Utveckling av android-applikation för kollektivtrafik

- Handskakning mellan applikation och buss

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Datateknik

Examensarbete:
Jimmie Jönsson

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund <2013>

Förord

Jag vill tacka Trivector för att jag har kunnat utföra mitt examensarbete hos dem och vill även tacka alla på företaget som hjälpt mig på diverse sätt. Speciellt tack till Klas Odelid, handledare på Trivector och Christian Nyberg, examintor.

Sammanfattning

Detta examensarbete har gått ut på att bygga och vidareutveckla en applikation kallad ”Var f-n är min buss?” för företaget Trivector Systems i Lund.

”Var f-n är min buss?” är en applikation till Smartphones som ska visa bussars avgångstid från en vald hållplats, linjesträckning, hållplatser och bussar för en vald linje och nästkommande hållplatser och avgångstider för dessa för en vald buss.

Applikationen hade redan påbörjats men det fanns problem då projektet började. Dels fanns där en bugg och dels så var det problem då applikationen skulle utvecklas till Android. Problemet var en begränsning som finns i Android.

Den första delen av projektet var alltså att först utveckla applikationen till Android och därmed lösa buggen och arbeta runt begränsningen för Android. Utöver detta så vill Trivector att det skulle finnas en handskakning mellan applikationen och den buss som användaren stiger på, för att applikationen ska veta vilken buss användaren befinner sig på och då kunna markera det i applikationen. Uppdraget var att finna den bästa teknik för att göra handskakningen möjlig och den teknik som är mest lämplig för detta ändamål är Wi-Fi.

Nyckelord: Handskakning, Buss, Kollektivtrafik, Wi-Fi, Android

Abstract

This thesis involves creating an extending an application called “Var f-n är bussen?” (something like “Where the hell is that bus?”) for the company Trivector Systems in Lund.

“Where the hell is that bus?” is an application for smartphones that will show departure time for buses from a chosen bus stop, linepath, bus stops and buses for a chosen bus line and the next bus stops and departure time from these bus stops for a chosen bus.

The application had already been partly developed but there were issues with it when the thesis started. There was a bug and there were also a problem when developing this application for Android. The problem was a limitation in Android.

The job was first of all to develop this application for Android, and by doing so, solving the bug and finding a work around for the limitation in Android. As an extension to this, Trivector wanted some kind of handshake between the application and the bus that the user gets on, so that the application can know what bus the user is currently is on, and by knowing that, being able to mark that bus in the application. The mission was to find the best technology to make this handshake possible and the technology best suited for this purpose is Wi-Fi.

Keywords: Handshake, bus, public transport, Wi-Fi, Android

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Trivector är ett av Sveriges mest framstående företag inom avancerade IT-system för kollektivtrafik. De skapade bland annat Sveriges första fordonsdator till kollektivtrafik, det första systemet som använder pekgränssnitt till reseplanering och det första systemet för radiobaserad signalprioritering. De erbjuder tjänster och produkter inom system för realtidsbaserad eller tidtabellsbunden trafikinformation, publika informationssystem och service och underhåll.
Inom system för trafikinformation har de utvecklat ett komplett integrerande informationssystem, kallat TriTrans, som består av tre delar, fordonssystem med depåsystem, centralsystem med trafikledning och presentationssystem.

Numera finns Trivectors system i fler än 1000 nordiska fordon samt drygt 300 järnvägsstationer, resecentra och busshållplatser, där aktuell trafikinformation presenteras på över 1500 skyltar, monitorer och talenheter från Trivector Systems. En tilltagande andel av dessa visar trafikinformationen i realtid. Arbetet kommer att gå ut på att utveckla en applikation till smartphones. Applikationen ligger inom området System för realtidsbaserad eller tidtabellsbunden trafikinformation och kommer att vara en applikation som visar bussar i realtid, om möjligt, annars gå efter tidtabellen och kunna visa var på en vald linje bussar befinner sig.

Detta examensarbete tillför ännu ett medium där kollektivtrafiksinformationen kan visas i realtid.

1.2 Syfte och mål

Syftet med mitt examensarbete är utbyggnaden av en applikation, skämtsamt kallad ”Vart f-n är min buss?”, som Trivector Systems har börjat utveckla, för att i realtid se var bussar befinner sig på en vald linje, eller när bussar anländer till en vald hållplats. Utbyggnaden ska bestå i att det ska finnas någon form av handskakning mellan applikationen och bussens fordonsdator när användaren stiger på en buss.

Målet med examensarbetet är att en användare ska kunna plocka upp sin mobiltelefon, öppna applikationen, välja en hållplats och sedan kunna se i realtid när nästa och nästnästa buss avgår från den valda hållplatsen. När användaren stiger på en buss ska det visas i applikationen vilken buss användaren befinner sig på.

1.3 Problemformulering

Frågan som ligger till grund för examensarbetet är vilken teknik som ska användas för att realisera problemet ”handshake med buss”.

För att svara på frågan ”hur handshake med buss ska realiseras” ska det undersökas vilken teknik som fungerar bäst för att handskakningen ska fungera så bra och felfritt som möjligt. De punkter som tas hänsyn till vid undersökningen av de olika metoderna är:

- Kostnad, pris för att kunna använda metoden, ifall något inköp behövs för företaget.
- Kostnad i batteriförbrukning, för telefonen.
- Metoden ska gå att använda på en stor mängd olika telefoner
- Metoden ska vara stabil, med andra ord ska det inte vara lättstörligt, och det ska inte vara buggigt.

De tekniker som finns som alternativ för att kunna göra denna handskakning möjlig är Bluetooth, Wi-Fi, NFC, QR-kod eller GPS.

I undersökningen av dessa kommer det att tas reda på information om varje alternativ genom sökning på Internet och analysera vilket av alternativ som passar bäst för handskakningen. De trådlösa alternativen kommer att testas i rörelse för att se huruvida tekniken klarar av det.

Förutom att undersöka vilken teknik som är bäst för handskakningen så kommer applikationen ”Var f-n är min buss?” att utvecklas. Vid examensarbetets början fanns där en bugg som gäller att tömma en tabell och dessutom problem med kartobjekt i Android. Stor vikt läggs på att lösa dessa problem.

1.4 Avgränsningar

Vid tidsbrist kommer applikationen endast att utvecklas till Android. Dessutom kommer ”handshake med buss” eventuellt inte implementeras utan endast analyseras.

2 Metod

Man kan säga att detta examensarbete delas upp i två olika steg. Det första steget är att utveckla applikationen ”Var f-n är min buss?”. Det andra steget är att analysera vilket alternativ som är bäst för handskakningen med bussen. Även om stegen benämns som första och andra steget så kommer de att fortlöpa parallellt då den färdiga applikationen inte är ett måste för själva analysen.

2.1 Utveckling av applikationen

Vid examensarbetets början fanns en första version av applikationen till iPhone. Denna del av examensarbetet gick ut på att utveckla applikationen till Android och att lägga till funktioner som inte fanns i den första versionen av applikationen. Applikationen är skriven i JavaScript och på grund av brist i kunskap om JavaScript användes en guide, http://www.w3schools.com/js/, och dessutom ett färdigt program i utvecklingsverktyget Titanium som är skapat just för inlärning, kallat KitchenSink. I KitchenSink finns exempel på de vanligaste sätten att använda JavaScript för applikationsutveckling i Titanium. Inlärning skedde även genom studerande av den första versionen av applikationen.

Utvecklingen av applikationen har skett på ett sätt liknande Kanban. Skillnaderna är att utvecklingen har utförts på egen hand (endast en person) och istället för en Kanban board så har de olika uppgifterna skrivits ner i ett häfte. Detta häfte har fungerat som underlag för vilken del i applikationen som fick bli nästa del att utveckla. De olika delarnas prioritering har diskuterats med handledare på Trivector och utvecklats efter prioriteringsföljd. Handledaren har tittat förbi då och då och bett om andra funktioner i applikationen och de har då prioriterats och lagts till i häftet. Har alla punkter vid något tillfälle kryssats av har diskussion med handledare inletts för att bestämma nästa del i utvecklingen. En ny punkt i häftet har ej påbörjats förrän den förra punkten är avklarad. Detta för att undvika att ha massa halvutvecklade funktioner. Med andra ord är WIP (work in progress) gränsen satt till ett (1).

2.2 Analys av tekniker för handskakningen med buss

Vid analysen har olika metoder för att ta reda på bästa möjliga alternativ använts. Informationssökning om varje enskilt alternativ har skett på Internet. Tester för de trådlösa alternativen har skett på 15 meters avstånd i ett tåg i rörelse för att se vilket av de trådlösa alternativen som bäst klarar av att hitta tillgängliga nätverk. Diskussioner med anställda på Trivector har även

genomförts för att komma fram till ett alternativ som passar bäst för fordonsdatorerna i bussarna.

2.3 Källkritik

Källorna (se 9 Referenser) som används vid sökningen på Internet känns trovärdiga då det är i största delen varje tekniks officiella hemsida.

En hemsida som däremot inte är en officiell hemsida är http://lifehacker.com/5369381/turn-your-windows-7-pc-into-a-wireless-hotspot. På denna hemsida förklaras det hur, som titeln avslöjar, man kan göra sin Windows 7 pc till en trådlös åtkomstpunkt. Eftersom det går att följa stegen på hemsidan, och eftersom att det fungerar så anses även denna hemsida trovärdig.

3 Analys

Den öppna fråga i examensarbetet är hur en handshake med en buss skulle kunna ske, det vill säga hur skulle man kunna koppla ihop applikationen med den buss man stiger på? Detta för att applikationen ska veta vilken buss man befinner sig på och då kunna markera den bussen i applikationen.

De olika alternativen som skulle kunna användas och som genomgås för att göra denna handshake möjlig är Bluetooth, NFC, QR-kod, Wi-Fi eller GPS.

3.1 Trådlös handshake

De trådlösa alternativen är Bluetooth och Wi-Fi. I båda alternativen kommer det behövas någon form av program, eller programdel som skannar efter tillgängliga Wi-Fi respektive Bluetooth nät, och läser av dess namn.
Den första delen av bussarnas Wi-Fi SSID eller BluetoothUUID kan identifiera exempelvis vilket trafikbolag bussen tillhör. Den senare delen av nätets namn kan användas för att identifiera vilken buss användaren har stigit på. För att identifiera bussen används bussens ID, som är tre eller fyra siffror. Då ett nät som ser ut på rätt sätt hittas avlägsnas ”Karlstadsbuss” och kvar finns endast bussens ID, och då vet applikationen vilken buss användaren befinner sig på.
Med andra ord behöver telefonen aldrig kopplas ihop med varken Wi-Fi eller bluetooth, utan det räcker med att applikationen skannar efter tillgängliga nät och läser ut bussens ID från SSID’t eller bluetoothnamnet.
När man skapar ett Bluetooth-nät kan man till skillnad från Wi-Fi inte själv välja ett namn utan nätet får ett namn beroende av vilken tjänst som enheten erbjuder. [9]
När man åker buss kommer den att förflytta sig och frågan är vilken teknik som är bäst, om det är Wi-Fi eller om det är Bluetooth, som bäst klarar av uppkopplingen.

3.1.1 Bluetooth

Fördelar med att använda Bluetooth som handshake-lösning är att räckvidden når genom hela bussen (klass 2 radio i de flesta telefoner, 10m), det är en billig teknik, dels pengamässigt, eftersom Bluetooth redan finns i de flesta telefoner och det är inget extra som behövs köpas till, och dels batterimässigt då Bluetooth drar väldigt lite batteri. Det är även en fördel att det som sagt finns i de allra flesta telefoner idag och är en väldigt stabil teknik och är som nämnt även motståndskraftigt mot störningar. [8] Vid test visade det sig även att Bluetooth fungerade utmärkt i rörelse och även på längre avstånd än 10 meter.

Det som är negativt är att det inte är säkert att fordonsdatorn i bussen har Bluetooth. Dessutom kommer det att krävas ett unikt UUID, som är Bluetooths motsvarighet till WiFis SSID. För att UUIDt ska bli unikt krävs det att fordonsdatorn erbjuder en unik tjänst.

Det finns dock en positiv sak med att det inte själv går att välja UUID, och det är att det är svårt att återskapa det. SSID går ju som sagt att själv välja och därför är det också väldigt lätt att återskapa, och alla med en smartphone skulle kunna göra det. Anledningen till att det är bra att det inte går att återskapa är för att det inte ska gå att luras. Någon skulle, om denne vet på vilket sätt applikationen är uppbyggd, kunna skriva in den första delen, som skulle kunna gå att lista ut genom att kolla tillgängliga nätverk på bussen, och sedan skriva in vad som helst som ID. Det är svårt att använda detta sätt att luras för att göra någon egentlig skada, men det kommer ändå eventuellt att förvirra användaren och kanske till och med få användaren att stiga av bussen i tron att ha hamnat på fel buss.

3.1.2 Wi-Fi

Vid användning av Wi-Fi för detta ändamål måste fordonsdatorn kunna fungera som en accesspunkt. Fordonsdatorn är en PC som kör Linux Ubuntu, så att sätta upp ett ad-hoc nätverk är inga problem.
För att använda datorn som accesspunkt måste ett ad-hoc nätverk skapas och datorn behöver ha ett ledigt trådlöst nätverkskort. Vill man exempelvis dela internet så måste datorn ha antingen internetuppkoppling med kabel, eller ha ett andra nätverkskort till internetanslutningen.[13] Det finns två alternativ på hur uppkopplingen med Wi-Fi skulle kunna fungera.

Antingen så kommer användaren endast behöva starta sitt Wi-Fi, se SSID, som är namnet på bussens Wi-Fi, och använda det för att tala om för applikationen vilken buss användaren befinner sig på, utan att ansluta till bussens Wi-Fi. Detta alternativ hade realiserats genom att varje fordonsdator har ett ad-hoc nätverk, som döps på ett unikt sätt. Detta unika sätt, kan vara exempel ”Karlstadsbuss” för alla bussar, följt av bussens ID för varje enskild buss. I applikationen kommer det att implementeras en skanner som skannar efter trådlösa nätverk. Då ett tillgängligt nätverk hittas, kontrolleras SSIDt och om det börjar med ”Karlstadsbuss” så kan applikationen strippa bort det och efter finns endast bussens ID, och applikationen vet då vilken buss det är som användaren befinner sig på.

Den andra lösningen är att användaren ansluter sig till bussens Wi-Fi, men problemet är då att samtidigt som telefonen kopplar upp till Wi-Fi, kopplas det

mobila nätet av och därför kommer det inte att finnas tillgång till internet längre, såvida inte användaren vid uppkoppling mot fordonsdatorn får tillgång till internet via bussens Wi-Fi. Detta hade inneburit att fordonsdatorn hade behövt ett ledigt trådlöst nätverkskort och dessutom hade det inneburit att det hade funnits gratis Wi-Fi på alla bussar. Frågan är om man vill och klarar av att ha det?

I båda fallen är det styrkan på nätet som avgör vilken buss användaren befinner sig på. Detta ifall användaren befinner sig i närheten av mer än en buss samtidigt. Ett problem här kan faktiskt uppstå ifall användaren sätter sig långt bak i bussen och det står en annan buss precis bakom. Då kommer signalstyrkan för den andra bussen vara starkast och alltså ange fel buss för användaren.

Den förstnämnda lösningen förutsätter att det finns en knapp i applikationen så att användaren själv kan bestämma om applikationen ska skanna efter tillgängliga nätverk. Detta av två anledningar. Dels för att applikationen inte ska skanna efter tillgängliga nätverk hela tiden, eftersom detta är väldigt batterikrävande och för att det hade krävt att WiFi är igång under hela tiden applikationen körs. Och dels för att om skanningen hade varit aktiv hela tiden så skulle det räcka att användaren står i närheten av en buss, eller rentav kanske det räcker med att en buss kör förbi, för att applikationen ska meddela att användaren befinner sig på den bussen.

Då en PC med Windows eller Linux skapar ett trådlöst nätverk så blir det ett ad-hoc nätverk. Android stödjer tyvärr inte ad-hoc nätverk utan för att en Android smartphone ska kunna hitta det nätverk som datorn skapar så krävs det att datorn fungerar som en AP (access point) och inte som ett ad-hoc nätverk.

Till Windows finns det ett program som heter Connectify som enkelt gör så att datorn fungerar som en accesspunkt istället för en enhet i ett ad-hoc nätverk [14].

Till fordonsdatorn, som använder Linux, är det lite jobbigare. Det krävs för det första ett program som heter Hostapd. Utöver detta program krävs det att både det nätverkskort som sitter i datorn och drivrutinerna till nätverkskortet stödjer ett så kallat master-mode. Dessutom måste nätverket konfigureras och detta är en process som inte tas upp i detta examensarbete.

Varken Connectify för Windows och Hostapd för Linux kräver att det finns ett ledigt nätverkskort för att dela Internet. Det problemet försvinner med andra ord också då något av dessa program används.

Wi-Fi testades för bussmiljö i ett tåg genom att en smartphone fick agera accesspunkt och en annan telefon fick försöka ansluta till accesspunkten. Avståndet var längre än 10 meter och tåget var i rörelse. Uppkopplingen fungerade utan problem.

Fördelar med Wi-Fi är att i stort sett alla nya telefoner har det inbyggt och de allra flesta användarna vet hur Wi-Fi fungerar. Dessutom visar testning att det fungerar utmärkt på längre avstånd än 10 meter och det fungerar i rörelse.

Nackdelen med det är att det är väldigt batterikrävande, särskilt om det skannar efter tillgängliga nätverk.

3.2 Icke-trådlös handshake

De icke-trådlösa alternativen är NFC och QR-kod. En bra anledning till att använda just någon av dessa två metoder är för att det finns ingen direkt störning då dessa används, just eftersom avståndet mellan telefonen och QR-koden eller NFC-sändaren är så kort. Detta gör dem även väldigt säkra.

3.2.1 NFC (Near Field Communication eller närfältskommunikation)

Nackdelen med NFC är att varje tag kostar en viss summa, beroende på utseende och form. Priset varierar även på hur många som köps in, ju större beställning desto bättre pris. För att folk ska förstå att NFC taggen tillhör just denna applikationen så kunde NFC taggarna ha ett speciellt utseende. Detta går att åstadkomma med custom-print nfc tags från www.rapidnfc.com[11] . Där kostar exempelvis 100 specialbeställda klistermärken £0.95 stycket (motsvarar 10.14kr enligt www.forex.se , 30/5-2013).

Det hade behövts köpas minst en tag till varje buss, men helst fler, då man inte vill ha folk som ska stå och läsa in information från taggen i samma kö som de som ska betala. Taggarna hade fått placeras på olika ställen i bussen, gärna där det är lätt att komma till. Allra bäst för varje enskild användare hade varit ifall det hade funnits en tag vid varje sittplats.
En annan nackdel med NFC är att det inte är alla telefoner som är kompatibla, utan det är i skrivande stund bara de nyare som kan använda den här tekniken. Då passar istället Bluetooth eller Wi-Fi bättre, eftersom den tekniken har funnits längre och i stort sett alla telefoner har dessa tekniker idag.

3.2.2 QR-kod

Fördelar med QR-kod är att program för att läsa denna kod finns att få tag på till alla nya mobiltelefoner. QR-kod kräver ingen extra tag eller liknande, som exempelvis NFC gör, utan det räcker att man skriver ut koden på vanligt papper.

Nackdelar med QR-koder är för att avläsa en QR-kod behöver användaren införskaffa en applikation till sin smartphone. Denna applikation är i och för sig väldigt lätt att få tag på och finns gratis både på Androids Google Play och på Apples AppStore. QR-koder hade fått sitta jämnt fördelade över bussen, kanske till och med vid varje sittplats, så det hade krävts en mängd QR-koder. Detta eftersom det inte ska uppstå köbildning vid påstigning av bussen. Positivt är att QR-koder kan skrivas ut av vem som helst och på vanligt papper så det hade inte blivit dyrt mer än kostnaden för papper och bläck.

3.3 GPS

GPS räknas varken som trådlöst eller som icke-trådlöst. Detta eftersom de inte fungerar alls på samma sätt som Wi-Fi eller bluetooth och inte heller som NFC eller QR-kod.

Uppkopplingen med GPS hade fungerat som så att applikationen hade kunnat jämföra användarens position från telefonens GPS med bussarnas positioner via applikationen. På så sätt går det att räkna ut vilken buss användaren befinner sig på. Problem med detta är att GPS kostar mycket ur batterisynpunkt. Dessutom kommer det att vara svårt att ta reda på vilken buss som användaren befinner sig på om det är många bussar i samma område, exempelvis i centrum.

4 Teknisk bakgrund

4.1 JavaScript

JavaScript är ett cross-platform, objektorienterat scripting språk. Språket är minnessnålt, lätt att implementera och har en enkel syntax men det är inte användbart för sig själv utan är designat så att det ska vara lätt att ha ”inbäddat” (engelska: ”*embedded”*) i andra produkter och applikationer, exempelvis web browsers. I en värdmiljö kan JavaScript kopplas till objekt i dess miljö för att ge kontroll över dem.
Core JavaScript innehåller en kärna av objekt såsom Array, Data och Math. Det innehåller även språkelement såsom operatorer, kontrollstrukturer och satser.
Core JavaScript kan bli utökat för en mängd olika ändamål genom att komplettera det med ytterligare objekt, till exempel

- *Client-side JavaScript* utökar kärnan av språket med objekt för att kontrollera en browser (Navigator eller annan web browser) och dess Document Object Model (DOM). Till exempel kan utbyggnader på klient sidan tillåta att en applikation placerar element på ett HTML-formulär och svara på användarhändelser såsom musklick, input och navigation på sidan.
- *Server-side JavaScript* utökar kärnan av språket med objekt som är relevanta för att köra JavaScript på en server. Till exempel kan utbyggnader på serversidan tillåta att en applikation kan kommunicera med en relationsdatabas, ge kontinuerlig information från ett anrop till ett annat anrop i applikationen eller för att utföra filmanipulationer på en server.

Genom JavaScripts LineConnect-funktion, kan man låta Java- och JavaScript-kod kommunicera med varandra. Från JavaScript kan man skapa Java-objekt och komma åt deras publika metoder och fält. Från Java kan man referera till JavaScript-objekt och komma åt deras attribut och metoder.

Språket utvecklades av Netscape och användes från början i Netscapes browsers.
För hela denna rubrik, se referens [4].

4.1.1 Tillstånd (eller händelser, eng ”events”)

Det finns olika tillstånd som ett fönster kan befinnas sig i som man måste ta hänsyn till. Dessa är, *”open”, ”close”, ”focus”* och *”blur”*.
Open – Öppen. Tillstånd då fönstret är öppet.
Close – Stängd. Tillstånd då fönstret är stängt.

Focus – Fokus, eller Skärpa. Tillstånd då ett öppet fönster är i fokus, det vill säga att det är det fönstret som ”är i visning” eller det fönster som syns för tillfället.
Blur – Oskärpa. Tillstånd då ett öppet fönster inte är i fokus.
Om ett fönster inte stängs manuellt så förblir det öppet men hamnar i oskärpa om man trycker vidare från det eller om man backar från det.
Man kan manuellt stänga och öppna fönster, men focus och blur kan ej åstadkommas manuellt. [2]

Då ett fönster ändrar tillstånd skickas ett event med namnet på det nya tillståndet. När andra fönster tar emot ett sådant event, om de lyssnar efter det, kan de agera därefter. En närmare förklaring finns i avsnitt 4.1.2 nedan.

4.1.2 fireEvent och addEventListener

I detta examensarbete används metoderna fireEvent och addEventListener två flitigt. Varje objekt i JavaScript förses med metoderna

fireEvent(”vad eventet ska heta”, {vad som ska skickas med});

och

addEventListener(”vilket event lyssnaren ska lyssna på”, function(e){
vad som ska hända då detta event inträffar
e är det som skickats med från fireEvent
});

Det största användningsområdet i applikationen är för att händelser i olika fönster ska kunna reagera på varandra. Exempelvis, då användaren från första fönstret väljer en ny busslinje så finns det en lyssnare som lyssnar på eventet ”click” i ”row”. Det första fönstret består av en enda stor tabell, med två sektioner. I den första sektionen finns endast en rad och den raden är ”Ny linje”. Sektion två består av senast valda linjer, och är max 10 rader lång.

```
row.addEventListener('click', function(e) {
            self.fireEvent('newSelected', {})
});
```

Då detta hänt skickas ett nytt event ”newSelected” till ”self” som är det egna fönstret.
Sedan lyssnar någon annan på om just detta fönster får ett event newSelected och vet då att användaren tryckt på ny linje och varifrån applikationen ska navigera efter det, i flesta fall med nya fireEvents och eventListeners.

4.2 Titanium

Titanium är en öppen och flexibel utvecklingsmiljö för att skapa applikationer för olika mobilenheter och operativsystem. Dessa inkluderar iOS, Android, Windows och Blackberry, men även hybrider och HTML5. Titanium bygger på en Eclipse-baserad IDE (Integrated Development Environment) och språket som används är JavaScript. Därför är JavaScript det enda språk som behövs för att kunna utveckla applikationer i Titanium. [6]

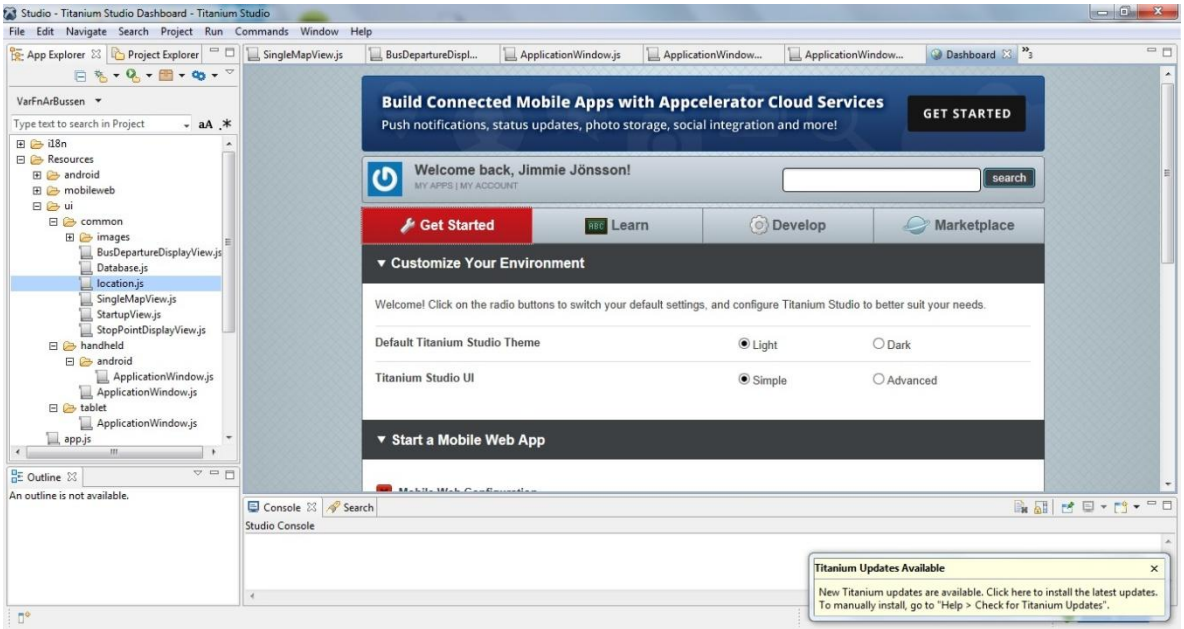
"It's very hard to find different native skills and combine them in one team, but it is comparatively easy to educate people in JavaScript."

"Det är väldigt svårt att hitta olika kompetenser för olika plattformar och kombinera dem i ett arbetslag, men det är förhållandevis lätt att utbilda folk i JavaScript"

– Marjin Deurloo,
CEO, imgZine

Dock måste man, utöver JavaScript, lära sig Titanium API (Application Programming Interface).

För Android och Blackberry användas Mozillas "Rhino", som är JavaScript implementerad i Java och för iOS används Apples JavaScriptCore. [5]



Startsidan för utvecklingsverktyget Titanium.

4.3 JSON

JSON (JavaScript Object Notation) är ett minnessnålt datautbytesformat som både är lätt för människor att läsa och skriva, och som är lätt för datorn att generera och tolka. [7]
I applikationen används olika http-förfrågningar för att få fram information om exempelvis bussar. Sidan som kommer som svar ser då ut på följande sätt:

```
{
  "latestUpdate":"10:19:39",
  "buses":[
    {
      "id":6187,
      "longitude":13.503630345383,
      "latitude":59.38054573976,
      "dateTimeStamp":"2013-04-19 10:19:28",
      "getData":"http:\/\/[Här fanns tidigare URL]
\/bussfan\/getBusDepartues.php?busID=6187"
    }
    (Här kan följa fler "bussar")
  ]
}
```

I detta fall endast med en buss som svarsdata från http-förfrågningen.
När förfrågan gjorts måste informationen som finns i svaret läsas ut och här används JSON, som följer:

```
var data = JSON.parse(this.responseText).buses;
```

Där *this.responseText* är hela texten som fås ut från förfrågningen (svaret). Sedan läses i detta fall ut det fält som döpts till ”buses” och läggs i en variabel ”data”. Efter detta är gjort kan ”data” stegas genom med en for-loop med samma längd som själva variabeln (fås av *data.length*), och skapa en notation för varje buss. I varje notation anges longitud, latitud och id och sedan kan denna notation placeras ut på en karta. I denna applikation används något som kallas restdata, det vill säga att utifrån varje svar från http-förfrågningar kan det fås ut ännu en http-förfrågning och därför behövs det inte skrivas in en URL varje gång, utan det räcker med att, i just detta fall, skriva *data[0].getData*. Då fås URLen för att få information om alla avgångar för bussen med just det ID’t som fanns i *data[0].getData*. (*data[0]* innehåller all information om den första bussen, *data[1]* innehåller all information om den andra bussen, osv., som lästs in men hjälp av JSON.parse.)

4.4 Hostapd

Hostapd är ett tredjepartsprogram som tillåter en dator som kör Linux att kunna skapa en AP (Access Point). Det enda som krävs är att nätverkskortet i datorn, och drivrutinen till nätverkskortet, stödjer ett så kallat ”*master-mode*” och att operativsystemet är byggt på en Linuxkärna. [17]

4.5 Bluetooth

Bluetooth är en enkel och säker teknik med kort räckvidd som kan användas överallt. Bluetooth finns i en mängd enheter som sträcker sig alltifrån mobiltelefoner och datorer till medicinska enheter och hemmaunderhållning. Syftet med Bluetooth är att det ska ersätta kablar men fortfarande hålla en hög nivå av säkerhet. De viktigaste egenskaperna hos Bluetooth är robusthet, låg effekt och låg kostnad.
Att två bluetoothaktiverade enheter ansluter till varandra kallas *”pairing”*, ”parning”. Strukturen i Bluetooth, och den globala acceptansen av Bluetooth, medför att alla Bluetoothaktiverade enheter, nästan överallt i världen, kan anslutas till andra Bluetoothaktiverade enheter som är inom räckvidd för varandra. Anslutningar mellan Bluetoothaktiverade enheter tillåter dessa enheter att kommunicera trådlöst genom kort räckvidds ad-hoc nätverk, kallat piconets. Piconets etableras dynamiskt och automatiskt allt eftersom Bluetoothaktiverade enheter tillkommer eller lämnar nätets täckningsområde. Varje enhet i ett piconet kan samtidigt kommunicera med upp till sju andra enheter inom samma piconet och varje enhet kan tillhöra flera piconets samtidigt. Detta betyder att den mängd olika sätt att koppla ihop Bluetooth enheter är nästan gränslösa.

En grundläggande styrka i Bluetooth tekniken är dess förmåga att samtidigt hantera data- och röstförbindelser, vilket ger användare en mängd innovativa lösningar såsom hands-free till telefonsamtal, skrivare och fax, synkronisering mellan datorer och mobiltelefoner.

Räckvidden för Bluetooth beror på applikationen. Bluetooths kärnspecifikation ger en minsta räckvidd på 10 meter, men det finns ingen fast gräns och tillverkare kan förbättra sina implementationer för att tillhandahålla en längre räckvidd. Räckvidden varierar beroende på vilken radio som används vid implementationen.

- Klass 3 radio har en räckvidd på upp till 1 meter.
- Klass 2 radio, den som är vanligast i mobiltelefoner, har en räckvidd på 10 meter.
- Klass 1 radio, används främst inom industriella lösningar, har en räckvidd på 100 meter.

Bluetooth kommunicerar mellan frekvenserna 2,4 GHz och 2,485 GHz. Detta frekvensband har blivit reserverat av internationella avtal för användning av industriella, vetenskapliga och medicinska enheter (ISM, Industrial, Scientific and Medical devices). Detta band är olicensierat i de flesta länder.

Bluetooths teknik, adaptiv frekvenshoppning (AFH, adaptive frequency hopping) var designad för att reducera störningar mellan trådlösa tekniker som delar 2,4 GHz spektrumet. AFH arbetar inom spektrumet och drar fördel av de lediga frekvenserna. Detta görs med en teknik som detekterar andra enheter i spektrumet och undviker frekvenserna som de använder. Denna adaptiva hoppning mellan 79 individuella slumpvis valda frekvenser med 1 MHz intervall ger en hög störningsimmunitet och tillåter mer effektiv transmission inom spektrumet. För Bluetoothanvändare ger denna hoppning bättre prestanda även när andra tekniker används tillsammans med Bluetooth.

Bluetooth är designat för att ha en väldigt låg effektkonsumering. Detta sker genom att radion tillåts att stängas av när den är inaktiv. Radion startas endast när dataöverföring ska ske.
För hela denna rubrik, se referens [8].

4.6 Wi-Fi

Wi-Fi använder en radioteknik som kallas 802.11 för att tillhandahålla en säker, tillförlitlig och snabb trådlös uppkoppling. Ett Wi-Fi nätverk kan användas för att koppla ihop enheter med varandra, till Internet och till trådbundna nät som använder Ethernet-tekniken. Wi-Fi-nätverk opererar i 2.4 och 5 GHz radiobanden, med några produkter som innehåller båda band (dual band). Det kan förse ett nätverk med liknande prestanda som vanliga grundläggande trådbundna nätverk.

Idag kan Wi-Fi produkter göra allt mellan att sända ett email till att streama video och visa internationella videokonferenser, det kan tillhandahålla Internet från ett plan 3 km upp i luften eller bara ett rum bort.

Wi-Fi-produkter opererar i 2.4GHz eller 5GHz bandet. Dessa band är licensfria, vilket innebär att individer får använda produkter som är gjorda för dessa nät utan en regeringslicens. Eftersom Wi-Fi banden är licensfria så blir det av större och större vikt för tillverkare att försäkra sig om att deras produkter uppfyller de standarder för inoperabilitet som fastställs av Wi-Fi-certifieringar. Och eftersom de också delar band med icke-Wi-Fi produkter, såsom radiostyrda leksaker, så försäkrar certifieringstester att Wi-Fi produkter är ”bra grannar” och kommer inte att störa signaler från dessa enheter.

Produkter från olika företag ska kunna arbeta tillsammans, så att användare inte blir låsta till ett enda märke av Wi-Fi produkter. Detta kallas ”Interoperabilitet”. The Wi-Fi Alliance har drygt 500 medlemmar och vilken som helst av dessa företag kan skicka produkter till certifiering, vilket betyder att det finns ett stort urval av produkter, och användare kan blanda mellan

olika produkter från olika företag och ändå vara säker på att de ska fungera tillsammans.

Dagens Wi-Fi-produkter är betecknas med ett Dewey Decimal-liknande system, som är utvecklat av IEEE (Institute of Electrical and Electronic Engineers, känt som ”I-triple E”) för att skilja mellan olika tekniska familjer. Wi-Fi produkter är identifierade som 802.11, och sedan ytterligare identifierat med en gemen bokstav, som identifierar vilken specifik teknik som används, exempelvis 802.11a. Bokstäverna definierar vilka egenskaper som finns, relaterat till prestanda, frekvens och bandbredd. Varje generation utvecklar också mer säkerhet och kan ha med andra funktioner som tillverkarna kan ha valt att implementera.
För hela denna rubrik, se referens [12].

4.7 NFC

NFC-tekniken tillåter mobiltelefoner med NFC att kommunicera med andra enheter som har en NFC-tag. Det räcker att dra sin telefon förbi en annan tag, i till exempel kön i en affär eller att hålla den mot sin kompis telefon för att dela det senaste spelet.
NFC liknar Bluetooth och Wi-Fi på ytan, alla tre är trådlös kommunikation och datautbyte mellan digitala enheter, exempelvis mobiltelefoner. Men NFC utnyttjar elektromagnetiska radiofält medan tekniker som Bluetooth och Wi-Fi istället använder radiotransmissioner.
NFC är en utlöpare (offshoot) av radio-frekvens identifikation (RFID, radio-frequency identification) med undantaget att NFC är designat för användning av enheter som är inom nära räckvidd till varandra.

Det finns tre olika sorters NFC tekniker, Typ A, Typ B och FeliCia. De är alla lika men kommunicerar på något annorlunda sätt. FeliCia finns först och främst i Japan.

Enheter som använder NFC kan vara aktiva eller passiva. En passiv enhet, såsom NFC-taggar, innehåller information som andra enheter kan läsa, men kan inte läsa någon information själv. Man kan tänka sig en passiv enhet som en skylt, andra kan läsa skylten, men skylten i sig gör ingenting förutom att förmedla informationen till auktoriserade enheter.
Aktiva enheter kan både läsa och sända information. En aktiv NFC-enhet, som en mobiltelefon, kan inte bara hämta information från NFC taggar, utan kan även utbyta information med andra kompatibla mobiltelefoner eller andra enheter och kan även ändra information i NFC-taggar, om enheter har den auktoriteten att göra sådana ändringar.

För att försäkra sig om säkerheten, upprättar NFC ofta en säker kanal och använder kryptering då känslig data, såsom kredit- och kortnummer, skickas. Användare kan skydda privat data ytterligare genom att ha ett anti-virus program aktiverat i sin telefon, och även se till att ha en kod till telefonen så att eventuella tjuvar inte kan använda den i händelse att telefonen blir stulen eller borttappad.
För hela denna rubrik, se referens [10].

4.8 QR-kod

QR-kod står för Quick Response Code är en streckkod som har två dimensioner, det vill säga att den kan lagra information både vertikalt och horisontellt. Medan vanliga streckkoder kan ha ungefär 20 siffror, så kan QR-kod, beroende på version, ha flera dussin till flera hundra gånger mer information.

QR-koder är kapabla att hantera alla typer av data, såsom numeriska och alfabetiska tecken, men även Kanji, Kana, Hirigana, symboler, binär och kontrollkoder. Upp till 7089 tecken kan kodas i en QR-symbol.
Eftersom QR-koder kan lagra information både vertikalt och horisontellt, så kan QR-koder lagra samma mängd data på en tiondel av utrymmet av en traditionell streckkod. QR-koder har även felkorrigering. Data kan bli återställt även ifall en symbol är delvis smutsig eller skadad.
Tack vare de tre större fyrkanterna, kallade positioneringsdetekteringsmönster, i de tre hörnen så kan QR-koden läsas från alla håll, i 360 grader.
För hela denna rubrik, se referens [19]

QR-kod

4.9 GPS

GPS står för Global Positioning System och ägs av USA. GPS tillhandahåller positionering, navigation och timing tjänster. Systemet består av tre segment, rymdsegmentet, kontrollsegmentet och användarsegmentet. U.S Air Force har utvecklat och färvaltar på rymd- och kontrollsegmenten.

Rymdsegmentet består av en så kallad stjärnbild av satelliter som sänder radiosignaler till användare. U.S Air Force förvaltar stjärnbilden för att åtminstone 24 satelliter ska vara tillgängliga 95% av tiden. Under de senaste åren har det funnits 31 operativa satelliter och 3-4 nedlagda satelliter, som kan bli återaktiverade om det behövs.

Kontrollsegmentet består av ett nätverk av markstationer som spårar GPS-satelliterna, övervakar deras sändningar, genomför analyser, och skickar kommando och data till stjärnbilden. För tillfället finns det en ”master control station”, en alternativ ”master control station” 12 kommando- och kontrollantenner och 16 övervakningsplatser.

Användarsegmentet är det segment som vi upplever. GPS i telefoner, uttagsautomater, vägarbetsfordon, bussar, och så vidare.
För hela denna rubrik, se referens [18]

5 Resultat

Efter analys av de olika alternativen för att handskakningen ska bli möjlig blev valet att använda sig av Wi-Fi. Anledningen är att det känns som det mest smidiga sättet att göra hanskakningen möjlig, eftersom de flesta användare vet hur Wi-Fi fungerar och eftersom då nätverket skapas går det att själv välja SSID.

Anledningen till att Bluetooth valdes bort är för att Bluetoothservern måste erbjuda tjänster beroende på vilket UUID man vill åstadkomma och för att även om Bluetooth inte är dyrt batterimässigt att ha igång, så är det fortfarande dyrt då det skannar efter enheter. Hade det gått att välja ett UUID själv så hade Bluetooth varit ett mer lockande alternativ.

QR-koder valdes bort eftersom de lätt kan bli utsatta för vandalism, övermålade eller förstörda.

NFC valdes bort för att det inte finns i tillräckligt många telefoner idag. NFC hade kunnat vara ett utmärkt val om ett par år. Eftersom en NFC tag inte kan ändras om man en gång valt att låsa det, kan ingen göra ändringar och därför är det en säker teknik. Tekniken är också väldigt snabb och sker genom att bara föra telefonen förbi taggen.

GPS valdes bort eftersom det inte var tillräckligt precist, skulle användaren befinna sig nära två olika bussar är det svårt att bestämma vilken buss det är som gäller genom att använda GPS. Ibland är det dessutom svårt att få sin egen position som användare och därför hade det inte gått att räkna ut vilken buss användaren befinner sig på.

Så valet landade alltså på Wi-Fi. Det visar sig att Android inte stödjer Ad-Hoc nätverk, utan för att en Androidenhet ska kunna ansluta till en dator behöver datorn fungera som en AP (Access Point). För att fordonsdatorn, som är Linux Ubuntu, ska klara av detta krävs dels att nätverkskortet i sig stödjer ett så kallat ”*master mode”* eller *”Access point mode”* och dels att själva drivrutinen till kortet också stödjer dessa.

Vid första försöket visade det sig att detta inte fungerar på den existerande fordonsdatorn, men efter installation av ett tredjepartsprogram, kallat ”Hostapd” fungerade det. Då kan alltså fordonsdatorn fungera som en WAP (Wireless Access Point) och även Androidenheter kan ansluta med Wi-Fi.

I Titaniums API finns det ingen modul för att hantera och kontrollera Wi-Fi. Därför utvecklas en modul som skannar efter Wi-Fi-nätverk. På

http://www.androidsnippets.com/scan-for-wireless-networks fanns en androidsnippet (”kod-snutt”) för att skanna efter trådlösa nätverk. Denna snippet byggdes ut till att leta efter nätverk vars SSID ser ut på ett visst sätt (Exempelvis börjar med ”Karlstadsbuss”) och läser sedan ut det som står efter. Denna modul ska läggas till i Titanium så att den används då skanning efter tillgängliga nätverk är nödvändig. Modulen är skriven i Java och utvecklad i utvecklingsverktyget Eclipse Indigo.

Förutom analysen om vilken teknik som fungerar bäst för handskakningen mellan applikationen och bussen har det även utvecklats en större del av själva applikationen ”Var f.n är bussen?”. Koden ärvdes av en anställd på företaget som hade börjat utveckla den, men slutat då han fick ont om tid för applikationen. Då examensarbetet startade fanns applikationen i en förstaversion till iPhone. De första fyra fönstren fanns redan, med annorlunda utseende och innehåll. Eftersom applikationen var utvecklad till iPhone fanns ingen bakåtknapp utan det gick endast att navigera ”framåt” i applikationen.

Då projektet togs över framgick det att Trivector helst ville ha applikationen till Android. Det hade testats att göra till Android också men de hade funnit att Android inte stödjer två kartobjekt i en och samma applikation, vilket iPhone gör (se referens [1]]. Därför var det tvunget att samma kartobjekt användes i både fönster 2 och i fönster 4. Det var ett stort problem. Dessutom fanns där en ”bugg” i fönster tre, skylten för hållplatser. Då det skulle läsas in nya hållplatser i tabellen rensades först tabellen med ett anrop som såg ut på följande sätt:
table.setData([]), där table är själva tabellen som innehåller raderna med busslinjer. Detta gjordes precis innan nya avgångar lästes in. I anropet försöks det alltså att sätta innehållet i tabellen till en tom array, alltså tömma tabellen helt. Detta anrop kastade ofta, men konstigt nog inte alltid, en nullPointerException. Detta var det första stora problemet. Efter många timmar på www.google.se och Appcelerators forum upptäcktes det att det var fler som hade problem med just detta anrop. Tydligen var det så att för att kunna tömma tabellen med table.setData([]) var fönstret som innehåller tabellen tvungen att vara i fokus (se 4.1.1 Tillstånd). Det visade sig alltså att även om anropet att läsa in nya avgångar gjorts (och därmed också först tömt tabellen), och även om fönstret innan dess egentligen hade öppnats så hade fönstret inte hunnit hamna i tillståndet ”focus” innan anropet för att tömma tabellen gjordes. Detta löstes genom att göra en metod vars enda uppgift är att anropa table.setData([]) och sedan kalla på denna metod då användaren trycker på ”back” eller på en linje från detta fönster. Då tömdes alltså tabellen så fort användaren navigerade vidare från fönstret som innehåller tabellen och alltså innan fönstret hann tappa ”focus”.

Anledningen att det krävdes en metod för att tömma tabellen var för att det är ett annat fönster lyssnar på vart användaren vill navigera (back, eller vidareklick) med en så kallad EventListener.(se 4.1.2 fireEvent och addEventListener).

Problemet med att Android inte tillåter mer än ett kartobjekt löstes genom att det helt enkelt fick användas endast ett kartobjekt, som fick delas mellan två fönster. När användaren navigerar vidare från en kartvy tas kartobjektet bort från den kartvyn och kartvyn stängs. Det skapas även två stycken arrayer, en array som innehåller alla hållplatser som ska ritas i den första kartvyn och en array som innehåller alla hållplatser som ska ritas i den andra kartvyn. Det skapas utöver arrayerna ett ruttobjekt (route object) som ritas i den andra kartvyn. Bussarna behövs inte sparas på samma sätt eftersom de ritas ut med 5 sekunders intervall och att de endast ritas i den andra kartvyn.

Anledningen till att fönstret stängs igen var för att det uppstod problem då användaren ska navigera bakåt till en kartvy för då finns där inget kartobjekt. Alltså ska kartobjektet läggas till igen, inga problem, och sen ska den kartvyn få fokus igen men här fungerar det inte. I JavaScript ska det gå att anropa fönsternamn.focus() för att det fönstret ska hamna i fokus men av någon anledning fungerade inte det i detta fall. Av denna anledning var fönstret tvunget att stängas endast för att kunna öppnas igen då användaren trycker back.

Navigering med back fungerar perfekt i Android från början, med Androids back stack, som är en stack som är ordnad efter vilken ordning olika aktiviteter (activities, som är fönstren i detta fall) öppnades. När en aktivitet öppnar en annan så blir den nya aktiviteten ”pushed” till toppen av stacken och får fokus. Den föregående aktiviteten finns kvar i stacken, men är stoppad. När en aktivitet stannar så behåller systemet dess nuvarande tillstånd. När en användare trycker på back-knappen, så blir den nuvarande aktiviteten ”popped” från toppen av stacken och den föregående aktiviteten fortsätter. Aktiviteter i aktivitetsstacken kan aldrig bli omflyttade utan bara pushade eller poppade. Pushade ovanpå stacken när aktiviteten startas av nuvarande aktivitet och poppade av från stacken då användaren lämnar aktiviteten med back-knappen. På så sätt jobbar back stacken som en lifo-kö, last in, first out. [3]

Men back stacken fungerar tyvärr inte om kartobjektet tas bort från kartvyn, eftersom kartobjektet isåfall måste tas bort innan aktivitetsbyte sker. Görs det på det sättet så kommer det alltså gå alldeles utmärkt att backa till en kartvy, men det kommer inte att finnas något kartobjekt att visa. I Titanium går det att lyssna till Androids hårdvaruknapp back med addEventListener(”android:back”, function(e){}); på varje enskilt fönster.

Från början var det tänkt att problemet kunde lösas på detta sätt, att bara lägga till kartobjektet igen så fort användaren tryckte back. Problemet här är att ifall utvecklaren väljer att lyssna på eventet android:back så används det inte någon backstack längre, utan då får utvecklaren själv definiera vad som ska hända vid tryck på back. På grund av detta blev det nödvändigt att helt definiera vad som ska ske då användaren navigerar, vilka fönster som måste öppnas och vilka som måste stängas. Vilka data som måste sparas för att kunna göra anrop igen då användaren backar och vilka event som måste lyssnas på. Dessutom fick det läggas till variabler för att veta ”från vilket håll” användaren navigerar till varje fönster, för att veta vilken data det är som ska ritas på kartan. När detta hade genomförts till de två fönster som behövde det, kartvy ett och kartvy två, så visade det sig att då var det tvunget att göras för de andra fönstren också, eftersom kartvyn var tvungen att läggas till igen då användaren backar från fönster också, med andra ord blev det nödvändigt att definiera android-back för alla fönster som skulle kunna ge back till en kartvy.

Det finns även ett event som heter ”blur”, det vill säga när en aktivitet tappar fokus. Tanken var då att istället för att lyssna på ”android:back” för att ta bort kartobjektet från kartvyn så går det att lyssna på ”blur”. I båda fallen kommer kartobjektet tas bort då användaren backar från en kartvy. Dessutom kommer det på ”blur” alternativet även ta bort kartobjektet från en kartvy då användaren trycker vidare ”framåt” i applikationen. Det behövs eftersom kartobjektet måste tas bort från kartvy ett innan det kan användas i kartvy två. ”Blur” är alltså ett mycket bättre alternativ än ”android-back”, dels för att backstacken finns kvar intakt och dels för att det då bara behövs lyssna på ett event, ”blur”, istället för både på ”android-back” och på vidarenavigering i applikationen.

På liknande sätt finns ett event ”focus” som kunde ha används till när användaren backar till en kartvy. Så fort kartvyn får fokus, lägg till kartobjektet igen.

Anledningen till att ”android-back” används istället för ”blur” och ”fokus” är att ”blur”-eventet inte skickades. Detta berodde på att när eventListeners ska läggas till så måste det göras innan fönstret öppnas för att kunna reagera på händelsen. Eftersom detta inte var känt under utvecklingsfasen så öppnades fönstret innan eventListenern lades till. Därför valdes en lösning där "android-back” användes under utvecklingsfasen. Att ändra detta så att ”blur” används i stället för ”android-back” finns med i avsnitt 7, Framtida utvecklingsmöjligheter.

Ett problem med att använda ”blur” istället för ”android-back” är att i backstacken sparas alla tidigare fönster som användaren befunnit sig i. Med

andra ord, om det finns ett ställe i applikationen där det finns möjlighet att ”loopa” mellan två eller flera fönster och det görs ett antal gånger, så kommer även backstacken att loopa genom alla dessa fönster på ”vägen tillbaka”.

6 Slutsats

Valet för handskakningen blev Wi-Fi, eftersom dess fördelar övervägde nackdelen att det är batterikrävande. Eftersom Android inte stödjer ad-hoc nätverk, som är standarden då en PC delar sitt nätverk, så måste ett tredjepart program användas för att datorn ska fungera som en accesspunkt istället för en enhet i ett ad-hoc nätverk. Det måste även skrivas en modul till Titanium eftersom det inte finns någon Titanium API för Wi-Fi. Modulen är en Wi-Fi skanner som ska skanna efter tillgängliga nätverk, för att kunna avgöra vilken buss en användare befinner sig på.

7 Framtida utvecklingsmöjligheter

Det skulle vara bra att utveckla en sorts reseplanerare som utbyggnad till denna applikation. Eftersom bussarnas tider är i realtid hade det därför varit möjligt att se försenade bussar och utifrån det räkna ut en ny rutt för användaren så att han eller hon hade sparat tid på att exempelvis byta buss på en hållplats där användaren inte annars hade bytt. Användaren hade fått välja utifrån två alternativ, antingen snabbaste resväg eller minsta antal byten.

Även om inte NFC var ett alternativ att räkna med i dagens läge så kanske det hade varit ett bättre alternativ om ett par år, beroende på hur snabbt tekniken utvecklas i exempelvis betalmöjligheter.

Det är också en god idé att ändra så att applikationen använder lyssnar på ”blur” och ”focus” event istället för ”android:back” (se 4.1.2 Tillstånd)

8 Terminologi

NFC	Near-Field Communication, eller Närfältskommunikation
QR-kod	Quick Response Code, är en streckkod som har två dimensioner, det vill säga att den kan lagra information både vertikalt och horisontellt
GPS	Global Positioning System, ett system bestående av satelliter för navigering och för att kunna ange positioner
SSID	Service Set Identifier, är en skiftlägeskänslig (”case sensitive) 32 alfanumeriska tecken lång unik identifierare som används för att skilja WLANs åt
UUID	Univeral Unique Identifier, är ett 128 bitar långt nummer för att identifiera objekt eller entiteter på Internet. Ett UUID är inte garanterat men extremt sannolikt att vara olik något annat UUID som skapas.
Dewey Decimalklassifikation	Dewey Decimalklassifikation är ett klassifikationssystem för böcker i bibliotek. Det är numera det mest spridda i världen.

9 Referenser

Titanium Map View API dokumentation
[1]http://docs.appcelerator.com/titanium/latest/#!/api/Titanium.Map.View
Hämtad 2013-03-03

Titanium Window API dokumentation
[2] http://docs.appcelerator.com/titanium/latest/#!/api/Titanium.UI.Window
Hämtad 2013-03-04

Android Back-Stack
[3] http://developer.android.com/guide/components/tasks-and-back-stack.html
Hämtad 2013-04-22

JavaScript
[4] https://developer.mozilla.org/en-US/docs/JavaScript/Guide/JavaScript_Overview
Hämtad 2013-04-14

[5] https://developer.mozilla.org/en-US/docs/JavaScript/About_JavaScript
Hämtad 2013-04-14

Titanium
[6] http://www.appcelerator.com/platform/titanium-platform/
Hämtad 2013-03-20

JSON
[7] www.json.org
Hämtad 2013-04-22

Bluetooth
[8] http://www.bluetooth.com/Pages/Basics.aspx
[9] https://www.bluetooth.org/en-us/specification/assigned-numbers-overview/service-discovery
Hämtad 2013-04-22

NFC
[10] http://www.nearfieldcommunication.org/
[11] http://rapidnfc.com/
Hämtade 2013-04-22

Wi-Fi
[12] http://www.wi-fi.org/

[13] http://lifehacker.com/5369381/turn-your-windows-7-pc-into-a-wireless-hotspot
Hämtade 2013-05-14

Conectify
[14] http://www.connectify.me/
Hämtad 2013-05-20

Hostapd
[15] http://wireless.kernel.org/en/users/Documentation/hostapd
[16] http://w1.fi/hostapd/
[17] http://www.cyberciti.biz/faq/debian-ubuntu-linux-setting-wireless-access-point/
Hämtade 2013-05-20

GPS
[18] http://www.gps.gov/systems/gps/
Hämtad 2013-05-18

QR-kod
[19]http://www.qrcode.com/en/about/
Hämtad 2013-05-18

10 Bilagor

10.1 Bilaga A

I applikationen finns åtta stycken filer, eller units. De är StartupView, Database, SingleMapView, StoppointDisplayView, BusDepartureDisplayView och tre stycken ApplicationWindow. Anledningen till att det är tre stycken ApplicationWindow är för att applikationen ska kunna bete sig olika beroende på vilket operativsystem som körs. De tre olika är till Android, iPhone och tablets(för både Android och iPhone).

Database är en databas som innehåller de senast valda hållplatserna. Detta för att användaren från första fönstret ska kunna välja en senaste och därför måste linjenummer, destination och direktion sparas undan.

SingleMapView är en vy som innehåller ett kartobjekt. I SingleMapView hämtas all data som ska ritas ut på hållplatserna, både för kartfönster ett och för kartfönster två. Så det finns en funktion som ritar hållplatserna till kartfönster ett. En funktion som ritar hållplatser och linjesträckning till kartfönster två. En funktion som ritar och uppdaterar bussar till kartfönster två. Dessutom finns det funktioner för att beräkna användarens position och vad som ska hända då användaren flyttar och zoomar kartan.

StoppointDisplayView är en vy som innehåller en tabell där avgångar visas. Här hämtas data för avgångarna ut, så fort en hållplats har valts. Denna vy fungerar på liknande sätt som en skylt på busstationer.

BusDepartureDisplayView är en vy som innehållet en tabell där nästkommande hållplatser visas. Här hämtas data för nästkommande hållplatser ut, så fort en buss har valts. Denna vy fungerar som skylten i en buss, fast inte bara nästa hållplats, utan alla nästkommande hållplatser på den valda linjen.

ApplicationWindow är motsvarande en monitor. Det är i ApplicationWindow som all kommunikation mellan de andra fönstren sker. ApplikationWindow lyssnar på händelser i alla fönster och skickar dessa vidare till andra fönster. Det är även i ApplikationWindow det definieras vad som ska hända då användaren trycker back.

I applikationen finns 5 olika fönster. (se Bilaga B)
StartupDisplayContainerWindow, som innehåller vyn StartupView.
SingleMapContainerWindow1 och
SingleMapContainerWindow2 som delar på vyn SingleMapView.

StoppointDisplayContainerWindow, som innehåller vyn StoppointDisplayView.
BusDepartureDisplayContainerWindow, som innehåller BusDepartureDisplayView.

StartupViewContainerWindow är det första fönster som visas då applikationen startas. Då användaren först kommer hit så kommer applikationen att använda GPS värden från telefonen. På Android finns det cachad position som kan vara gammal och därför görs det en koll på den cachade positionens timeStamp för att se åldern på positionen. Är den äldre än 10 minuter får användaren välja att antingen används den gamla datan eller få defaultvärde från Karlstad. Anledningen att det kollas om data är gammal istället för om GPS är aktiverat över huvudtaget är för att i Titaniums API finns det en funktion: `Ti.Geolocation.locationServicesEnabled` som kollar om någon locationService är igång, inte bara GPS. I Android finns det en passiv locationService som alltid är igång och därför kommer denna funktion alltid att returnera true, och därför är det inte möjligt att kolla om GPS är av eller påslagen.

I StartupDisplayContainerWindow går det sedan att välja mellan Ny linje eller en av de Senaste linjer man valt.
Om användaren väljer Ny linje kommer denne till kartvy1, SingleMapContainerWindow1, och där visas hållplatser inom 500 meters radie från användarens position.
Om användaren väljer en av Linjerna från Senaste så kommer denne till kartvy2, SingleMapContainerWindow2 och där visas linjesträckning, hållplatser längs linjen och var på linjen bussar befinner sig.

Vid back från StartupViewContainerWindow stängs applikationen av.

I SingleMapContainerWindow1 ser användaren alla hållplatser som finns inom 500 meters radie från användarens position.

Om användaren trycker på en hållplats kommer denne till StoppointDisplayContainerWindow och ser då alla linjer med linjenummer och destination som avgår från den valda hållplatsen och tider när de avgår.

Vid back från StoppointDisplayContainerWindow kommer användaren tillbaka StartupViewContainerWindow.

I StoppointDisplayView finns alla linjer som avgår från den valda hållplatsen, med linjenummer, destination och avgångstid.

Om användaren trycker på en linje sparas denna linje till Senaste och sedan kommer användaren till kartvy2, SingleMapContainerWindow2 där linjesträckning, hållplatser längs linjen och bussar längs linjen visas.

Vid back från StoppointDisplayView kommer användaren, beroende på vad förra fönstret var, antingen tillbaka till SingleMapContainerWindow1 eller till SingleMapContainerWindow2.

I SingleMapContainerWindow2 visas linjesträckning, hållplatser och bussar på den valda linjen.

Om användaren trycker på en hållplats kommer denne tillbaka till StoppointDisplayContainerWindow och får se avgångar från den nyligen valda hållplatsen.

Om användaren trycker på en buss kommer denne till BusDepartureDisplayContainerWindow där bussens kommande hållplatser och när de avgår från dessa hållplatser visas.

Vid back från SingleMapContainerWindow2 finns det två alternativ.
Om användaren kom till SingleMapContainerWindow2 via klick på en Senaste i startupViewContainerWindow så kommer back att navigera till StartupViewContainerWindow.
Om användaren kom till SingleMapContainerWindow2 via klick på Ny linje i StartupViewContainerWindow (och därmed även navigerade genom SingleMapContainerWindow1 och StoppointDisplayContainerWindow) så kommer back att navigera till StoppointDisplayContainerWindow med sparad data från senaste valda hållplats.

I BusDepartureDisplayContainerWindow visas alla nästkommande hållplatser och när bussen avgår från dessa för den valda bussen.

Vid tryck på back från BusDepartureDisplayContainerWindow kommer användaren tillbaka till SingleMapContainerWindow2

10.2 Bilaga B

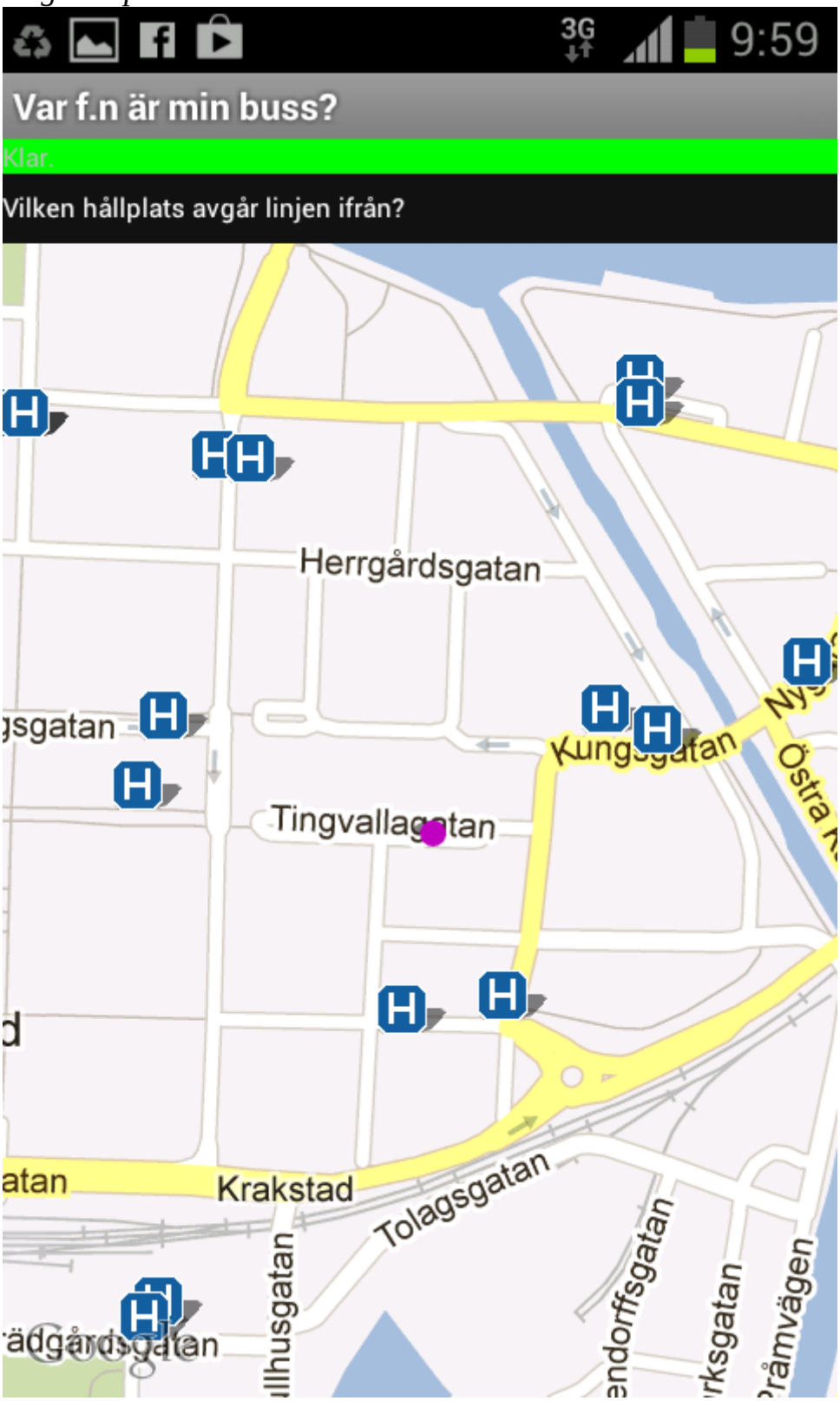
Här följer bilder från applikationens olika fönster.
StartupViewContainerWindow

Startfönster för applikationen. I Android finns cachade positionsdata och därför finns förfrågan om man vill använda GPS värden som är äldre än 10min.

StartupViewContainerWindow

Första fönstret på applikationen.

SingleMapContainerWindow1

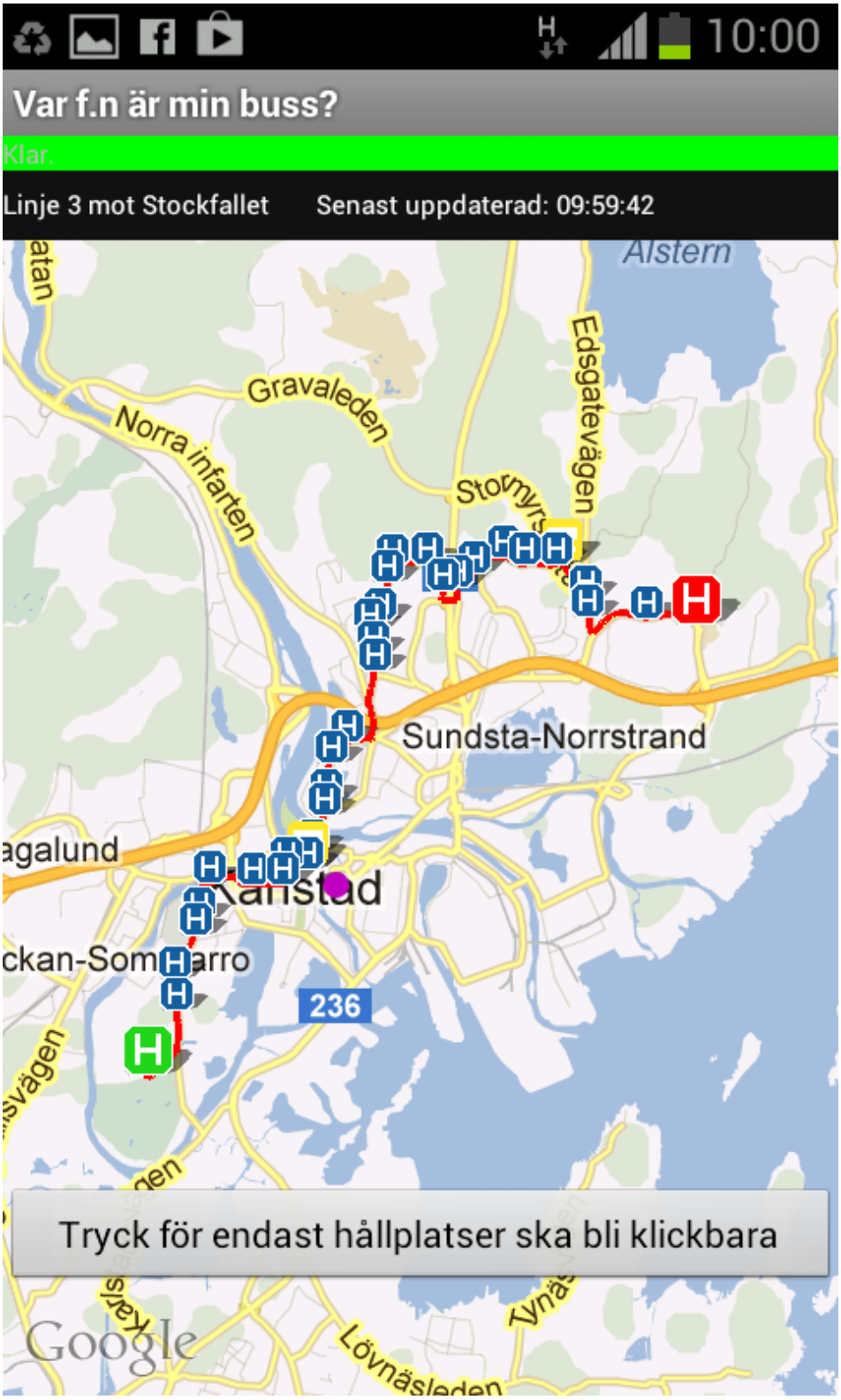


Andra fönstret på applikationen, vita H'n på blå bakgrund är hållplatser. Den lila pricken är var användarens GPS position är. Här väljer man vilken hållplats bussen avgår ifrån.

StoppointDisplayContainerWindow

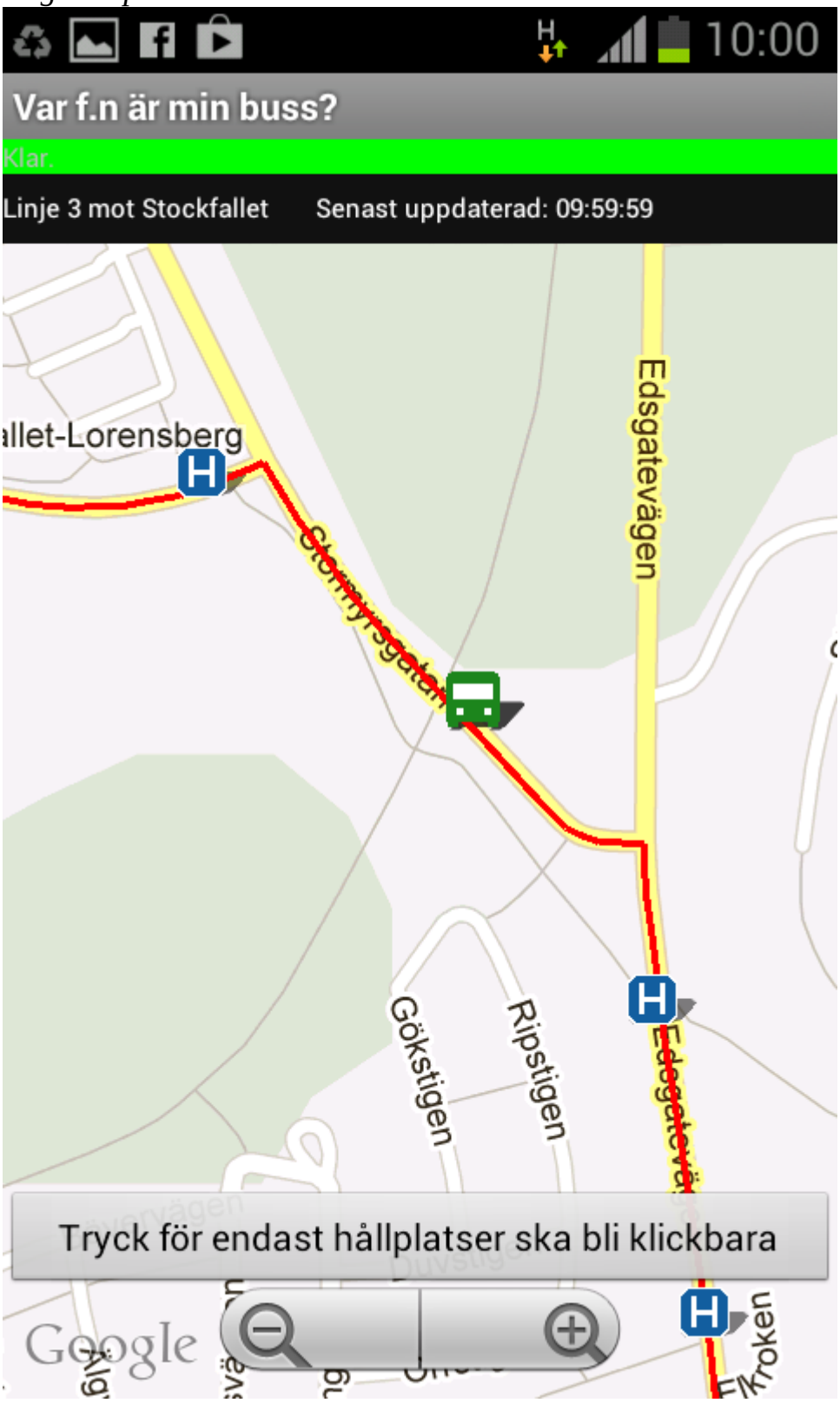
Tredje fönstret i applikationen. Här har användaren valt hållplatsen Stora Torget.

SingleMapContainerWindow2



Fjärde fönstret i applikationen. Här har användaren från förra fönstret valt ”Linje 3 mot Stockfallet”. Starthållplats i grönt, sluthållplats i rött. Gröna bussar är i tid, gula bussar är försenade. Från detta fönster kan man välja om hållplatser eller bussar ska vara klickbara.

SingleMapContainerWindow2



Inzoomad på en buss.

BusDepartureContainerWindow

Femte och sista fönstret. Här har användaren klickat på en buss och ser då kommande hållplatser för den bussen, och när bussen beräknas anlända dit.