svår uppgift för en dator att lösa kommer schemaläggaren att bli en flaskhals när det kommer till väntetider för den som använder systemet. Därför specificeras krav på de olika delarna av schemaläggaren enligt följande. (I samtliga krav avser begreppet *festival* ett projekt i storleksordningen någon av Sveriges allra största festivaler.)

Skapandet av arbetspass är vanligen något som görs en gång inför en festival, några veckor innan festivalen börjar. I detta skede är ett acceptabelt scenario att koordinatorn förbereder indatan under dagen, startar schemaläggaren på kvällen, och vaknar nästa morgon med en färdiggenererad uppsättning arbetspass på skärmen. Kravet för denna del är att systemet ska ta maximalt åtta timmar på sig för att producera en användbar uppsättning arbetspass.

Även ihopparning av chaufförer och arbetspass sker i normalfallet en gång inför en festival, åtminstone en vecka innan festivalen börjar. Kravet är att systemet ska kunna generera ett användbart arbetsschema på maximalt åtta timmar.

När det kommer till planering av vilken bil som ska utföra vilken körning är förhållandet annorlunda. Ett sannolikt förfarande är att detta görs en gång med en stor mängd data strax innan festivalens början, och sedan ett antal gånger under festivalens gång med betydligt färre variabler i indatan. Därför ställs två olika krav. Med indata motsvarande samtliga körningar för en festival ska systemet, givet en uppsättning bemannade bilar, producera ett användbart schema på under tio minuter. Om indatan innehåller lika många körningar men enbart femton av dem inte redan är ihopparade med och låsta till en bil ska systemet göra samma sak på under en minut.

2.3 Användarvänlighet

Systemet kommer att användas av koordinatorer och chaufförer. Dessa är personer som är anställda och får betalt för att göra ett jobb och som nödvändigtvis inte har en stor datorvana. Majoriteten av den tid användarna kommer att interagera med systemet kommer att vara medan en festival fortlöper.

De olika användarna har helt olika behov av systemet och är i helt olika situationer när de använder systemet. Systemet måste därför vara väl anpassat efter de olika

ändamålen och dess användare. Nedan följer beskrivningar av de olika fallen.

Koordinatorer
Koordinatorerna sitter på ett kontor där de samordnar transporter. De har ständig kontakt med chaufförer och turnéledare. Arbetsdagar upp till 18 timmar per dag är inget ovanligt under en festival enligt Elvaett. I ett jobb där små detaljer är viktiga och att kunna ge snabba svar blir användarvänligheten viktig. Därför behövs ett gränssnitt som tillåter användaren att utföra sin uppgift och inte se systemet som som ett hinder utan som ett hjälpmedel. Koordinatorerna kommer dock att använda systemet i en sådan utsträckning att det kan anses acceptabelt att det tar lite tid att lära sig att använda systemet för att utnyttja det till max. Det kan vara saker som kortkommandon för att nå olika funktioner eller att skriva in datum på rätt format för att slippa använda musen för att klicka i en grafisk kalender. Målet är ett system som i så stor grad som möjligt är fritt från excise [5] vilket är onödiga steg användaren måste gå igenom för att komma till sitt mål. Systemet ska aldrig hindra användaren från att schemalägga "fel", däremot ska det föreslå det "korrekta" sättet och hjälpa användaren för att göra dess jobb så enkelt som möjligt och undvika den mänskliga faktorn. Exempelvis bör systemet inte hindra användaren från att planera två körningar med samma chaufför samtidigt, dock ska systemet ge en varning för att på ett smakfullt sätt låta användaren veta att det kan uppstå ett problem.

Chaufförer
Den andra stora användaren av systemet är chaufförerna vilka ska använda en smartphoneapplikation för att interagera med systemet. Speciellt i detta fallet är att chaufförerna till stor del kommer använda systemet när de sitter i en bil. Systemet ska vara designat så att chaufförerna inte behöver interagera med applikationen medan de kör. Applikationen ska bara användas när chaufförerna har parkerat och behöver information inför nästa körning. Därför ska de notifikationer som applikationen producerar inte synas förrän att användaren själv öppnar telefonen. Att chaufförerna fokuserar på körningen kommer alltid att vara det viktigaste för att försäkra chaufförerna och dess passagerares säkerhet.

Utöver det ska applikationen vara designad på ett sådant sätt att chauffören ska få tillgång till all information den behöver utan att behöva leta eller använda alltför många skärmtryck. Det bör inte heller finns någon onödig information och chaufförerna bör endast ha tillgång till sina egna körningar och sin egen information.

För att tröskeln att börja använda applikationen ska vara så låg som möjligt ska det visuella vara designat med de designmönster som finns etablerade på marknaden idag. Om knappar ser ut som knappar brukar göra och navigation i applikationen funderar som navigation brukar göra kommer det vara lätt för användarna att börja använda systemet då de redan är familjära med de visuella elementen.

2.4 Kommunikation

För att kunna koordinera och genomföra en festival behöver koordinatorer kunna kommunicera med bland annat chaufförer och turnéledare. De behöver få tag på relevant information samt kunna skicka eller begära information från chaufförerna och tanken är att detta ska ske genom en mobilapplikation.

Då körningar har tilldelats till chaufförerna behöver koordinatorerna få bekräftelse på att de har tagit del av denna information och avser att genomföra dessa körningar. För varje körning ska sedan ytterligare bekräftelser skickas från chauffören till koordinatorn vid körningens start och dess slut. Allt detta för att hålla koll på om allting går som planerat, eller om problem har uppstått någonstans.

2.5 Feltolerans och manuell kontroll

Det är nästintill omöjligt att bygga ett system som kan hantera exakt alla typer av händelser. Därför är det viktigt att delar av det har stöd för manuella ändringar som åsidosätter de vanliga rutinerna. Detta är essentiellt för användbarheten på sikt då det måste vara tolerant för specialfall som kräver avvikande lösningar. Moderna system är ofta beroende av andra system för att få viss funktionallitet, till exempel använder många system internet för att kommunicera. Det skapar ett krav på system att hantera situationer då ett externt system inte fungerar som det ska. Ett krav på detta system är att koordinatorerna ska kunna fortsätta sitt arbete om deras kontakt med internet bryts.

2.6 Lagra information

Det kanske ses som självklart att för att schemaläggningen ska fungera så måste det lagras information om chaufförer, bilar och artister. För att underlätta arbetet för koordinatorerna måste systemet också kunna lagra information som är relevant för de praktiska arbetet men inte för schemaläggaren. Detta är information som användarna måste tillhandahålla. Även den informationen som algoritmen ger som resultat måste lagras. Exempel på information som behöver lagras är:

- Alla geografiska platser involverade i festivalen.
- Profilbilder på förarna.
- Telefonnummer till nyckelpersoner inom festivalen.
- Vilken chaufför som kör vilken bil vid en vis tidpunkt.

- Alla bilars registreringsnummer.

Denna information måste gå att påverka i efterhand, göra ändringar i, lägga till ny och ta bort information.

3

Teknisk bakgrund

I följande kapitel förklaras de tekniska verktyg som gruppen använde under projektets gång. Begrepp som är viktiga för förståelsen av systemet reds ut och tekniken bakom systemet presenteras.

3.1 HTTP

HTTP [6] är ett protokoll som används för att skicka data. Det bygger på en förfrågan-svar modell där en klient skickar en förfrågan till en server som svarar. En förfrågan innehåller en uppgift som klienten vill att servern ska utföra, exempelvis att spara data på servern eller att servern ska skicka data som den har lagrad. Servern svarar med information om resultatet.

Ett vanligt användningsområde är kommunikation mellan webbservrar och webbsidor, men det kan också användas av andra applikationer.

3.2 WebSocket

WebSocket [7] är ett protokoll som skapades för att ge webbläsarapplikationer tillgång till tvåvägskommunikation med webbservrar. Innan WebSockets-protokollet fanns användes HTTP till all trafik. HTTP tillåter endast att en server skickar data efter att en klient har skickat en förfrågan. Så för att en klient kontinuerligt ska få ny information från en server måste denna hela tiden skicka förfrågningar till serven. När man använder HTTP på detta sätt uppstår en del problem, bland annat att servrar måste ha flera TCP-anslutningar för varje klient.

För att enklare lösa detta används protokollet WebSocket. Det tillåter både klienten och servern att skicka data samtidigt över en TCP-anslutning. WebSocket är inte byggt för att ersätta HTTP utan för att användas tillsammans med det. Båda protokollen kan användas samtidigt av en server utan att de påverkar varandra.

3.3 JSON

JSON [8] är ett textformat för serialisering av data som bygger på Objekt-notationen i JavaScript. Det har stöd för 6 olika typer av data; strängar, tal, booleska variabler, null, vektorer samt objekt. En vektor är en ordnad mängd av data. Ett objekt är en oordnad mängd av nyckel-värde-par där namnet är en sträng och värdet är data av någon av de tillåtna typerna. Det använder 6 olika tecken för att beskriva strukturen på data. En vektor avgränsas med fyrkantsparenteser([]) och datan i vektorn separeras med kommatecken(,). För avgränsning av objekt används klammerparentes ({}) och data separeras med kommatecken(,) medan nycklar och värden separeras med kolon(:).

Exempel på text formaterad enligt JSON:

```
{"id" : 32,
"namn" : "Anders Andersson",
"position" : {
    "latitud" : 57.6891,
    "longitud" : 11.9787
    },
 "husdjur" : ["Fido", "Misse"]
}
```

3.4 Schemaläggning

Schemaläggning är ett brett uttryck som kan appliceras inom många olika områden. Det kan handla om allt från att planera ett skolschema för en lågstadieklass till att schemalägga flygvärdinnor till ett internationellt flygbolag. I teorin är schemaläggning ett komplext problem då man måste utvärdera alla potentiella scheman för att kunna dra slutsatser om vilket som är optimalt. Vid schemaläggning som tar hänsyn till väldigt många parametrar blir det svårare och svårare för människan att ta beslut som generar ett bra schema. Datorer är därför ett utmärkt verktyg för beräkningstunga operationer som dessa. *Nurse Scheduling Problem* (NSP) eller *Nurse Rostering Problem* som det kallas av vissa är en tillämpning av schemaläggning som det bedrivs forskningsprojekt kring inom datavetenskapen [9]. Detta problem handlar om att schemalägga sjuksköterskor på ett sjukhus på ett optimalt sätt givet vissa krav. Hårda krav, som att en sjuksköterska inte ska schemaläggas att vara på två ställen samtidigt, måste vara uppfyllda för att ett scheman ska anses vara giltigt. Mjuka krav, som att en sjuksköterska inte ska tilldelas två pass som är tätt inpå varandra, uppfylls i den mån det är möjligt. Det optimala schemat kan då definieras som det schema som inte bara är giltigt utan också uppfyller flest antalet mjuka krav

[9]. Sökningen efter det optimala schemat bör struktureras på ett smart sätt då det är orimligt att söka genom alla möjliga scheman och jämföra dessa. Det finns en stor mängd vetenskapliga publikationer innehållandes teorier om hur man på bästa sätt löser NSP [3, 10, 11].

Ett sätt att lösa NSP är att representera problemet som ett *Satisfiability Problem* (SAT) och använda en SAT-lösare för att lösa det [10]. En SAT-lösare utvärderar om variablerna i ett booleanskt uttryck kan sättas på ett sätt så att hela uttrycket blir sant. Till exempel blir följande uttryck sant om och endast om $a = sant$, $b = sant$ och $c = falskt$:

$$a \wedge b \wedge \neg c$$

För detta uttryck finns det bara en möjligt lösning. Det är dock inte något som gäller för SAT i allmänhet. Ett sätt att tillämpa SAT på NSP är att använda variablerna för att uttrycka huruvida en sjuksköterska ska arbeta ett pass eller inte [10]. Exempelvis skulle värdet på den booleanska variabeln X_{sbd} indikera huruvida sköterska s ska jobba på block b dag d eller ej. Med hjälp av dessa kan uttryck skrivas baserat på vilka krav som ställs på det schema som ska genereras. Ett exempel på uttryck skulle kunna vara:

$$(X_{111} \vee X_{211}) \wedge (\neg X_{111} \vee \neg X_{211})$$

Detta uttryck säkerställer att endast en av variablerna X_{111} och X_{211} får vara sann, vilket i praktiken innebär att endast en av de två sjuksköterskorna får arbeta det första passet den första dagen.

3.5 AngularJS

AngularJS är ett JavaScript-ramverk för att skapa webbapplikationer [12]. AngularJS bygger på MVC-strukturen som står för "Model View Controller" och är ett designmönster för att strukturera mjukvara [13]. Applikationer byggda i AngularJS består ofta av flera huvudvyer där varje vy har skapats med varsin kontroller, vy och modell. Vyn består av en eller flera HTML-filer som bestämmer vad och vart saker ska visas, samt några CSS-filer som ger sidan stil och färg. Modellen består av den data som ska visas och den kan antingen lagras direkt i applikationen eller hämtas från exempelvis en databas. Kontrollern består av en eller flera JavaScript-filer som kopplar samman modellen och vyn med nödvändiga funktioner och eventuell importering av externa bibliotek.

3.6 Android

Android är ett operativsystem för mobiltelefoner och surfplattor och är ett av de två operativsystem som den mobila Eciton-applikationen utvecklas till. Systemet baseras på Linux-kärnan och utvecklas av Google [14], som publicerar Androids källkod under open-source licenser [15]. Applikationer utvecklas både i programmeringsspråket Java och i märkspråket XML, där Java används för applikationens klasser och gränssnitt, medan XML används främst för resurser som exempelvis layout-element, textsträngar eller konstanter. Ett flertal olika så kallade integrerade utvecklingsmiljöer (IDE) kan användas för Androidutvecklingen utöver den officiella IDE:n *Android Studio* [16], som innehåller bland annat en simulator för olika Android-versioner, samt ett stort antal verktyg.

3.7 iOS

Operativsystemet som används för att driva Apples surfplattor och mobiltelefoner heter iOS, och är det andra operativsystemet som Eciton-appen utvecklas till. För utveckling i iOS används deras egen IDE *XCode* [17], som finns tillgängligt för alla Mac OSX-användare. Den innehåller ett antal standardbibliotek som kan behövas för utvecklingen samt verktyg för verifiering, exempelvis iPhone-simulatorer. Valet av programmeringsspråk står mellan Objective-C, som är en variant av det klassiska programmeringsspråket C, och Apples eget programmeringsspråk Swift. Swift är det nyare språket av de två men det fungerar felfritt tillsammans med gammal Objective-C kod.

4

Genomförande

När man utvecklar en produkt som denna är det viktigt att det inte slutar med en produkt som löser ett helt annat problem. I detta fallet kan det vara lätt att fokusera för mycket på den initialt givna produktspecifikationen. Den bestod av ett antal krav på det system som skulle utvecklas. Ofta är fallet att beställaren inte vet vad den vill ha och att den är fast i ett tankesätt om hur den brukar arbeta. Det är därför viktigt att försöka sätta sig in i det verkligt övergripande problemet. I detta fall är problemet att artister behöver vara på olika platser vid olika tidpunkter och de behöver transportera sig där emellan på ett så smidigt och kostnads- och resurseffektivtsätt som möjligt, och det är det problemet som behöver lösas. I detta projekt sker det genom att skapa ett IT-system som förhoppningsvis är bättre än det som först specificerades av Elvaett.

En stor fördel med detta projekt är att det genomförs tillsammans med några av de avsedda användarna. Det ger projektgruppen som utvecklare stor möjlighet att kunna skapa en produkt som är vad användarna faktiskt vill ha och inte vad man trodde från början att de ville ha. Då användarna får testa produkten med jämna intervall upptäcks snabbt hur produkten borde användas. Det är något som inte hade varit möjligt om projektet genomfördes utan att samarbeta med användarna på det sätt som arbetsmetodiken har möjliggjort. I det följande kapitlet beskrivs arbetsprocessen under vilken systemet implementerades.

4.1 Arbetsmetodik

I det här avsnittet beskrivs gruppens tillvägagångssätt under projektets förlopp. Det diskuteras hur arbetet fördelades, vilken utvecklingsmetodik som användes samt vilket sätt gruppen valde för att synkronisera arbetet.

4.1.1 Arbetsgrupper

Med ett stort system med många delar var det naturligt att utveckla det i arbetsgrupper. Genom att dela upp arbetet i arbetsgrupper á två eller tre personer kunde systemet utvecklas snabbt men även med återkoppling från de andra utvecklarna i

projektgruppen. Arbetsgrupperna skapade diskussion och ett samarbete där frågor lyfts och besvarades fort. Några av arbetsgrupperna under projektet var Android, schemaläggare och webbapplikation. Värt att notera är att medlemmarna i arbetsgrupperna växlade under arbetets gång.

4.1.2 Agilt utvecklande

En traditionell metod som ingenjörer arbetar efter är den så kallade vattenfallsmodellen där det först planeraras och designas, sedan implementeras och testas, för att till sist driftsätta det som utvecklats. När mjukvara ska utvecklas visar det sig dock att det är väldigt svårt att planera allt från början. Istället utförs arbetet iterativt. Detta brukar kallas ett agilt utvecklingssätt. Det här systemet utvecklades med en version av detta som kallas Scrum [18]. I Scrum finns det olika roller och ett antal beståndsdelar vars mest väsentliga egenskaper beskrivs nedan.

Det börjar med att det finns en lista med alla önskvärda funktioner. Av dessa väljs några som ska implementeras i nästa programversion. De önskvärda funktionerna för det här systemet kommer från den problembeskriving som gavs av förslagslämnaren Elvaett.

De funktioner som valts ut för implementering i en programversion prioriteras, tidsestimeras och delas ut på så kallade sprints. En sprint är en kortare tid, vanligtvis en eller ett par veckor, där det implementeras några av de funktioner som ska finnas med i programversionen. Det planeras alltid en sprint i taget och den utvärderas efter att tiden gått ut. Tanken är att efter varje avslutad sprint så ska det finnas en körbar version av systemet. Funktionerna som implementeras kan vara en enkel version som är menad att byggas vidare på senare men de måste fungera.

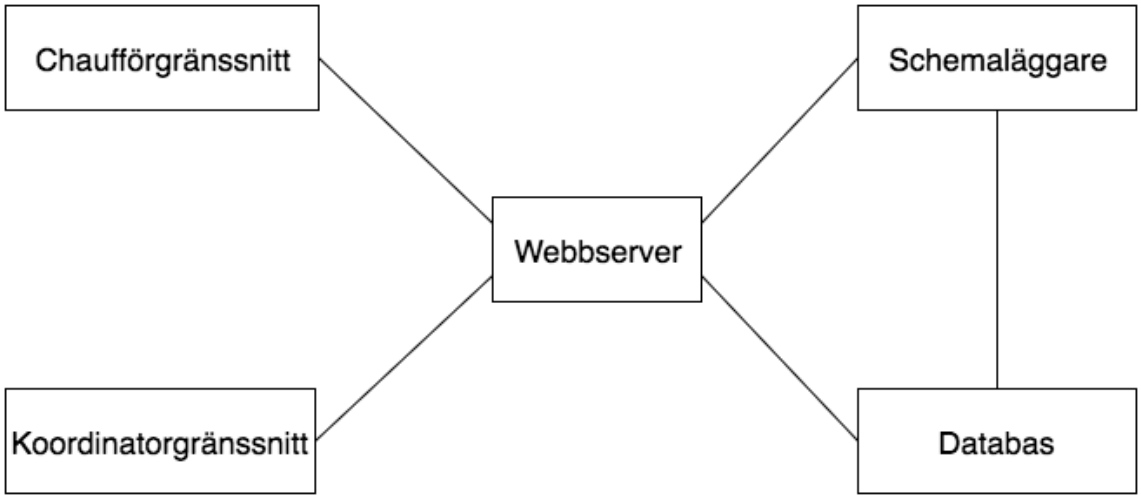
4.1.3 Versionshantering

Då flera arbetsgrupper utvecklade olika systemmoduler parallellt krävdes en central plats där all kod kunde samlas och synkroniseras. För detta valdes den webbbaserade tjänsten *Github* [19], som använder sig av det distribuerade versionshanteringssystemet *Git*. Med över 14 miljoner användare och över 35 miljoner mjukvaruprojekt är Github det populäraste webbhotellet för lagring av källkod. Versionshanteringssystemet Git låter användare skapa grenar så att utvecklandet av flera moduler kan ske parallellt. Det sparar också en historik över alla ändringar. Att Git är ett distribuerat system innefattar att gruppens samtliga medlemmar har sin egen kopia av projektet och projektet är därmed skyddat mot ett potentiellt bortfall av Github respektive dess webbhotell.

Varje arbetsgrupps utveckling skedde i arbetsgruppens gren på Github. Dessa grenar sammanfördes till den så kallade *master-grenen* med regelbundna mellanrum.

4.2 Systemöversikt

För att kunna lösa problemet behövdes flera olika delar som samarbetade. Ett system med 5 olika delar skapades där alla delarna hade olika ansvar. Systemet består av en databas, en schemaläggare, ett koordinatorgränssnitt, ett chaufförsgränssnitt och en server (se Figur 4.1).



Figur 4.1: Översikt av systemet och dess kommunikationskanaler

- Databasen har som funktion att permanent lagra data som kan användas av de andra delarna av systemet.
- Schemaläggarens uppgift är att lösa schemaläggningsproblemet.
- Koordinatorgränssnittets funktion är att presentera information samt ge tillgång till de funktioner som koordinatorer behöver för att använda systemet.
- Chaufförsgränssnittets funktion är att presentera information för en chaufför om de körningar denne ska utföra samt låta chauffören skicka några olika bekräftelsemeddelanden till servern.
- Serverns funktion är att skapa ett API för användargränssnitten så att de kan interagera med schemaläggaren och databasen.

4.3 Databas

Datan i databasen behöver kunna nås av samtliga moduler och därmed spelar databasen en central roll i systemet. Därför valdes och implementerades en databas väldigt tidigt. Att välja en relationsdatabashanterare var naturligt eftersom databasen skulle lagra mycket relaterad information, till exempel vilken chaufför och

vilken bil som är kopplade till vilken transport. I projektet användes den fria databashanteraren *MySQL* [20], då det finns mycket stöd och bra dokumentation för denna.

Då databasen implementerades i början av projektet kunde alla arbetsgrupper nyttja den i ett tidigt stadium. Det medförde dock att den första implementationen bara innehöll en grundläggande struktur och framtida utvecklingar var delvis svåra att förutse. Eftersom schemaläggningsalgoritmen utvecklades stegvis och mobil-applikationerna utvidgades efter återkoppling från Elvaett, behövde databasen ändras och anpassas kontinuerligt. De nödvändiga nya tabellerna, attributen och relationerna skapades följaktligen utifrån de olika behov som uppstod i schemaläggaren och mobil-applikationerna.

4.4 Server

Servern implementerades som en webbserver och använder sig av HTTP och WebSocket protokollen för kommunikation. HTTP valdes då det kan användas på alla de plattformar som ska interagera med servern. HTTP har en dock en del begränsningar så WebSocket-protokollet användes också för att ge utökad funktionalitet såsom tvåvägskommunikation.

Servern tillåter anslutna klienter att interagera med data i databasen. Vilken data som klienter kan få tillgång till beror på vilken roll användaren har. Till exempel har en chaufför endast tillgång till de körningar som den själv ska utföra medan en koordinator har tillgång till samtliga chaufförers körningar.

För att kunna kontrollera vem som startar schemaläggningsprocessen sker all interaktion med schemaläggaren via webbservern. Det gjorde också att alla funktioner som klienterna kan använda finns samlade på ett ställe.

Ramverket Sails valdes för utvecklingen av webbservern. Det valdes då det hade stöd för alla de funktioner som behövdes vid projektets start. Sails [21] kan kommunicera med MySQL-databaser, hantera WebSocket-anslutningar samt sammankopplas med de grafiska gränssnitt som systemet använder. Det har även andra funktioner som gjorde utvecklingen lättare, bland annat återanvändbara säkerhetsregler.

4.5 Schemaläggning

I planerandet av en operation förekommer det så pass situationsspecifika krav att vissa delar av planen måste utformas manuellt av koordinatorn. Majoriteten av operationsplanen är däremot av sådan karaktär att koordinatorn kan låta systemet generera den automatiskt. I denna sektion beskrivs utformandet av den automatiska

schemaläggaren, det vill säga den del av systemet som på algoritmisk väg genererar en plan utifrån givna behov som koordinatorn har matat in i systemet.

4.5.1 Olika faser i schemaläggningsprocessen

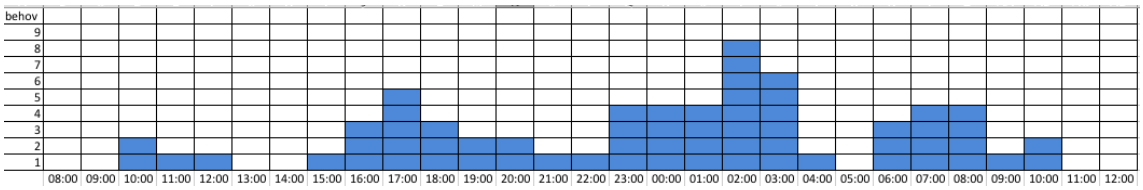
Vid intervjuer med Elvaett framkom att det huvudsakligen finns tre olika schemaläggningar som systemet skulle kunna göra. De tre olika faserna, som måste göras vid olika tidpunkter i planeringsarbetet, är följande.

- Allokering av bilar och skapande av arbetspass utifrån uppskattade resursbehov.
- Schemaläggning av chaufförer på arbetspass utifrån en mängd av anställda chaufförer och en mängd av skapade arbetspass.
- Planering av vilken bemannad bil som ska genomföra varje körning.

Dessa specificeras i sin helhet i avsnitt 2.1 och implementeringen utgår i huvudsak från den specifikationen.

4.5.2 Skapande av arbetspass

Indatan till schemaläggaren i den här fasen består av en specifikation av antalet bilar i varje bilkategori som behöver vara bemannade varje festivaltimma. En giltig uppsättning arbetspass tillfredsställer samtliga sådana behov. Om alla arbetspass skulle ha samma längd hade problemet gått att lösa med en girig algoritm genom att helt enkelt gå igenom behoven från första till sista timmen och sätta ut arbetspass så fort den stöter på ett otillfredsställt behov tills dess att alla behov är tillfredsställda. Men det faktum att arbetspass kan vara antingen åtta eller tolv timmar långa gör att en mer avancerad algoritm behövs. Nedan beskrivs den algoritm som utformades för detta.



Figur 4.2: Ett exempelhistogram som visualiserar behovet av bemannade bilar varje timma för en viss bilkategori under en kort festival.

Om man tänker sig behovet för en viss bilkategori som ett histogram (se exempel i Figur 4.2) börjar algoritmen vid den första timmen till vänster i histogrammet. Om höjden av stapeln är ≤ 0 är denna timme tillfredsställd och algoritmen går vidare och behandlar nästa timme. Om höjden av stapeln är > 0 är behovet för timmen i fråga ännu inte tillfredsställt. Då skapar algoritmen två nya instanser av problemet. I den ena nya instansen lägger den till ett arbetspass som startar

vid timmen ifråga och löper åtta timmar framåt. I den andra nya instansen görs motsvarande men passet som läggs till fortsätter istället i tolv timmar framåt, givet att antalet utplacerade tolvtimmarspass i instansen inte överstiger det specificerade maximala antalet tolvtimmarspass som koordinatorn har angivit. Om maxantalet tolvtimmarspass överskrids stryks denna instans och algoritmen går bara vidare med åttatimmars-instansen. När ett pass läggs till i en instans subtraheras höjden av staplarna i histogrammet för alla timmar som passet täcker med ett. Algoritmen går sedan rekursivt vidare och fortsätter lösa varje ny instans av problemet på samma sätt.

Det bildas ett binärträd av instanser, där varje blad i trädet är en instans med en uppsättning av arbetspass som täcker alla behov i det ursprungliga histogrammet. Samtliga blad-instanser är således en giltig lösning av problemet. För varje giltig lösning skapas nu dummy-bilar som tilldelas alla arbetspass i lösningen. Minsta möjliga antal dummy-bilar skapas givet att ingen bil får ha mer än ett arbetspass samtidigt vid något givet tillfälle.

För att välja ut den bästa lösningen av de giltiga lösningarna beräknas antalet bilar som krävs för varje lösning och den lösning som kräver färst bilar väljs ut. Om flera lösningar kräver samma antal bilar beräknas det totala antalet arbetstimmar i var och en av dessa och den av dem som kräver färst arbetstimmar väljs ut. Om flera av dessa lösningar kräver samma antal arbetstimmar väljs den av dem som har störst antal tolvtimmarspass. Prioriteringen av dessa optimeringsparametrar överrenstämmer med specifikationen i avsnitt 2.1. När den bästa lösningen valts ut skrivs arbetspassen och dummy-bilarna i den till databasen.

När algoritmen var implementerad och testkördes kunde det konstateras att den behövde arbeta väldigt länge om de inmatade resursbehoven var i storleksordning motsvarande en större festival. Det beror på att binärträdets storlek, och därmed tidskomplexiteten för algoritmen, ökar exponentiellt som funktion av antalet arbetspass som behövs. För att minska tiden det tar att köra algoritmen implementerades en funktionalitet som delar upp histogrammet i ett histogram per dygn och kör algoritmen på ett sådant mindre histogram i taget. Innan ett sådant histogram matas in hanteras eventuella överlappande pass från förra körningen. Detta gjorde att optimeringen inte fungerar fullt ut omkring dygnsövergångar, men förbättrade körtiden radikalt.

4.5.3 Schemaläggning av chaufförer

Enligt specifikationen ska programmet som automatiskt schemalägger chaufförer på arbetspass köras efter att arbetspassen har skapats samt efter att koordinatorn har matat in de anlitade förarna i systemet. Det implementerade programmet börjar därför med att läsa in dessa data från databasen.

Problemet som programmet ska lösa här, att para ihop arbetare med arbetspass,

kan liknas vid det välkända beslutsproblemet Nurse scheduling problem (NSP) som beskrivs i avsnitt 3.4. Det finns en mängd av regler som ett schema ska uppfylla för att vara en giltig lösning, såsom att alla pass ska vara bemannade, att en arbetare inte kan bli tilldelad två pass som överlappar varandra, att en arbetare max får ha ett pass per dag, med mera. Eftersom ett problem av den här typen är väldigt komplext och svårt att utforma en lösningsalgoritm för (se stycket om NSP i avsnitt 3.4) bestod det inledande arbetet av att hitta en väletablerad lösningsmetod som kunde appliceras på det här systemets specifika version av problemet. Efter att ha övervägt ett flertal olika lösningsmetoder beslutades att använda en SAT-lösare. Att använda SAT-lösare är ett effektivt sätt att lösa NSP-relaterade problem och det bedömdes att en sådan lösningsmetod skulle vara behändig för gruppen att implementera i detta system.

För att kunna lösa schemaläggningsproblemet med en SAT-lösare behövde problemet modelleras i form av booleska variabler och uttryck. Som variabler definierades alla möjliga par av en chaufför f och ett arbetspass a. En sådan variabel $x_i = (f_p, a_q)$ kan ha värdet *sant* eller *falskt*. Om varibeln x_i har värdet *sant* betyder det att chauffören f_p ska arbeta på arbetspasset a_q. Om x_i har värdet *falskt* ska chauffören f_p inte arbeta på arbetspasset a_q. Reglerna modelleras som booleska uttryck på konjunktiv normalform. Exempelvis regeln att varje pass i $[a_1...a_n]$ måste vara bemannat av någon chaufför i $[f_1...f_m]$ översätts till uttrycket

$$((f_1, p_1) \vee (f_2, p_1) \vee (f_3, p_1) \vee ... \vee (f_m, p_1))$$

$$\wedge((f_1, p_2) \vee (f_2, p_2) \vee ... \vee (f_m, p_2))$$

$$\wedge... \wedge ((f_1, p_n) \vee (f_2, p_n) \vee ... \vee (f_m, p_n))$$

Uttrycken för samtliga regler sätts ihop till ett enda stort uttryck med $\wedge$ emellan eftersom samtliga regler måste följas för att ett schema ska vara giltigt. Därefter skickas uttrycket in i SAT-lösaren och den svarar med en uppsättning variabelvärden som gör att hela det stora uttrycket evalueras till *sant* om en sådan uppsättning variablevärden finns. Reglerna implementerades enligt de specificerade kraven i avsnitt 2.1. En översättningsfunktionalitet implementerades som minns hur variablerna definierades innan de skickades in i SAT-lösaren för att sedan kunna översätta lösningen i form av tilldelade variabelvärden till data som säger vilken chaufför som ska arbeta vilka arbetspass. Denna data skrivs sedan till databasen.

Ingen optimering över förarnas önskemål implementerades då det ansågs att prototypsystemet kan fungera utan sådan funktionalitet och att annat var mer prioriterat att lägga arbetstid på.

4.5.4 Tilldelning av körningar till bilar

Att skapa ett program som bestämmer vilken bemannad bil som ska utföra varje körningsuppdrag på det vis som Ecitons schemaläggare ska göra är en till synes

relativt outforskad domän. Projektgruppen lyckades inte hitta någon dokumentation av något problemlösningsarbete som tagit sig an något likt det som skulle göras.

För att modellera problemet testades en mängd olika angreppspunkter. En utgångspunkt som länge verkade lovande, tills den visade sig bli väldigt komplicerad och ohanterligt, var att representera körningarna som någon form av graf. Istället valdes till slut en modell i form av booleska variabler och uttryck. Förmodligen var det inspiration från lösningen i föregående del av schemaläggaren som ledde fram till denna insikt.

Eftersom problemet i grunden handlar om att para ihop bilar med körningar definierades som grundläggande variabler alla möjliga par av en bil b och en körning k. En sådan variabel $y_i = (b_p, k_q)$ kan ha värdet *sant* eller *falskt*. Om varibeln y_i har värdet *sant* betyder det att bilen b_p har fått i uppdrag att utföra körningen k_q. Om y_i har värdet *falskt* ska bilen b_p inte utföra körningen k_q.

Oavsett hur problemet skulle lösas krävdes det att det definierades en funktion $KanUtföra(b_p, k_a)$ som bestämmer huruvida bilen b_p uppfyller alla villkor som krävs för att få utföra k_a eller ej. Dessa villkor är:

- Bilen b_p måste tillhöra en bilkategori som koordinatorn har definierat som tillräcklig för att få utföra körningen k_q.
- Bilen måste vara bemannad under hela den tidsperiod som körningen väntas pågå.

Utifrån den data som antas finnas i databasen vid körningen av detta program går det att härleda den information som behövs för att beräkna värdet på $KanUtföra(y_i)$ för alla tänkbara y_i. Detsamma gäller nästa funktion som behövde definieras. En funktion $KanKombineras(k_a, k_b)$ som utifrån vilket par av två körningar som helst bestämmer huruvida samma bil kan utföra båda körningarna eller ej. Om följande villkor är uppfyllt kan samma bil utföra både körning k_a och körning k_b.

- Antag att starttiden för k_a, t_{start_a}, inträffar tidigare än starttiden för k_b, t_{start_b}. Antag också att körtiden t_k mellan den geografiska slutpositionen för k_a och upphämtningspositionen för k_b är definierad av koordinatorn. Samma bil får utföra både k_a och k_b om och endast om sluttiden för k_a, t_{slut_a}, inträffar före $t_{start_b} - t_k$. Med andra ord måste bilen hinna köra från slutpositionen för den körning som inträffar först och vara framme på startpositionen för nästa körning innan nästa körning ska börja.

Det finns också några mer grundläggande krav, nämligen följande.

- Varje körning ska tilldelas en och endast en bil.
- Körningar som är låsta till en bil ska tilldelas den bil som de är låsta till.

Eftersom funktionerna $KanUtföra()$ och $KanKombineras()$ gick att implementera kunde alla villkor beskrivna ovan implementeras som booleska uttryck. Eftersom

samtliga villkor ska vara uppfyllda för att planen ska vara giltig binds dessa booleska uttryck samman med $\wedge$ till ett stort uttryck. Detta uttryck matas sedan in i en SAT-lösare som undersöker om det finns någon uppsättning varibelvärden som gör uttrycket sant.

För att optimera schemat behövs ett stort antal giltiga scheman att försöka välja ut det bästa av. För att skapa många olika giltiga scheman körs SAT-lösaren många gånger. För att den inte ska generera samma schema flera gånger negeras uttrycket för varje giltigt schema som genereras och läggs till som en term i det uttryck som bli indata i nästa körning. Eftersom det är möjligt att det finns ett väldigt stort antal potentiella giltiga scheman kan koordinatorn ställa in en maxtid som begränsar hur länge SAT-lösaren får fortsätta generera scheman. En optimeringsrutin implementerades som applicerar en poäng-funktion på varje genererat giltigt schema. Poäng-funktionen mäter följande egenskaper hos schemat.

- Väntetiden mellan varje par av körningar. Om slutpositionen för den första körningen i ett par är samma som startpositionen för den andra körningen i paret ges en bonuspoäng om väntetiden är kortare än en kombo-tid som koordinatorn har definierat för den geografiska zon som positionen befinner sig i. Annars ökar poängen exponentiellt som funktion av väntetiden.
- Rankingvärdet på bilkategorin som den bil som tilldelats varje körning tillhör relativt rankingvärdet på körningens lägsta tillåtna bilkategori. En bil med högre rankingvärde ger mindre poäng än en bil med lägre rankingvärde eftersom användandet av en för bra bil anses vara ett slöseri med resurser.
- Antalet halvlåsta körningar som har tilldelats den bil som de är halvlåsta till. Om en körning tilldelas en annan bil än den bil som den är halvlåst till ges minuspoäng eftersom det är önskvärt att bryta mot så få halvlåsningar som möjligt.

De värden som beskriver dessa egenskaper multipliceras med en viktvektor som kalibrerades fram i samråd med Elvaett. Dessa olika viktade komponenter summeras sedan ihop till en skalär som blir schemats poäng. Det schema som fått högst poäng väljs ut och skrivs till databasen.

4.5.5 Programmeringsspråk för schemaläggaren

Ett krav på implementeringen av schemaläggaren var att det i framtiden ska vara lätt att hitta utvecklare som kan underhålla, utveckla, och bygga ut den. Därför ansågs det vara fördelaktigt att välja ett programmeringsspråk som många programmerare behärskar. Dessutom ansågs det fördelaktigt att välja ett programmeringsspråk som individerna i projektgruppen hade vana att arbeta med så att så mycket fokus som möjligt i implementeringsarbetet skulle kunna läggas på problemlösning och algoritm-design. Mot den bakgrunden beslutades att implementera schemaläggaren

med programmeringsspråket Java.

4.5.6 Val av SAT-lösare

Det finns ett flertal olika implementerade SAT-lösare på marknaden. SAT-lösaren SAT4J är implementerad i programmeringspåket Java. Den arbetar inte lika snabbt som en SAT-lösare implementerad i exempelvis programmeringsspråket C, men den har fördelen att den är lätt att interagera med från den implementerade schemaläggaren eftersom de båda är implementerade med samma programmeringsspråk. Det beslutades att använda SAT4J tillsvidare och byta ut den mot en snabbare SAT-lösare om det senare skulle visa sig att SAT-lösarens snabbhet var en kritisk faktor för hur systemet skulle uppfylla kraven som stipulerats i problemspecifikationen. Under projektet beslutades inte att byta SAT-lösare.

4.6 Webbapplikation

För att interagera med systemet som administratör eller koordinator skapades en webbapplikation. Den byggdes med AngularJS och består av några huvudvyer och en service som hämtar data från servern. Det används också några externa bibliotek och det som används främst är Moment.js och Angular Material [22] [23]. Moment.js är ett bibliotek som hjälper till med att hantera datum och tider medan Angular Material har massor av komponenter från Googles Material design som kan användas.

4.7 Android & iOS

I mobilmarknaden dominerar Android och iOS [24]. Därför togs beslutet att utveckla mobilapplikationen för båda dessa plattformar för att inte utesluta en stor användargrupp.

Ett väsentligt mål i utvecklingen var att erbjuda samma grundfunktionalitet för båda plattformarna, men inte nödvändigtvis skapa två identiska applikationer med avseende på användargränssnittet. Funktionaliteten och grunddesignen för huvudvyerna utvecklades gemensamt, men för att förhöja användarupplevelsen samt skapa en bekant känsla för användaren följer Android-applikationen Googles Material Design [25], medan designen av det grafiska gränssnittet i iOS-applikationen inspirerades av designmönstren från Apples egna applikationer [26].

Båda applikationerna består av olika vyer innehållandes element som användaren kan interagera med. Det visuella representeras i Android med hjälp av XML-filer (se även avsnitt 3.6) och i iOS med hjälp av en karta bestående av *storyboards* med

relationer till varandra. För iOS-applikationen valdes programmeringsspråket *Swift*, eftersom det gör koden mer lättläst och syntaxen liknar Javas, som används för Android-applikationer.

Ytterligare ett viktigt beslut angående designen är kravet på den minsta Android- respektive iOS-versionen. Det beslutades att kräva minst API-level 15 för Android-telefoner, vilket korresponderar med Android 4.0.3, respektive minst iOS version 9.0. Android 4.0.3 körs på över 97% av alla Android-telefoner [27]. iOS 9.0 stöds av iPhone 4S och nyare telefoner [28] och används av över 93% av iPhone-användarna [29].

Ett centralt element i applikationen är kommunikationen med projektets server. Denna kommunikation sker genom att skicka HTTP-förfrågningar till servern, som i sin tur hanterar dessa. Två typer av HTTP-förfrågningar används i applikationen: GET-förfrågningar för att hämta data från servern och POST-förfrågningar för att skicka data till servern, till exempel användarnamn och lösenord vid inloggningen. För att användaren ska kunna upprätthålla kommunikationen med servern utan att behöva identifiera sig med sitt användarnamn och lösenord vid varje förfrågan används *kakor*. Dessa är unika textsträngar som lagras lokalt på telefonen och används som substitut för användarnamnet och lösenordet. Detta innebär att POST-förfrågan endast behöver skickas en gång.

För att kunna skicka push-notiser från servern till applikationerna används Googles Cloud Messaging. GCM valdes för att tjänsten erbjuder bra stöd för både Android och iOS. När en klient registrerar sig hos tjänsten får klienten en så kallad *registrerings-token*. Denna registrerings-token identifierar klienten och skickas sedan till servern. För att skicka en push-notis till en klient skickar servern en notis med klientens token till GCM-tjänsten, som i sin tur vidarebefordrar notisen till telefonen. GCM innebär två stora fördelar för servern: Klientens operativsystem spelar ingen roll för servern, det vill säga inga anpassningar för Android eller iOS krävs, och en push-notis behöver inte skickas igen om en klient inte är åtkomlig.

5

Resultat

I detta kapitel presenteras de resultat som åstadkommits under projektets gång. Resultaten utgår från problemspecifikationen i kapitel 2 och är tillsammans en prototyp av ett system som är nära att kunna användas i praktiken. Mer om vad som saknas diskuteras i avsnitt 6.2 och 6.3.

5.1 Server & databas

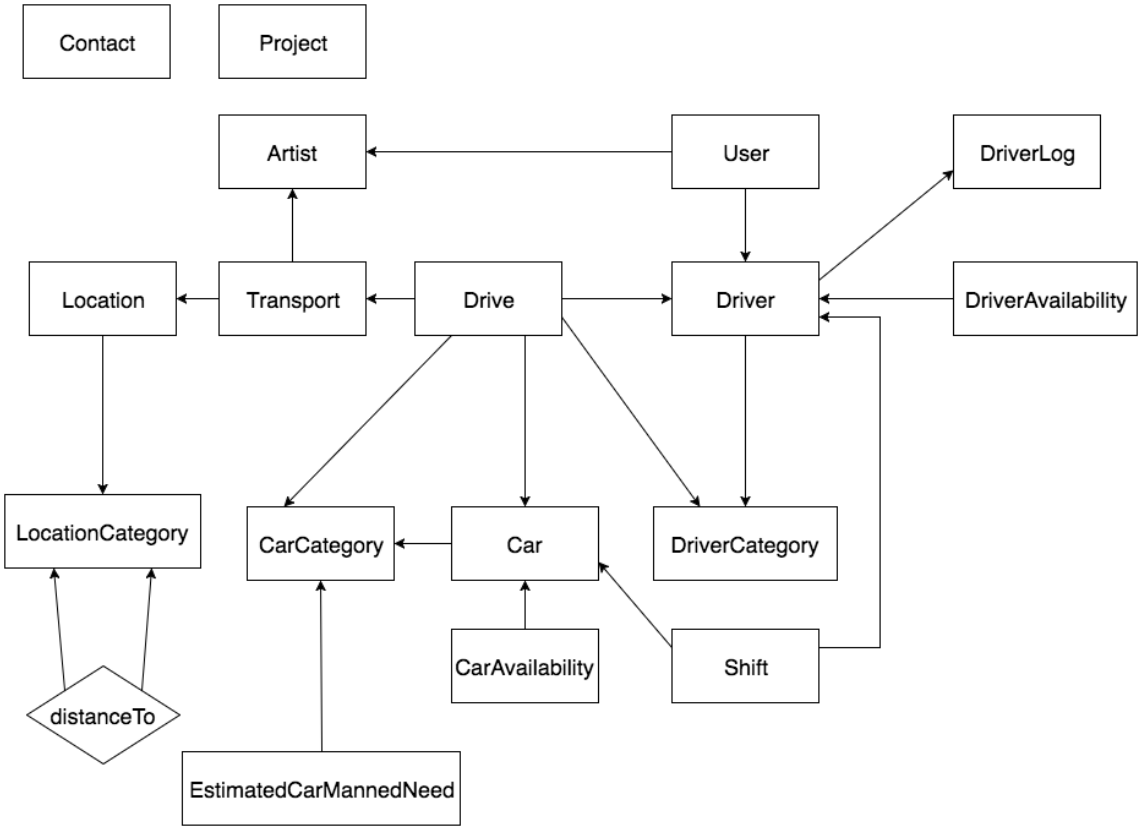
Databasen kan lagra information som schemaläggaren och användare behöver under en festival. Databasens struktur beskrivs i Figur 5.1.

Webbservern tillåter klienter att interagera med datan som är lagrad i databasen. Olika användare har tillgång till olika data och det finns två olika typer av användare: administratörer och chaufförer.

Administratörer har åtkomst till funktionerna för att hämta, skapa, uppdatera, och radera för alla typer av data som finns i databasen. När en administratör förändrar eller tar bort en post i databasen skickar servern information om förändringen till alla inloggade administratörer. Administratörer har även tillgång till schemaläggarens alla funktioner och kan starta dem via servern.

Chaufförer har endast tillgång till information om de körningar de har blivit tilldelade, kontaktuppgifter, samt information om alla platser för upphämting och avlämning.

För att tillåta koordinatorer att fortsätta sitt arbete utan internetuppkoppling kan databasen exporteras till filer som kan importeras till Excel.



Figur 5.1: UML-diagram som beskriver databasens struktur

5.2 Schemaläggaren

Schemaläggaren anropas från olika knappar i webb-gränssnittet beroende på vilken typ av schemaläggning som ska göras. Det förutsätts att datan som är nödvändig för den del av schemaläggaren som anropas finns i databasen vid anropet. Varje del av schemaläggaren börjar med att läsa in den data som den behöver från databasen, utför sedan sin schemaläggningsuppgift, och skriver till sist resultatet till databasen. I denna sektion beskrivs hur varje del av schemaläggaren arbetar för att lösa sin uppgift, hur lång tid respektive del behöver för att lösa sin uppgift, samt hur en expert med mångårig erfarenhet från branschen bedömer lösningens kvalitet.

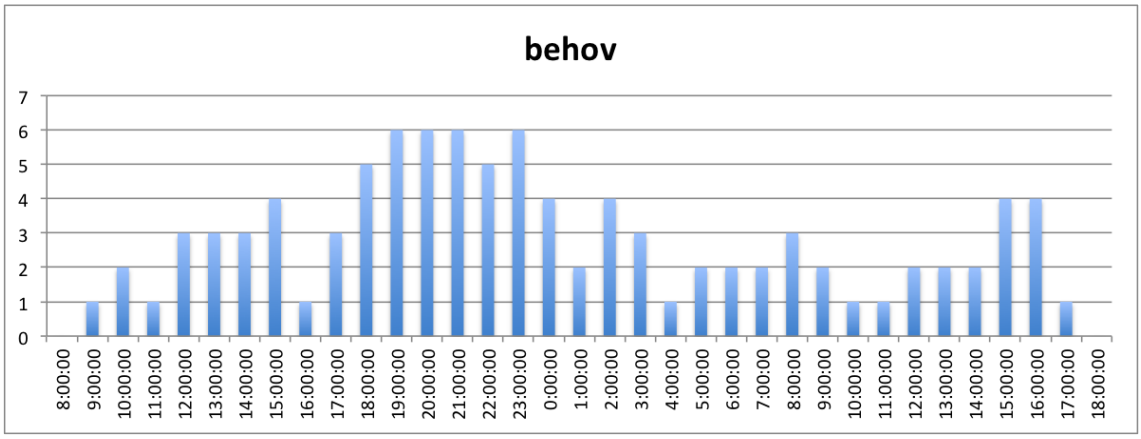
För att utvärdera exekveringstiderna har två, enligt branschexperten realistiska, festivalscenarier konstruerats. Det ena av dem, i fortsättningen benämnd som *festival1*, har en storlek och omfattning motsvarande en liten festival och det andra av dem, framöver benämnd som *festival2*, har en storlek och omfattning motsvarande en av Sveriges allra största festivaler. Datan i festival2 är delvis baserad på data från en verklig festival som ägde rum under år 2015. Exekveringstiderna har mätts upp vid körning av systemet på en bärbar dator med 1,3 GHz Intel Core i5-processor och 4 GB RAM-minne. För den kvalitativa utvärderingen utförd av en branschex-

pert har den mindre av de två testfestivalerna använts. I Tabell 5.1 beskrivs några nyckelegenskaper hos den data som koordinatorn matar in i systemet för de två testfestivalerna. Figur 5.2 illustrerar de behov av bemannade bilar per timme som koordinatorn definierat för festival1.

egenskap	festival1	festival2
festivallängd [timmar]	34	144
antal bilkategorier	1	6
största behov*	6	25
maxantal 12h-pass**	5	5
antal körningar	29	622
antal chaufförer	11	282

Tabell 5.1: Nyckelegenskaper i indata för testfestivaler

*höjden av den högsta stapeln i ett histogram som beskriver det uppskattade behovet av antal bemannade bilar per bilkatergori för varje timme under festivalen.
**maxantal 12h-pass per bilkategori per dygn.

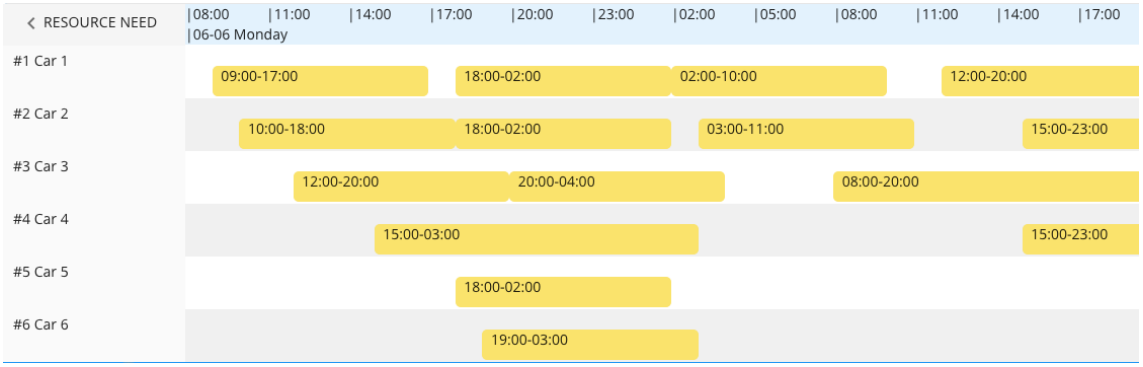


Figur 5.2: Histogram över de behov av bemannade bilar per timme som koordinatorn definierat för festival1

5.2.1 Skapande av arbetspass och allokerande av bilflotta

Den del av schemaläggaren som skapar arbetspass och allokerar bilar utifrån uppskattade resursbehov försöker tillfredsställa behoven i varje bilkategori för sig med pass som är åtta alternativt tolv timmar långa. Den placerar ut pass över otillfredsställda behov tills dess att alla behov är tillfredsställda. Varje gång den ska placera ut ett nytt pass testar den att göra en instans där det nya passet är åtta timmar långt och en instans där det nya passet är tolv timmar långt så länge som den inte använder fler än det definierade maximala antalet långa pass per dag. När detta maxantal är uppnått skapar den bara instanser där det nya passet är åtta timmar långt. Detta

leder till att den i slutändan har skapat många olika lösningar som består av olika kombinationer av åtta- och tolv-timmarspass. Den väljer sedan den bästa lösningen utifrån i första hand antalet bilar som lösningen kräver, i andra hand utifrån det totala antalet arbetstimmar som lösningen kräver, och i tredje hand utifrån hur nära lösningens antal tolvtimmarspass är det definierade maxantalet tolvtimmarspass.



Figur 5.3: Arbetspass genererade utifrån resursbehoven för festival1

Resultatet efter att schemaläggaren har skapat arbetspass och allokerat bilar för festival1 illustreras i Figur 5.3. Branschexpertens bedömning av lösningens kvalitet är att lösningen är användbar. Som koordinator hade han förfinat lösningen genom att omarrangera ett fåtal av passen under natten för att undvika att något pass börjar klockan 02:00 eller 03:00. En sådan omarrangering skulle gå att åstadkomma genom ett fåtal klick i webb-gränssnittet. Exekveringstiden som schemaläggaren behövde för att skapa arbetspass och allokera bilar för festival1 och festival2, samt några nyckelegenskaper hos lösningarna, beskrivs i Tabell 5.2.

	festival1	festival2
antal arbetspass	15	144
antal 12-h-pass	2	26
antal bilar	6	53
exekveringstid [m:s]	0:0.21	13:36

Tabell 5.2: Nyckelegenskaper i lösningar och exekveringstider, skapande av arbetspass och allokering av bilar.

Vid försök att testa modifierade varianter av festival2 där all indata varit samma förutom att maxantalet tolvtimmarspass har varierats har det noterats att exekveringstiden ökar exponentiellt som funktion av maxantalet tolvtimmarspass. Det innebär att exempelvis ett scenario där man vill att hälften av de pass som genereras för festival2, vilket skulle kunna inträffa, skulle få en orimligt lång beräkningstid.

5.2.2 Tilldelning av arbetspass till chaufförer

Den del av schemaläggaren som parar ihop chaufförer med arbetspass uttrycker en mängd regler som ett stort booleskt uttryck på konjunktiv normalform och använder en SAT-lösare för att utvärdera om det finns en lösning som gör att alla regler är uppfyllda. Om någon giltig lösning finns returneras den första giltiga lösningen som SAT-lösaren hittar. En lösning är giltig om och endast om alla arbetspass har parats ihop med en och endast en chaufför, alla chaufförer tillhör en chaufförskategori som alla bilar de parats ihop med accepterar, alla chaufförer är tillgängliga under alla sina tilldelade arbetspass, ingen chaufför som föredrar korta pass har tilldelats ett långt pass, samt ingen chaufför har tilldelats mer än ett arbetspass per dygn.

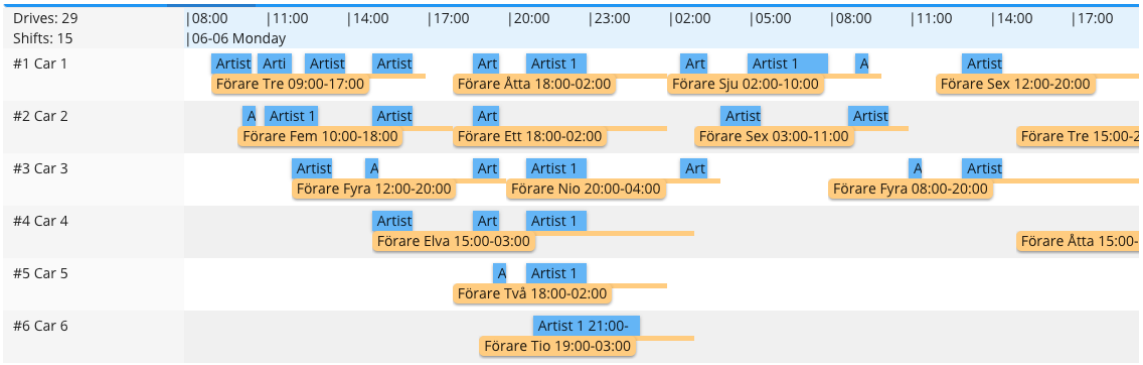
Branschexpertens bedömning av hur schemaläggaren har parat ihop arbetspass med chaufförer när den applicerats på festival1 är att lösningen är användbar. Han har inga väsentliga invändningar mot lösningens kvalitet. Exekveringstiden som schemaläggaren behövde för att para ihop chaufförer med arbetspass för festival1 och festival2 beskrivs i Tabell 5.3.

	festival1	festival2
exekveringstid [s]	1.51	44.98

Tabell 5.3: Exekveringstider, tilldelning av chaufförer till arbetspass.

5.2.3 Tilldelning av körningar till bilar

Den sista delen av schemaläggaren parar ihop bemannade bilar med körningar. Den uttrycker en mängd regler som ett stort booleskt uttryck på konjunktiv normalform och använder en SAT-lösare för att utvärdera om det finns lösningar som gör att alla regler är uppfyllda. Om den hittar någon giltig lösning fortsätter den att generera giltiga lösningar tills dess att den hittat alla giltiga lösningar alternativt tills dess att den stoppas. Koordinatorn kan definiera en maxtid efter vilken SAT-lösaren ska sluta generera lösningar om den inte har hittat alla innan dess. En lösning är giltig om och endast om alla körningar har blivit tilldelade en och endast en bil, alla bilar tillhör en chaufförskategori som alla körningar de parats ihop med accepterar, alla bilar är bemannade under alla pass som de har blivit tilldelade, ingen bil har tilldelats körningar på ett sådant vis att den inte kan hinna med att utföra samtliga sina körningar inräknat tiden det tar att köra emellan dem, samt ingen tidigare tilldelning som koordinatorn har definierat som låst har ändrats. Varje genererad giltig lösning tilldelas en viktad poängsumma utifrån längden på pauserna i varje bils körschema, antal kombinationer av körningar som tilldelats samma bil där den ena körningen slutar strax innan den andra börjar givet att den första slutar på samma geografiska plats som den andra börjar, varje bils bilkategori i relation till vad de körningar den tilldelats kräver, samt antalet halvlåsta tilldelningar som ändras. Den lösning som får störst poängsumma returneras.



Figur 5.4: Komplett schema för festival1

Resultatet efter att schemaläggaren har parat ihop bemannade bilar med körningar när den applicerats på festival1 visualiseras i Figur 5.4. När lösningen på bilden togs fram tilläts SAT-lösaren generera lösningar i tio minuter innan den stoppades. Branschexpertens utlåtande angående lösningen är att den verkar vara användbar. Han konstaterar att det är svårt att helt säkert avgöra om en lösning skulle fungera i verkligheten. I övrigt är hans väsentliga kommentarer att lösningen har god spridning och bra marginaler. Eftersom antalet giltiga lösningar är så stort att det tar orimligt lång tid att generera och poängsätta alla bedöms exekveringstiden istället genom att räkna hur många giltiga lösningar schemaläggaren hinner generera och bedöma om den får arbeta i tio minuter. Detta beskrivs i Tabell 5.4.

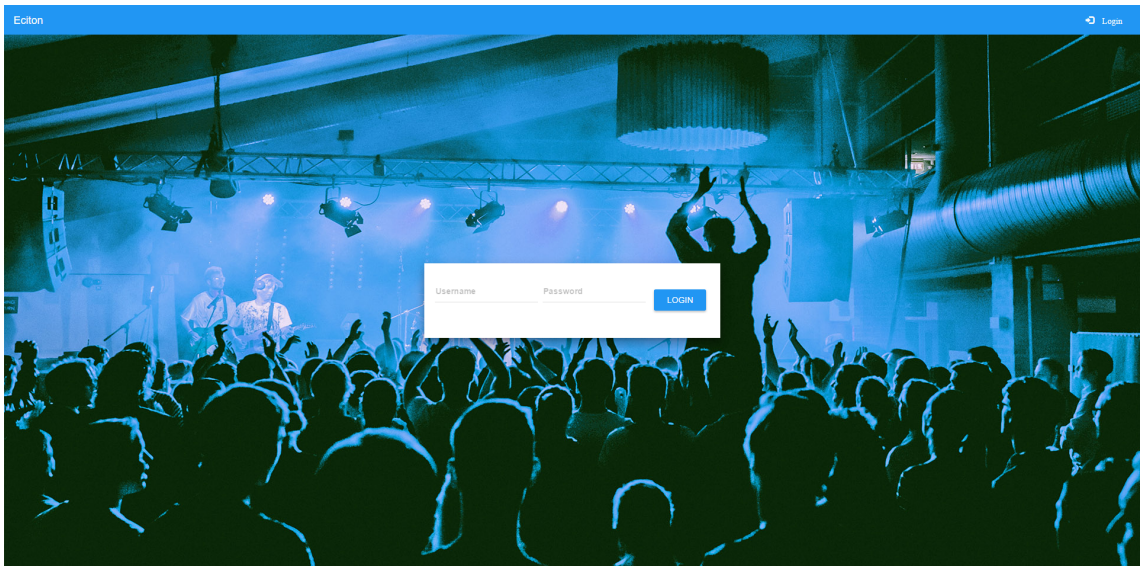
	festival1	festival2
antal	185577	20764

Tabell 5.4: Antal giltiga lösningar som hann genereras och poängsättas under tio minuter, tilldelning av körningar till bilar.

5.3 Webbapplikation

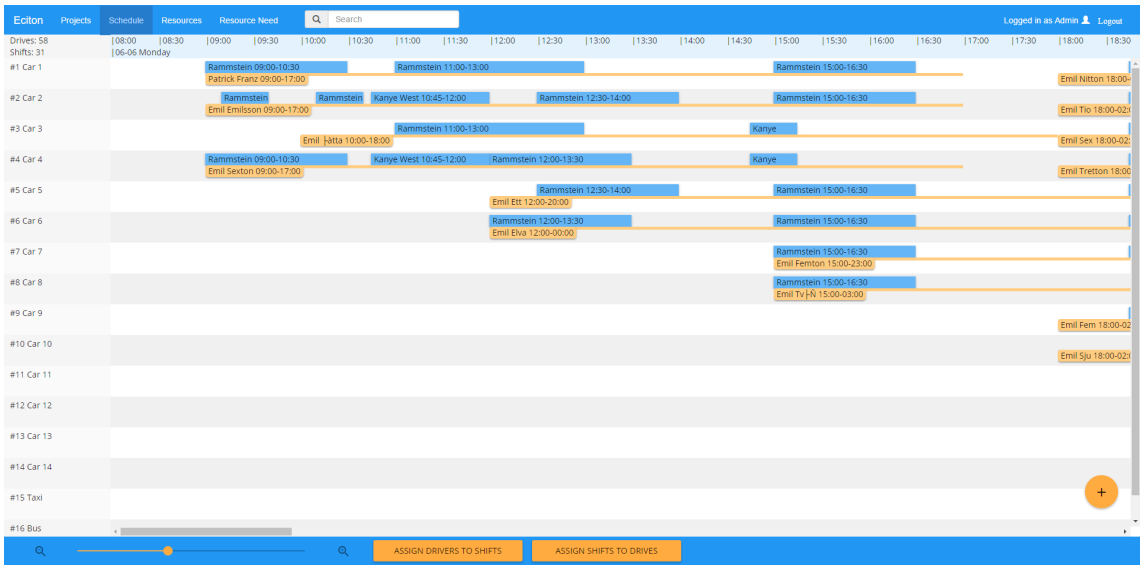
Webbapplikationen består av ett flertal vyer och gränssnitt. Det första en användare möts av är ett inloggningsformulär, se Figur 5.5. Där matas användarnamn och lösenord in för att komma åt systemet. När en användare har loggat in dyker det upp fem alternativ i huvudmenyn, *Projects*, *Schedule*, *Resources* och *Resource Need*. Genom att trycka på dessa alternativ navigeras man till applikationens huvudvyer med samma namn. I menyn finns även en sökruta, information om vem man är inloggad som och en knapp för att logga ut. Denna meny återfinns alltid högst upp i gränssnittet var man än befinner sig i webbapplikationen.

Vid inloggning befinner sig användaren i den första huvudvyn: *Projects*. I *Projects* finns samtliga projekt som användaren har tillgång till och med ett knapptryck



Figur 5.5: Inloggningsvyn i webbapplikationen

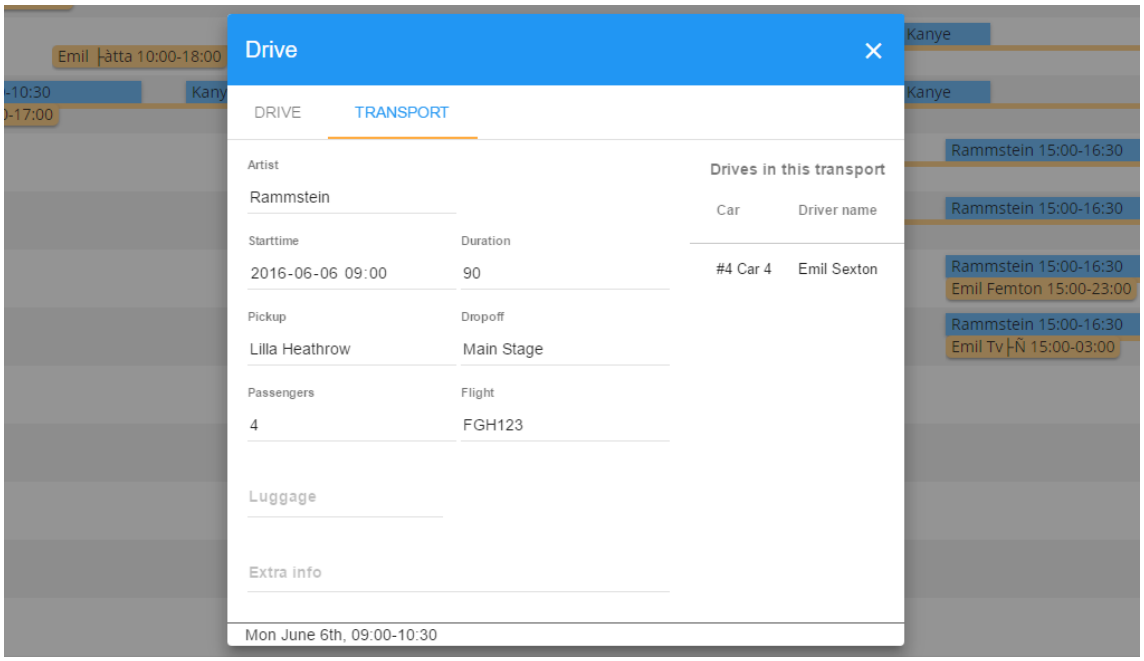
laddas all data in för det valda projektet samt navigerar applikationen till den andra huvudvyn: *Schedule*.



Figur 5.6: Schemavy

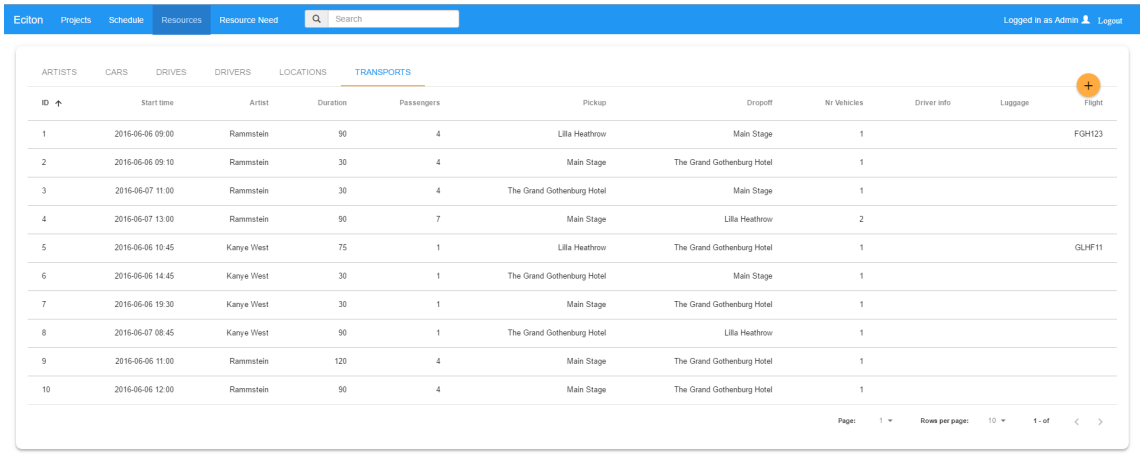
I *Schedule* återfinns schemat för det valda projektet, se Figur 5.6. Schemat består av ett skrollbart fönster samt en panel med ett reglage som justerar zoomnivån för schemat. Högst upp i det justerbara fönstret finns en tidslinje, till vänster finns en lista med alla bilar och i mitten återfinns samtliga körningar och arbetspass för projektet. Varje rad i schemat motsvarar den bil som finns i listan till vänster. På raderna finns de körningar och arbetspass som tillhör bilen på den raden. Med knapparna *Assign Drivers to Shifts* och *Assign Shifts to Drives* kan användaren

skicka förfrågningar till schemaläggaren om att tillsätta chaufförer till arbetspass respektive tillsätta arbetspass till körningar. Svaren från dessa förfrågningar visas i en dialogruta på samma sätt som i *Resource Need* som visas i Figur 5.10.



Figur 5.7: Dialogruta för en transport

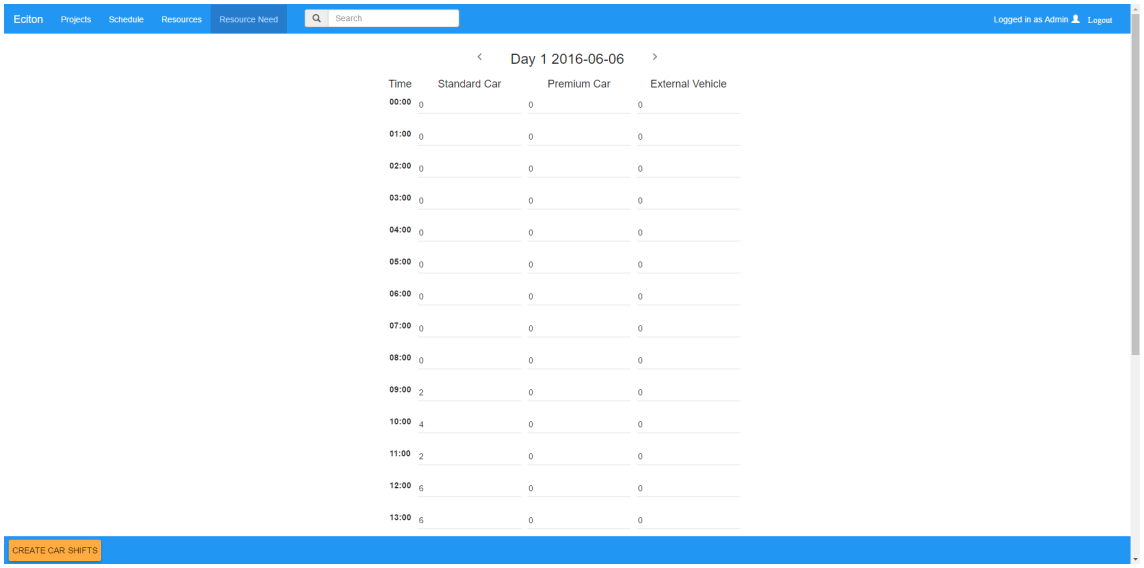
Trycker man på en körning öppnas en dialogruta med information om körningen samt en flik för den tillhörande transporten, se Figur 5.7. I transport-fliken kan användaren redigera information för transporten, exempelvis tiden då den inträffar. Till sist finns en knapp i det nedre högra hörnet. Vid tryck navigeras man till en vy där man kan lägga till en ny transport.



Figur 5.8: Resursvy

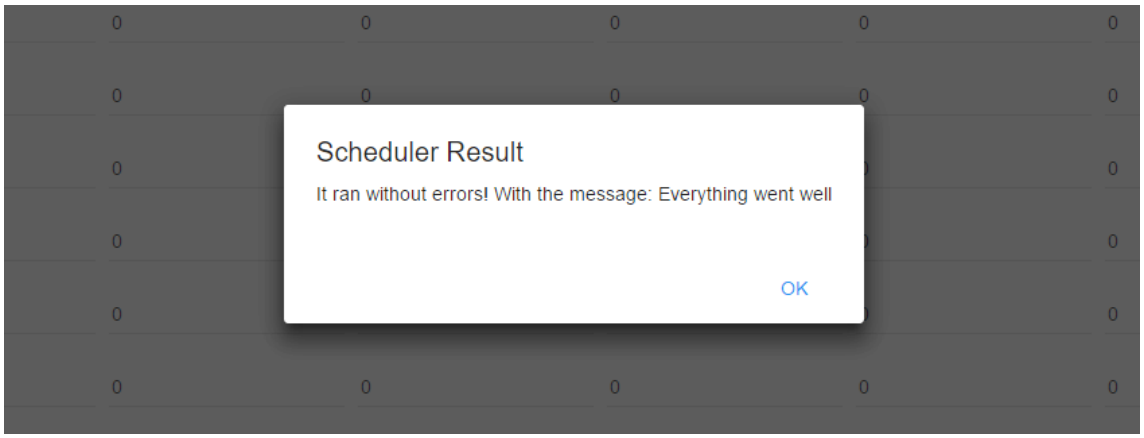
Nästa huvudvy är *Resources* (Figur 5.8). I den finns alla resurser för det valda

projektet. Detta innefattar artister, chaufförer, bilar, körningar, transporter, och platser. Samtliga resurser förutom körningar kan läggas till nya av, redigeras och tas bort. Enskilda körningar kan inte interageras med; för att ändra körningar måste transporten som de tillhör ändras.



Figur 5.9: Vy för resursbehov

Den sista huvudvyn, *Resource Need* (Figur 5.9), är en del av gränssnittet mot schemaläggaren. Med hjälp av den här vyn kan användaren mata in resursbehovet för projektet till schemaläggaren. Detta görs långt innan projektets start för att skapa arbetspassen innan några körningar existerar i schemat. Resursbehovet infogas med inmatningsfält för varje bilkategori varje timme under projektet. Efter detta har gjorts går det att trycka på *Create Car Shifts* för att skapa arbetspass baserat på resursbehovet. Schemaläggarens svar på förfrågan visas med en dialogruta, Figur 5.10.

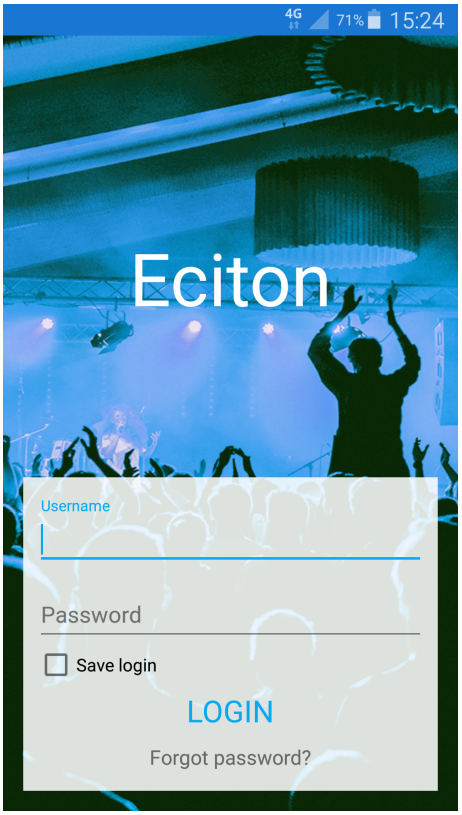


Figur 5.10: Dialogruta för svar från schemaläggaren

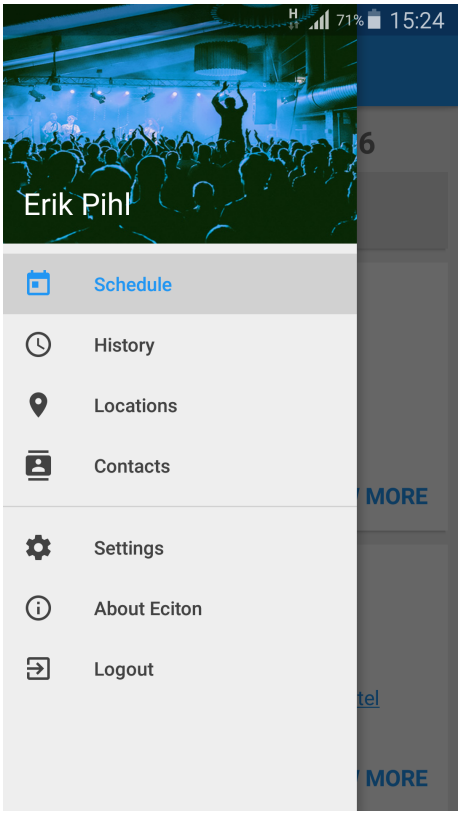
5.4 Android & iOS

Applikationen som chaufförerna använder finns i två versioner; en för Android och en för iOS. Förutom att de är för olika plattformar är meningen att de ska erbjuda samma funktionalitet. I nuläget anses Android-versionen vara en klar prototyp medan iOS-versionen fortfarande har lite implementeringsarbete som återstår innan den kan anses vara brukbar. De olika versionerna följer de designmönster som är konvention idag. Slutarbetet så som typsnitt, skuggor och hur knappar ser ut, ofta kallat för "look and feel" skiljer sig i de olika versionerna på så sätt att de är anpassade för respektive operativsystem. Android-applikationen ser ut som en Android-applikation och iOS-applikationen ser ut som en iOS-applikation.

När applikationen startas möts chauffören av en enkel inloggningsskärm, som visas i Figur 5.11. Applikationerna består av ett antal olika vyer mellan vilka man navigerar med en så kallad "Navigation drawer", se Figur 5.12.



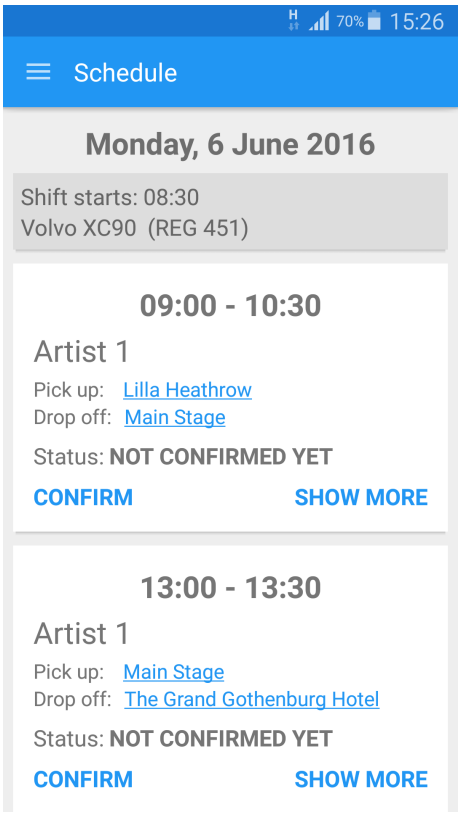
Figur 5.11: Inloggningsskärmen i Android-applikationen



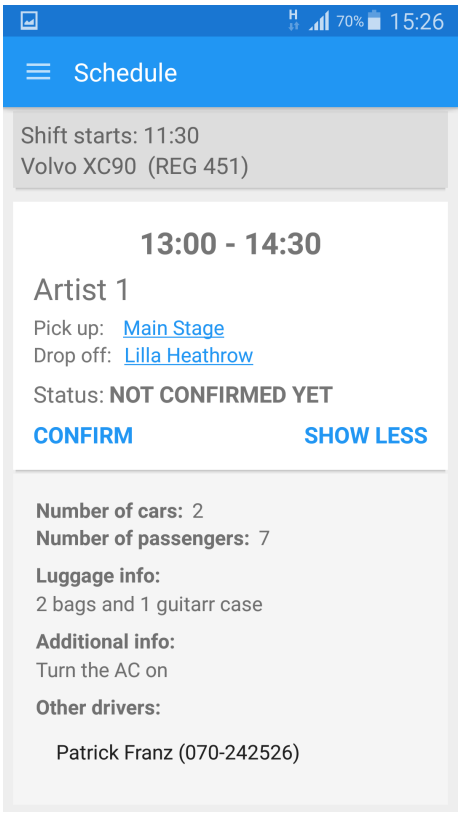
Figur 5.12: Navigation drawer i Android-applikationen

Efter inloggning presenteras vyn som kallas "Schedule". Det är den vy där chaufförerna kan se sitt schema av körningar. Varje körning representeras av ett kort där informationen som en chaufför behöver för respektive körning visas. Detta illustreras i Figur 5.13. För att få mer detaljerad information kan användaren klicka på "show

more”-knappen. Kortet expanderas då och chauffören kan bland annat se kommentarer som koordinatorerna har matat in angående körningen, samt vilka andra chaufförer som är involverade i samma transport. Detta illustreras i Figur 5.14.



Figur 5.13: Schemavyn i androidapplikationen



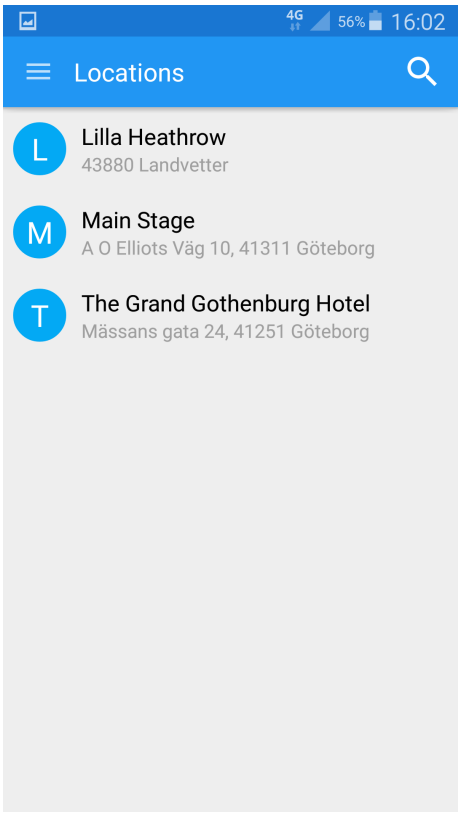
Figur 5.14: En körning där detaljerad information visas.

Körningarna ligger i en lista sorterade i kronologisk ordning med en fristående rubrik för varje dag. Det finns även information i listan för när chaufförens arbetspass börjar och slutar. Det är under den tiden som chauffören måste vara redo att arbeta, även om det inte finns planerade körningar under hela arbetspasset. Det är även där som vilken bil chaufförerna ska använda meddelas. Listan med kort är sorterad kronologiskt uppifrån och ner. Chauffören jobbar med ett kort i taget och kan se vad som kommer härnäst genom att läsa i listan.

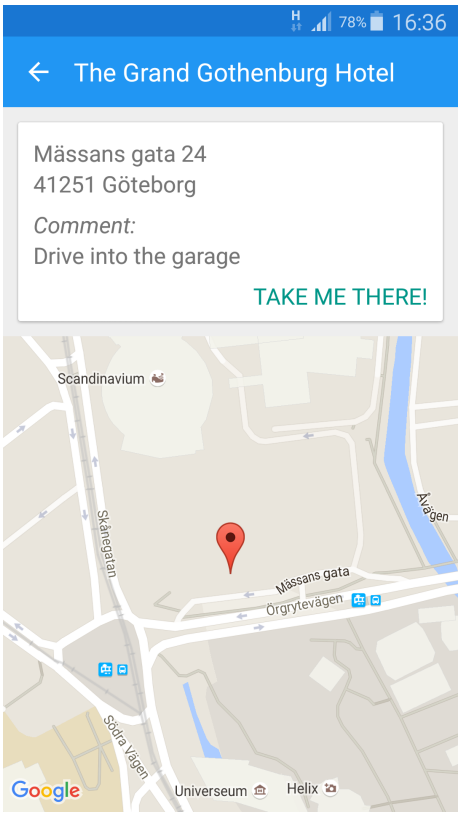
På varje körning finns en knapp för att bekräfta att informationen är läst och uppfattad vilket då meddelas till koordinatorerna. Den går att se i Figur 5.13. När en körning bekräftas ändras den knappen till en “start”-knapp, som är till för att meddela att en körning startats. Motsvarande funktion finns sedan för att stoppa en körning efter att den startats.

Det finns en vy som kallas “Locations”. Det är en lista med alla tänkbara platser som kan bli intressanta för en chaufför under en festival, se Figur 5.15. Dessa platser är fördefinierade av koordinatorerna innan festivalens början. Klickar chauffören på

en av platserna visas en vy med detaljerad information om platsen såsom adress, koordinatorns kommentarer om platsen samt en karta, se Figur 5.16. Klickar chauffören på knappen "Take me there!" kommer applikationen att med hjälp av Googles kartapplikation visa närmaste rutten, från chaufförens nuvarande plats till målet.



Figur 5.15: En lista med platser.

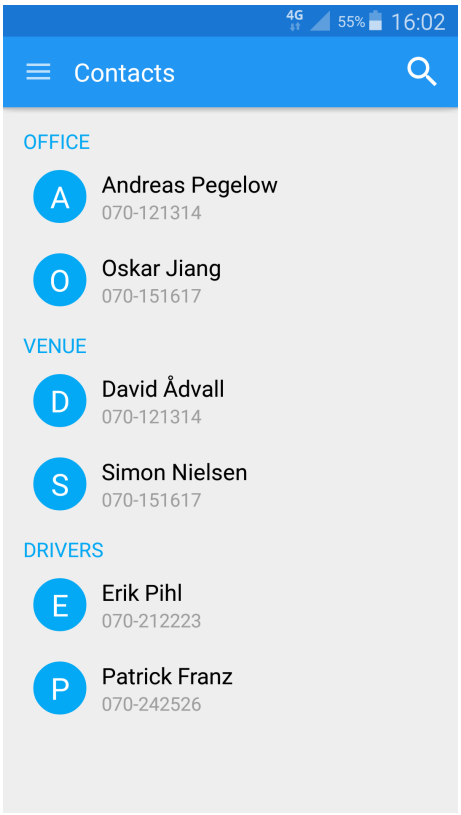


Figur 5.16: Detaljerad information om en plats under festivalen.

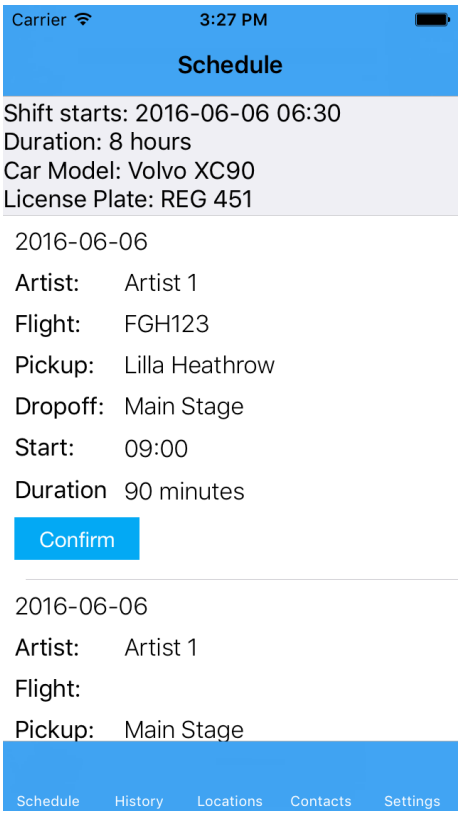
Det finns även en vy som kallas "Contacts" som är en lista vilken innehåller alla andra personer som ingår i projektet som kan var relevanta för en chaufför att kontakta. De är sorterade under kategorierna "Office", "Venue" och "Drivers". Detta går att se i Figur 5.17. Dessa kontakter är likt platserna fördefinierade av koordinatorerna.

Som det syns i "Navigation drawern" i Figur 5.12 finns det också två vyer som heter "Settings" och "About Eciton". I Settings kan användaren välja hur applikationen ska bete sig; till exempel finns valet att ändra mellan 24-timmars och 12-timmars klocka. I About Eciton finns information om applikationen och dess skapare. Dessa två funktionaliteter är dock inte helt färdigutvecklade i prototypen.

Alla skärmdumpar som har visats i detta kapitel är tagna från Android-versionen av applikationen. Som redan nämnt finns samma funktion i iOS versionen, om än inte lika finputsat. I Figur 5.18 går det att se hur schemavyn ser ut på iOS versionen i dagsläget.



Figur 5.17: Lista med intressanta kontakter.



Figur 5.18: iOS schemavyn.

I slutet av projektet utfördes ett användartest av applikationen med en person som har arbetat som chaufför vid många festivaler. Testet beskrivs i sin helhet i intervjuprotokollet (bilaga A). Enligt vad som framkom vid användartestet är det ingen komplicerad applikation, men det är viktigt att den är korrekt. Resultatet av intervjuerna var bra. Allt applikationen i nuläget kan göra var uppskattat med enbart ett fåtal förslag på funktioner som saknades. Testpersonen kunde lätt navigera sig utan att testledarna behövde lägga sig i. Det mest signifikanta intervjuerna gav var diskussioner kring formuleringar och hur knappar och dylikt är namngivna. Vilka ord som används får stor betydelse i förståelsen av applikationen.

6

Diskussion

I detta kapitel diskuteras projektet i sin helhet. Det innehåller gruppens egna tankar och jämförelser kring vad som gick bra och vad som gick mindre bra och borde ha gjorts annorlunda.

6.1 Metoddiskussion

Här föjer en diskussion kring metoden och genomförandet av de olika delarna av projektet. Har de val av ramverk och verktyg som gjorts varit rätt? Finns det tillfällen där det borde ha sökts efter andra alternativ?

6.1.1 Systemarkitektur

Under planeringen identiferades flera olika möjliga systemarkitekturer. En tidig insikt var att det behövdes en central punkt att lagra data då en del av uppgiften var att distribuera data till de olika chaufförerna. Databaser är dock endast bra på att lagra data så det behövdes ett lager mellan klienter och databasen och detta löses med en webbserver.

För att administratörsgränssnittet enkelt skulle kunna stödja många olika plattformar så implementerades det som en webbapplikation. Ett annat alternativ var att göra en applikation för att köra direkt i ett operativsystem.

Chaufförsgränssnittet kunde lösas på tre olika sätt: En webbsida, en native-applikation eller en hybrid-applikation. Att bygga det som en webbsida valdes bort tidigt då mobila nätverk fungerar dåligt under festivaler enligt Elvaett samt att det då inte är möjligt att skicka push-notiser till användaren. Valet att bygga två native-applikationer istället för en hybrid-applikation gjordes då kunskapen fanns för att snabbt skapa en Android-applikation. För att demonstrera chaufförsgränssnittet krävdes endast att en av native-applikationerna hade full funktionalitet.

6.1.2 Server

Ramverket Sails var avancerat och innehöll många delar. Det gjorde att det tog tid att sätta sig in i allt och utvecklingen började innan förståelse för alla delar fanns. Det resulterade i att en del saker som implementerades i början visade sig kunde göras på mycket bättre sätt senare. På grund av detta behövdes vissa delar, som implementerades i början, byggas om.

Överlag fungerade Sails som ramverk bra. Det fungerade att implementera de funktioner som systemet behövde även om alla krav inte var specificerade när valet av ramverk gjordes.

6.1.3 Schemaläggaren

Vid planeringsstadiet för schemaläggaren märktes det tydligt att problemet var för omfattande för att lösa med hjälp av en enda schemaläggning, vilket gjorde att de olika delarna isolerades och löstes var för sig. Detta var en av huvudanledningarna till att schemaläggaren är så pass komplett som den är eftersom att det möjliggjorde parallellt samt effektivare arbete med hjälp av abstraktion. En funktion som också gjordes tillgänglig genom detta var att köra olika delar av schemaläggaren separat beroende vad man har för data och vilket syfte man vill uppnå.

I ett tidigt skede bestämdes det att schemaläggaren skulle använda sig utav en SAT-lösare för att skapa scheman vilket inte var ett helt självklart val. Under inlärningsstadiet hittades artiklar som behandlade schemaläggningsteori som pekade på att genetiska algoritmer var en effektiv metod för att lösa problemet [3]. Detta alternativ ströks relativt snabbt då det bedömdes orimligt att implementera en så tillsynes avancerad algoritm under projektets tidsram utan några som helst förkunskaper inom det området. Istället var, tillsammans med SAT-lösaren, *Simulated Annealing* ett alternativ som diskuterades. Denna algoritm är en adaption från en algoritm som tillämpas inom fysik och använder sig delvis utav slumpen för att göra ett försök att hitta en lösning inom specificerade begränsningar [30]. Det var tänkt att denna optimeringsalgoritm skulle söka igenom alla möjliga scheman och välja ett schema som var bra nog inom en kortare tidsperiod än SAT-lösaren. Vidare efterforskning visade att *Simulated Annealing* förmodligen skulle fungera för vårt scenario men att den på grund av dess slumpmässiga natur hade en stor brist: Då lösning ej kan finnas kan det antingen bero på att det ej fanns en eller att den inte hittades. Den har dessutom svårt för att peka ut vad som kunde ha gjorts för att hitta lösningen. Felhanteringen var alltså ett stort problem och därför valdes, det eventuellt långsammare alternativet, SAT-lösaren för schemaläggaren.

Schemaläggaren i detta systemet är skriven i Java, vilket är ett programmeringsspråk som alla i gruppen är bekväma med. Detta gjorde att alla i gruppen kunde sätta sig in i algoritmen och bidra med tips och åsikter utan större svårigheter. Interaktionen

med databasen sker väldigt smidigt på grund av att språket är objektorienterat, vilket gör det möjligt att representera data från databasen som objekt. Detta underlättar mycket i två av schemaläggarens tre komponenter då schemaläggaren agerar mellanhand mellan databasen och SAT-lösaren, som också är implementerad med hjälp av Java. Ett problem med Java är dock att avancerade funktioner som består av mycket kod riskerar att bli väldigt svårlästa och svåra att felsöka. En lösning på detta som diskuterades var att att använda ett funktionellt programmeringsspråk istället. Dock hade gruppen sämre kunskap inom dessa språk och dessutom bedömde gruppen att det skulle bli svårare att hitta folk med tillräcklig kunskap inom dessa språk vid en eventuell överlämning av projektet för vidareutveckling.

Under projektet har arbetstid varit en begränsad resurs. Ambitionen har varit att ta fram ett system som fungerar i enlighet med syftet snarare än att ställa bortom all rimlig tvivel att det system som utvecklats använder den allra bästa teknik och teori som existerar i alla avseenden. Eftersom det finns väldigt mycket dokumenterad forskning som berör schemaläggningsteori som skulle kunna gå att applicera på de problem som är aktuella för Ecitons schemaläggare har inte varje tänkbart fält kunnat studeras i detalj. Istället har vägval gjorts utifrån den begränsade information som projektgruppen har kunnat tillgodogöra sig under begränsad tid, med fokus på att relativt snabbt finna en lösningsmetod som skulle fungera. Områden som under projektet, av olika anledningar, har förkastats efter korta översiktliga studier är *Integer Linear Programming* och det tidigare nämnda *Simulated Annealing*. Vid eventuell framtida utveckling av systemet skulle studier av dessa och möjligheten att förbättra systemet genom att applicera dem kunna bli aktuellt

6.1.4 Webbapplikation

Utvecklandet av webbapplikationen skedde förhållandevis långsamt på grund av att det fanns bristande kunskaper inom webbutveckling innan projektets start. Det var även av denna anledning som ramverket AngularJS 1 valdes. AngularJS är troligen det mest använda webbramverket just nu och är med råge det mest sökta vilket indikerar dess populäritet [31]. Detta innebär att det finns mycket mer information på nätet om Angular än något annat ramverk vilket är väldigt hjälpsamt. Detta var också anledningen att version 1 valdes då version 2 är såpass ny att den gick ur sitt beta-stadie den andra maj detta år.

En stor svårighet under webbapplikationens utveckling var också konfigureringen av gränssnittet med hjälp av CSS. Att framställa gränssnitt med hjälp av CSS är komplext och tidskrävande men väldigt viktigt för en bra användarupplevelse. Då schemat skapades från grunden utan att använda ett externt bibliotek krävdes mycket kodande i CSS men även JavaScript för att det skulle fungera.

6.1.5 Android & iOS

Gruppens beslut att utveckla applikationerna parallellt innebar både fördelar och nackdelar under utvecklingsstadiet. De funktioner som implementerades mer eller mindre samtidigt resulterade i bra genomtänkta beslut men också att ändringar behövde göras två gånger för att anpassa applikationerna till bland annat förändringar i databasen. Utvecklingstiden kortades dock ner en aning då det var möjligt att samarbeta för att lösa i princip samma problem. Något som påverkade utvecklingen långsiktigt var det faktum att det fanns två Android-utvecklare och endast en iOS-utvecklare. Detta, tillsammans med det faktum att det i gruppen fanns bättre förkunskaper inom Android-utveckling, ledde till att iOS-versionen hade svårt att hålla samma utvecklingstempo som Android-utvecklingen. Som följd av detta prövades senare en annan utvecklingsmetod som innebar att funktioner implementerades först på Android, testades och utvärderades för att sedan anpassas till iOS. Med detta kortades naturligtvis utvecklingstiden för iOS-applikationen ned samtidigt som Android-utvecklingen inte bromsades upp märkvärt. Mycket av detta berodde på att resten av systemet kunde anpassas ordentligt innan iOS-versionen utvecklades vilket sparade mycket arbete.

6.1.6 Arbetssätt

Tidigt under projektets gång beslutades det att utveckla systemet med hjälp av en typ av agilt utvecklande som kallas för Scrum. Detta med motivationen att det var ett populärt och väl omtalat arbetssätt som fungerar för många team. Men det tog inte lång tid innan många av de regler och speciella metoder som Scrum innefattar slutade användas eller tas på allvar. Ett exempel på något som används i Scrum som inte fungerade för detta projekt var tidsestimeringen. Tanken är att det som ska göras delas upp i mindre uppgifter som tidsestimeras. Sedan antecknas den tid det tog att utföra uppgiften och till sist följs det upp hur mycket jobb som blivit gjort. Detta fungerade inte då mycket av arbetet i ett kandidatarbete är väldigt svårt att tidsestimera, speciellt när mycket är helt nytt för utvecklarna. Projektet har dock under hela tiden använt sig av ett agilt arbetssätt med många av dess grundläggande principer så som den iterativa processen med uppföljande återkoppling.

En annan del av arbetssättet för projektet var att arbetet delades upp på arbetsgrupper. Dessa grupper har fungerat mycket väl och är något som rekommenderas att användas i framtiden av andra. Den enda nackdelen som uppkom var att kommunikationen mellan arbetsgrupperna stundtals hade kunnat vara bättre.

Till sist var versionshantering ett bra och i mångt och mycket ett självklart val för ett system i denna skala. Varje arbetsgrupp hade sin egen gren där de arbetade, som sammanfördes till mastergrenen vid release. Något som blev aningen svårt var när en arbetsgrupp behövde tillgång till de ändringar som en annan arbetsgrupp hade gjort för att kunna fortsätta sitt arbete men samtidigt inte kände sig redo för

att sammanföra grenarna. Ett exempel på detta var när det gjordes förändringar i databasen på dess gren som webbapplikationen behövde.

6.2 Resultatdiskussion

Här diskuteras projektets resultat. Är den slutgiltiga prototypen något som går att använda och är det den produkt som specificerades i början av projektet? Förekommer det funktioner från problemspecifikationen som inte finns med eller funktioner som inte borde finnas med och är ett resultat av en felaktig problemspecifikation?

6.2.1 Server & databas

Databasen implementerar inte alla relationer som behövs för att systemet ska få full funktionallitet. Det resulterar bland annat i att databasen inte kan hantera mer än ett projekt åt gången. Bristerna i databasen påverkar även webbserverns funktionalitet. Det gör att uppdateringar om skapade poster i databasen inte kan implementeras i Sails.

Information om uppdateringar fungerar inte med chaufförsapplikationerna då dess kommunikation inte sker via WebSocket-protokollet. Arbete med att skicka uppdateringar via GCM påbörjades men hann inte slutföras innan projektets slut. Systemet klarar att skicka meddelanden över GCM men information om uppdateringar i databasen skickas inte ut.

Många av dessa funktioner behövs om systemet ska användas av Elvaett men behövs inte för att demostera systemets huvudfunktionalitet.

6.2.2 Schemaläggaren

De scheman som skapas av schemläggaren har en del förbättringspotential. Den automatiska schemaläggaren skulle i nuläget kunna underlätta samt effektivisera det administrativa arbetet runt festivalen, men det har inte gått att fullständigt bekräfta att scheman som skapats går att använda i praktiken. Giltigheten för resultatets kvalitet är också diskutabel då schemaläggaren ej testades under en faktiskt festival, utan med påhittad data baserad på data en gammal festival. Mängden data som fanns tillgänglig för schemläggaren under testningen gav dock en relativt rättvis bild av hur det var i verkligheten då den arbetade helt utifrån data som koordinatorerna hade tillgång till. Det som inte gick att simulera var alla ändringar som kommunicerades till koordinatorerna i realtid, något som schemaläggaren hade tillgång till från början. Därför är det möjligt att kvaliteten på schemat hade sett annorlunda under förändrade förutsättningar. Större mängder testdata kunde också ha skapats för att testa funktionaliteten mer utförligt, men tidsbristen gjorde det svårt att skapa en

större mängd trovärdig testdata.

Beräkningstiden som de olika komponenterna i schemaläggaren behövde för att bli färdiga skiljde sig både mellan komponenterna sinsemellan och mellan den lilla och den stora testfestivalen. Beräkningstiden för komponenten arbetspassproduktion är väldigt snabb för små festivaler men väldigt mycket långsammare vid större festivaler. För en knappt tio gånger större festival, där förhållandet mellan tolvtimmarspass och åttatimmarspass är nästintill likadant jämfört med den mindre festivalen, ökar beräkningstiden i sekunder alltså med en faktor på nästan 4000(!). Men som tidigare nämnt i avsnitt 4.5.2 så ökar binärträdets storlek i algoritmen exponentiellt som funktion av antalet arbetspass som behövs, vilket verkar ha en stor inverkan på beräkningstiden. Det råder alltså stora problem med skalbarhet i denna delen av schemaläggaren. I de två andra komponenterna, där SAT-lösaren används, syns inte lika extrema skillnader i beräkningstid mellan de olika festivalerna. I komponenten som tilldelar arbetspass till chaufförer tar det ca 30 gånger längre tid att schemalägga den stora festivalen och i komponenten som tilldelar körningar till bilar hinner man poängsätta ca 9 gånger fler scheman för den lilla festivalen jämfört med den stora. Systemet skalar alltså i dagsläget dåligt mycket på grund av första komponenten.

6.2.3 Webbapplikation

Den resulterande webbapplikationen har väl lyckats lösa de problem som beskrevs i kapitel 2 gällande vad den ska kunna göra och dess användarvänlighet. Som tidigare nämnt i resultatkapitlet kan en användare göra följande.

- Se och redigera schemat för festivalen
- Hantera festivalens viktigaste resurser såsom
  - artister
  - bilar
  - chaufförer
  - platser
  - och transporter
- Interagera med schemaläggarens alla komponenter

För att systemet ska kunna användas saknas dock några funktioner som gås igenom nedan.

- Att kunna lägga till, redigera och ta bort körningar från en transport. Då körningar ändras väldigt mycket under en festivals gång enligt Elvaett är detta en funktion som måste finnas för att systemet ska vara användbart.
- Att kunna låsa körningar. Den funktionen är mycket viktig för att schemaläg-

garen ska kunna användas i praktiken eftersom en koordinator inte alltid vill schemalägga alla körningar på en gång. Till exempel då körningar tilldelats chaufförer tidigare och det är för nära inpå för att ändra på dem.

- Möjligheten att skapa ett nytt projekt för en festival.
- Ange när chaufförer är tillgängliga för att bli tilldelade arbetspass.
- Att kunna hantera användare. Detta är väldigt viktigt särskilt när det gäller chaufförer eftersom en chaufför måste logga in i mobilapplikationen och då måste den få ett användarnamn och lösenord.

Även om dessa funktioner saknas så har syftet med projektet uppnåtts vilket var att skapa en prototyp för att lösa problemet.

6.2.4 Android & iOS

Som tidigare nämnts i avsnitt 6.1.5 fanns det större resurser för Android-applikationen än för iOS-applikationen och Android-varianten innehåller således fler funktioner. I en intervju med en av Elvaetts chaufförer om Android-applikationen (se bilaga A) var återkopplingen till övervägande del positiv. Applikationen är ett steg framåt jämfört med det gamla systemet via SMS. Det är viktigt att applikationen är intuitiv och att användaren får en bekant känsla, det vill säga att användaren kan hantera applikationen utan större introduktioner. Vi fick några tips på vilka ställen mer detaljerad information är önskvärd. Till exempel var en viss bil är parkerad eller att kunna få viktiga meddelanden direkt i applikationen. Som kritikpunkt nämndes att det inte är tydligt vad knapparna för att bekräfta och starta en körning gör och när en chaufför förväntas använda dem.

Android-applikationen erbjuder alla grundfunktioner, men schema- och historik-vyn skulle behöva följa Googles Material Design mer konsekvent. Ändå kan applikationen betraktas som en färdig prototyp.

Jämfört med Android-varianten saknar däremot iOS-applikationen några funktioner. Dessa är historik-vyn, möjligheten att bekräfta/starta/avsluta en körning samt push-notiser.

6.3 Framtid

Ambitionen med projektet var att fram en prototyp av ett system som skulle kunna användas av Elvaett för att samordna det så kallade “Ground Transport” under framtida festivaler. Resultatet av detta blev något som enligt Elvaett nästan går att använda, ett system som är mer än bara en prototyp. Om de mindre sakerna som inte fungerar fixas efter projektets slut kan systemet användas för att koordinera under nästa festival enligt Elvaett.

6.3.1 Vad är det som saknas?

Det finns alltid saker att vidareutveckla i ett system som detta. Men det enda som saknas för att systemet ska kunna användas är följande som har diskuterats mer ingående tidigare:

- Att kunna lägga till, redigera, och ta bort körningar från en transport i webbapplikationen
- Att kunna låsa körningar i webbapplikationen
- Möjligheten att skapa ett nytt projekt för en festival i webbapplikationen
- Att i webbapplikationen kunna ange när chaufförer är tillgängliga för att bli tilldelade arbetspass
- Hantera användare i webbapplikationen
- Att iOS-applikationen kan skicka att en chaufför bekräftat, startat och avslutat en körning

Detta är förhållandevis små tillägg som förmodligen kommer att utvecklas inom kort tid efter projekets slut för att systemet ska användas.

6.3.2 Framtida utveckling utanför projektets avgränsning

Utöver det som behövs för att systemet ska gå att använda finns det en mängd förbättringar och funktioner som är efterfrågade av Elvaett. Några av dessa förbättringar gäller schemaläggaren som inte fungerar optimalt i den nuvarande versionen och har diskuterats tidigare. Schemaläggaren har en stor potential till att hjälpa till med att spara stora mängder pengar och arbetskraft genom optimeringar, därför är detta en väldigt intressant del av systemet. Men det är också en stor utmaning då det inte är trivialt att avgöra om ett schema är bra eller dåligt. Det kommer krävas mycket jobb för att få en bra schemaläggare, både genom utvecklande av algoritmerna men också med mycket testning i samarbete med erfarna koordinatorer.

Följaktligen listas några av de både mindre och större förbättringarna och funktionerna som kan utvecklas i framtiden:

- Föreslå lediga ekipage av chaufförer och bil vid ny transport i webbapplikationen
- En markör i schemat i webbapplikationen som visar var i schemat man befinner sig tidsmässigt
- Notifikationer till chaufförerna i mobilapplikationen. Då de till exempel tilldelats en ny körning
- Snygg design i iOS-applikationen.

Avslutningsvis finns en annan spännande aspekt som kan utvecklas för att få ett

bättre system. Det finns en idé om att modifiera mobilapplikationen så att den även kan användas av artister och turnéledare. Detta skulle underlätta kommunikationen mellan artist/turnéledare och koordinatorer då de till exempel skulle kunna se sina transporter direkt i mobilapplikationen och få uppdateringar med push-notiser.

6.4 Ecitons roll i samhället

De transporter som sker under festivaler, ibland uppemot hundratals, står tillsammans för väldigt mycket utsläpp samtidigt som stora ekonomiska resurser går åt till att administrera festivalen. Bättre schemaläggning av transporter skulle innebära att antalet körda kilometer under en festival skulle kunna minskas vilket i sin tur ger mindre utsläpp. En effektivisering inom denna bransch skulle därför innebära ekonomiska fördelar samt bidrag till en hållbar framtid för miljön, en effektivisering som Eciton förhoppningsvis kommer att vara en del av. Ett skräddarsytt program som detta för koordinering av transporter kommer också bidra till en förbättrad psykosocial arbetsmiljö i form av mindre stress då Eciton underlättar för de koordinatorer som sitter och arbetar med schemaläggning under festivalen.

6.5 Slutsats

I enlighet med projektets syfte har en prototyp för ett trafikledningssystem som schemalägger och koordinerar transporter vid musikfestivaler och andra event tagits fram inom ramarna för projektet. Därmed kan det anses visat att det är möjligt att skapa ett sådant system.

I arbetet har flera olika sätt att lösa varje delproblem undersökts och utvärderats i syfte att finna den bästa lösningsmetoden för respektive delproblem utifrån projektets förutsättningar. Eftersom projektets arbetstidsresurser varit begränsade har dessa undersökningar inte varit av så omfattande karaktär att det är ställt bortom all rimlig tvivel att den lösning som valts är den allra bästa utifrån projektets mål. Däremot har varje undersökning och utvärdering varit tillräckligt omfattande för att det ska kunna anses säkerställt att den lösning som valts fungerar och är rimlig utifrån projektets mål.

Arbetet som projektgruppen har utfört har stundtals varit svårt, men allt som projektgruppen har erfarit pekar mot att resultatet är väldigt bra i relation till vad projektgruppen hade hoppats kunna åstadkomma. Det är sannolikt att den prototyp som tagits fram, med relativt enkla medel, kan utvecklas till en kommersiell produkt som kan användas vid skarpa operationer.

Det har varit en mycket lärorik process för projektgruppen, där dess medlemmar

har fått god träning i såväl problemlösning som i alla de olika tekniker som har använts.

Litteratur

[1] Google Developers. *Cloud Messaging*. URL: https://developers.google.com/cloud-messaging/ (hämtad 2016-05-11).

[2] INFORMS. *What is Operations Research?* URL: https://www.informs.org/About-INFORMS/What-is-Operations-Research (hämtad 2016-05-12).

[3] E. Burke m. fl. "A memetic approach to the nurse rostering problem". I: *Applied intelligence* 15.3 (2001), s. 199–214. URL: http://search.proquest.com.proxy.lib.chalmers.se/docview/879434089?pq-origsite=360link (hämtad 2016-05-06).

[4] M. S. Gondane och D. R. Zanwar. *Staff Scheduling in Health Care Systems*. URL: http://www.iosrjournals.org/iosr-jmce/papers/vol1-issue6/E0162840.pdf (hämtad 2016-05-15).

[5] A. Cooper, R. Reimann och D. Cronin. "About Face: The Essentials of Interaction Design". I: 3. utg. 10475 Crosspoint Boulevard Indianapolis, IN 46256: Wiley Publishing, Inc, maj 2007. Kap. 11, s. 223–248.

[6] R. Fielding m. fl. *Hypertext Transfer Protocol - HTTP/1.1*. URL: https://tools.ietf.org/html/rfc2616 (hämtad 2016-04-22).

[7] I. Fette och A. Melnikov. *The WebSocket Protocol*. URL: https://tools.ietf.org/html/rfc6455 (hämtad 2016-04-22).

[8] D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. URL: https://tools.ietf.org/html/rfc4627 (hämtad 2016-04-22).

[9] E. K. Burke m. fl. "The State of the Art of Nurse Rostering". English. I: *Journal of Scheduling* 7.6 (nov. 2004), s. 441–499. URL: http://proxy.lib.chalmers.se/login?url=http://search.proquest.com/docview/232425164?accountid=10041 (hämtad 2016-05-16).

[10] S. Kundu och S. Acharyya. "A SAT approach for solving the nurse scheduling problem". I: *TENCON 2008 - 2008 IEEE Region 10 Conference*. Nov. 2008, s. 1–6. DOI: 10.1109/TENCON.2008.4766380. URL: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/xpls/icp.jsp?arnumber=4766380 (hämtad 2016-05-06).

[11] B. M. W. Cheng, J. H. M. Lee och J. C. K. Wu. "A nurse rostering system using constraint programming and redundant modeling". I: *IEEE Transactions on*

Information Technology in Biomedicine 1.1 (mars 1997), s. 44–54. ISSN: 1089-7771. URL: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/xpl/articleDetails.jsp?arnumber=594027 (hämtad 2016-05-06).

[12] Google. *JavaScript-biblioteket AngularJS*. URL: https://angularjs.org/ (hämtad 2016-05-15).

[13] Google. *Beskrivning av MVC*. URL: https://developer.chrome.com/apps/app_frameworks (hämtad 2016-05-15).

[14] Android Open Source Project. *The Android Source Code*. URL: https://source.android.com/source/index.html (hämtad 2016-04-22).

[15] Android Open Source Project. *Licences*. URL: https://source.android.com/source/licenses.html (hämtad 2016-04-21).

[16] Android Developers. *Android Studio*. URL: https://developer.android.com/sdk/index.html (hämtad 2016-04-21).

[17] Apple Inc. *What's new in Xcode 7*. URL: https://developer.apple.com/xcode/ (hämtad 2016-05-15).

[18] Scrum Alliance Inc. *Learn about Scrum*. URL: https://www.scrumalliance.org/why-scrum (hämtad 2016-05-30).

[19] GitHub. *About GitHub*. URL: https://github.com/about (hämtad 2016-04-22).

[20] MySQL. *MySQL Community Downloads*. URL: https://dev.mysql.com/downloads/ (hämtad 2016-04-22).

[21] M. McNeil. *Sails.js*. URL: http://sailsjs.org/ (hämtad 2016-05-16).

[22] *JavaScript-biblioteket MomentJS*. URL: http://momentjs.com/ (hämtad 2016-05-15).

[23] Google. *JavaScript-biblioteket Angular Material*. URL: https://material.angularjs.org/ (hämtad 2016-05-15).

[24] Gartner. *Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015*. URL: https://www.gartner.com/newsroom/id/3215217 (hämtad 2016-05-16).

[25] Google. *Material design*. URL: http://www.google.com/design/spec/material-design/introduction.html (hämtad 2016-05-11).

[26] Apple Inc. *iOS Human Interface Guidelines: Designing for iOS*. URL: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ (hämtad 2016-05-16).

[27] Android Developers. *Dashboard*. URL: https://developer.android.com/about/dashboards/index.html (hämtad 2016-05-11).

[28] Apple Inc. *Nyheter i iOS*. URL: https://www.apple.com/se/ios/whats-new/#compatibility (hämtad 2016-05-11).

[29] Localytics. *Research Shows the iPhone 6 Is the Most Adopted iPhone Model With the Highest App Engagement*. URL: http://info.localytics.com/blog/research-shows-the-iphone-6-is-the-most-adopted-iphone-model-with-the-highest-user-engagement (hämtad 2016-05-11).

[30] S. Kirkpatrick, C. D. Gelatt och M. P. Vecchi. “Optimization by Simulated Annealing”. I: *Science* 220 (1983), s. 671–680. DOI: 10.1126/science.220.4598.671. URL: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.123.7607&rep=rep1&type=pdf (hämtad 2016-05-09).

[31] *Google Trends*. URL: https://www.google.com/trends/explore#q=backbone%20js,%20ember%20js,%20angular%20js&cmpt=q&tz=Etc/GMT-2 (hämtad 2016-05-16).

A

Intervju med Jens Clausen

Jens Clausen har jobbat som chaufför för Elvaett i många år.

1.) Allmänna frågor: Vad tycker du om det nuvarande systemet? Var är bra? Var är dåligt? Vad kan förbättras? Hur kunde det förbättras? Vad för telefon använder du idag (Android eller iOS)?

Jens använder Android i vanliga fall.“Det nuvarande systemet funkar sådär. Jag får det jag behöver via sms. Det är dåligt när nätet ligger ner kring festivalområderna. Det är viktigt att veta vilka de andra förarna är, och då är telefonnummer viktigt även flightnummer är bra att se. Man vill inte flagga med artistens namn, internt används artistens namn men visas inte utåt för att undvika autografjägare. Specialinfo är bra att få, det har varit rörigt innan. Möjligheten att fylla i extra info är viktigt tycker Jens.”

2.) Logga in med driver/1234

Det gick fint!

3.) Browsa i appen i 1-2 minuter

Det första Jens säger är “Det är inte speciellt svårt [att ta sig runt i appen]”
Confirm, verkar lite luddigt, bekräftad av vem?
Efter en stund förstår Jens vad confirm betyder utan av testledarna behöver berätta.
Luggage är viktigt, bra! I show more finns extra info, bra! “Jaha, här ser man andra förare, och hans nummer. Det vore ju bra om man kan klicka på de så man kan ringa dit!” [Det kan man]
Tour manager, sankas. Det är viktigt!
I skift kortet hade det varit bra om det står var bilen är parkerad. Detta har tidigare gjorts med postit lappar.

Det jag ser hittills tycker jag om!

Om det blir ändringar vill man inte att koordinatorerna ringer. När man kör pratar men inte i telefon. Bättre att de kommer upp i telefonen så att när man stannar och avslutar sin körning kan kolla ändringar då.

Jens kollar vidare i appen och kommer på många funktioner han vill ha, upptäcker sedan att det redan finns implementerade.

4.) När börjar och slutar ditt nästa skift? Vilken bil ska du köra?

Börjar måndag 6 juni 06:30, slutar 14:30 Volvo XC90. Jens hittade rätt.

Jens skulle vilja veta ID:et på bilen. Det är det som används när man pratar om bilarna.

5.) Gå till din första körningen? Känns infon rätt/lämplig? Hittar du all relevant info? Saknas det någonting? Är det svårt att hitta info?

Se svaren på fråga 3 men i stort sätt finns allt med.

Det hade varit bra om koordinatorena kan skicka info till förarna som inte direkt har något att göra med en körning. Väg avstängd, andra förare sjuka eller liknande genom en Push notis, meddelande funktion eller något i den stilen. Meddelandena går båda vägarna mellan förare och koordinator.

(Vi förutsätter, att du inte behöver en karta för den här körningen) 6.) Vad gör Confirm-knappen?

Jo, jag förstår! säger Jens. Vi håller med.

7.) Bekräfta körningen. Är det tydligt nog?

Bra med popuppen, vill du verkligen bekräfta!

8.) Vad gör Start-knappen?

Denna funktion är mer oklar. När startar man. För Jens börjar uppdraget när han åker från Slottskogen, inte bara när artisten är i bilen. Bara det är tydligt definierat vilket som gäller så är de bra.

9.) Starta körningen. Är det tydligt nog?

Åter igen, bra med popuppen.

10.) Avsluta körningen. Är det tydligt nog?

11.) Försök hitta körningen i historiken. Är det bra eller överflödigt att ha historikvyn? Varför?

Det är bra att logga för lönen, det är hittills skötts analogt.

12.) Vi förutsätter nu med att du måste kolla startplatsen, men vet hur du tar dig till målet.

Det fungerade bra.

13.) Vi förutsätter nu med att du inte vet hur man tar sig från start till målet. Välj den tredje körningen.

Jens tar sig lätt via schemat till Googlemaps och får körinstruktioner där ifrån.

14.) Kolla hur man kommer från start till mål.

Googlemaps är super bra! Bra med standardsätt som man redan känner till, det blir

intuitivt och bra.

15.) Ring den andra föraren för körning nr. X, eftersom du är lite sen.

Den andra föraren hittas lätt. Än så länge kan man inte klicka på föraren i show more tabben i schemat. Nu fick Jens gå omvägen via Contacts vyn.

16.) Bekräfta sista körning. Ring Oskar på kontoret och berätta att du inte kan ta körningen.

Jens förstår vad som måste göras. Efter att han bekräftat sin körning så går han till Contacts vyn och hittar numret till en som befinner sig under office. I praktiken så har man ett nummer till kontoret som man ringer inte till specifika personer förutom i nödfall.

17.) Push Notiser

Jens fick en notifikation om att hans nästa körning börjar om 30 minuter. Snyggt! 30 minuter känns instinktivt bra. Efter lite funderande blir det mer oklart. Det är 30 minuter innan man behöver agera som gäller. Får man de 30 minuterna innan man ska hämta på landvetter så hinner man inte. Ditt schema har uppdaterats är en viktigt notis. Det vore trevligt om appen direkt visar vilken av körningarna som har uppdaterats eller tillkommit.

18.) Vad är ditt allmänna intryck av appen? Något som saknas?

Det känns bra och ett stort steg framåt mot en lång lista med SMS. Det är viktigt att appen är intuitiv, den ska se ut och funkar som appar brukar göra så att man inte behöver tänka. Man kanske kunna lägga in lite fler steg i statuskedjan av bekräfta, starta, stoppa. Typ, jag sitter i bilen har har börjat mitt skift, jag har vaknat eller jag är påväg till slottksogen.