CHALMERS

Sim Goods Reception

Utveckling av plattformsoberoende mobilapplikation

Examensarbete inom data- och informationsteknik

Erik Hermansson
Olof Spetz

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

EXAMENSARBETE

SIM Goods Reception

Utveckling av plattformsoberoende mobilapplikation

Erik Hermansson
Olof Spetz

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2015

Sim Goods Reception
Utveckling av plattformsoberoende mobilapplikation
Erik Hermansson
Olof Spetz

Examinator: Lars Svensson

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 100

Institutionen för data- och informationsteknik
Göteborg 2015

Sammanfattning

Apper Systems är ett IT-konsultföretag med fokus på administrativ systemutveckling. En av deras mjukvaruprodukter är Supply Integration Manager (SIM). SIM används för kommunikation mellan leverantörer och kunder. En del av leveransprocessen är mottagningen av gods. Detta sker idag med hjälp av skanningsenheter. Dessa är ofta dyra och kräver underhåll.

Syftet med examensarbetet var att utveckla en plattformsoberoende mobilapplikation som kan vara ett alternativ till skanningsenheter och integreras med SIM. Med plattformsoberoende utveckling kan man nå flera plattformar med en applikation. Detta effektiviserar arbetet, men det finns många olika utvecklingsmetoder och det är svårt att välja rätt. En kortare utvärdering av utvecklingsalternativ utfördes för att avgöra vilket som skulle användas under projektet. Applikationen utvecklades med webbtekniker som HTML, CSS och JavaScript. Den har stöd för följande plattformar: Android, IOS och Windows Phone 8.

Rapporten beskriver genomförandet av utvärderingen och utvecklingen av applikationen. Resultatet av utvärderingen jämför sex olika utvecklingsalternativ, ett av dessa används för att utveckla applikationen. Resultatet är en applikation som fungerar enligt kravspecifikation.

Nyckelord: Plattformsoberoende, Ionic, Cordova, app, applikation

Abstract

Apper Systems is a IT consultant company which focuses on administrative system development. One of their software products is Supply Integration Manager (SIM). SIM is used for communication between suppliers and customers. A part of the delivery process is the recieving of goods. This is done with the use of scanning units. Scanning units require maintenance and can be expensive.

The purpose of this project was to develop a cross-plattform mobile application which can be an alternative to scanning units and integrate with SIM. With cross-platform development a single application can reach multiple platforms. This streamlines the work process but there are many different developement alternatives and it can be hard to choose. A short evaluation of development alternatives was performed to determine how the application would be developed. The application was developed using web technologies such as HTML, CSS and JavaScript. It has the following supported platforms: Android, iOS and Windows Phone 8.

This thesis report describes the performed evalution and the development of the application. The resulting evaluation compared six different solutions, one of these were chosen to develop the application. The resulting application works in accordance to the requirements.

This report is written in Swedish.

Keywords: Crossplatform, Ionic, Cordova, app, application

Förord

Detta examensarbete har utförts under våren 2015 vid institutionen för Data- och Informationsteknik, Chalmers Tekniska högskola.

Vi vill tacka Apper Systems AB för möjligheten att göra detta projekt och för all hjälp under arbetets gång. Vi vill även tacka Joachim von Hacht för handledning och stöd med rapportskrivande.

Erik Hermansson och Olof Spetz
Göteborg, Juni 2015

Ordlista

AngularJS: AngularJS är ett JavaScript-klientramverk för webbapplikationer utvecklat av Google. AngularJS har som mål att förenkla utveckling och testning av webbapplikationer. Detta uppnås genom användning av MVC-arkitektur (Model-View-Controller) samt inbyggda funktioner. AngularJS är en öppen programvara.

API: Application Programming Interface. Ett gränssnitt mellan ett bibliotek och en applikation.

Backlog: En lista med arbetsuppgifter baserade på användarfall och funktionalitet som skall implementeras.

CLI: Command Line Interface. Kommandotolken, ett textbaserat gränssnitt där användare kan skriva in kommandon och interagera med program.

CSS3: Cascading style sheet definierar utseende för en HTML-sida. CSS skapades 1997 med avsikten att dela upp innehåll (HTML) och visuell design (CSS) [1]. CSS är tillsammans med HTML och JavaScript en viktig del av modern webbutveckling.

DOM: En intern representation av en HTML-sida i form av ett parse-träd samt ett antal API:er för manipulering av trädet.

Git: Ett distribuerat versionshanteringssystem för filer.

GitHub: En webbtjänst som agerar lagringsserver och distributionscentral för Git-repon.

GUI: Grapical User Interface. Det grafiska gränssnittet som användaren ser och interagerar med.

HTML5: HTML5 är en vidareutveckling av HTML och XHTML. HTML5 är avsedd att ersätta HTML 4 och XHTML 1, som ofta kräver tilläggsprogram för full funktionalitet. HTML5 har stöd för att inkludera många olika sorters multimedia utan att använda insticksprogram. Detta ökar både tillgänglighet och säkerhet. HTML5 är även bakåtkompatibelt mot äldre webbläsare.

IDE: Integrated Development Enviroment är ett program för mjukvaruutveckling. Innehåller oftast en texteditor, kompilator och avlusare.

JavaScript: JavaScript är ett objekt baserat skriptspråk som bland annat används för att manipulera DOM-trädet i webbläsaren. JavaScript stöds av alla moderna webbläsare. Genom att använda JavaScript blir webbsidan mer interaktiv och tillåter dynamiska uppdateringar av hela sidan eller delar av den. JavaScript filer laddas ner automatiskt vid sidförfrågning och förvaras i webbläsarens cache.

JSON: JavaScript Object Notion. JSON är ett textbaserat JavaScript objekt-baserat format för att sända och ta emot data på ett strukturerat sätt.

MVC: Ett designmönster som delar upp en applikation i model (data), view (presentation) och controller som sköter kommunikation mellan model och view. Avsikten är att separera logiken från gränssnitt.

Ramverk: Ett ramverk är en struktur med en samling tjänster som används inom applikations- utveckling. Ramverket lägger en grundstruktur för projektet och ökar abstraktionsnivån, vilket låter utvecklaren fokusera på problemlösningen.

Renderings-motor:	Ett program som hanterar märkspråk med tillhörande data och formaterar denna information för visning på bildskärm.
REST:	En modell för en kommunikationsstruktur mellan klient och server. Kommunikationen görs oftast via vanliga HTTP kommandon som GET, POST och DELETE. Typen av kommando i kombination med adress anger vilken typ av funktion som utförs
RESTful:	Ett system som tillämpar REST.
Scrum:	Ett arbetssätt inom mjukvaruutveckling. Utvecklaren använder sig av en backlog med uppgifter som bearbetas under sprintar.
Sprint:	I scrum delas utförandet av arbete upp i sprintar. De är tidsbestämda och består av planering, utförande av arbete, presentation och till slut en granskning av hela sprinten.
SDK:	Sofware Development Kit. En plattformsberoende samling av utvecklingsverktyg.
Webview:	En generell term som innefattar den interna webbläsaren som finns tillgänglig hos mobila plattformar. Den använder plattformens renderingsmotor för att visa webbsidor. Detta finns som standard i någon form hos de flesta smarttelefoner.
WireFrame:	En skiss över en webbsidas utseende/design.
User Story:	Ett användarfall som beskriver funktionalitet hos en applikation ur användarens perspektiv.
XML:	Extensible Markup Langauge. Ett märkspråk för informationsutbyte mellan olika system.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Examensarbetet utförs hos Apper Systems som är ett IT-konsultföretag med fokus på administrativ systemutveckling. Apper Systems erbjuder ett brett sortiment av programvaror och tjänster. Apper Systems är en del av Apper Group som är lokaliserat i Göteborg.

Supply Integration Manager, SIM är en av Apper Systems mjukvaruprodukter. Konceptet för SIM är att strukturera och assistera kommunikationen mellan leverantörer, transportörer och mottagare. Utifrån storleken på leverantörer och mottagare erbjuder SIM flera olika sätt att upprätta kommunikation. Kommunikationen kan ske med lösningar som är fullt integrerade med kunders affärssystem, till exempel via webbportal eller via e-post.

Flödet i SIM är följande: Efter att ha mottagit en order skapar leverantören en elektronisk avisering som bland annat innehåller information om leverans, produkter och kvantitet. Detta lagras därefter i SIMs databas. Informationen användas för att skapa en orderbekräftelse och en godsmarkering som bland annat innehåller streckkoder. Godsmarkeringen placeras på godset. Detta gör det möjligt för mottagaren att få en överblick över inkommande leveranser med hjälp av sin kommunikation till SIM. På så vis kan godsmottagning förberedas på ett effektivt sätt. Vid mottagning kan godsmarkering skannas med en skanningsenhet och information fås om leveransen för att urskilja möjliga avvikelser. Detta rapporteras tillbaks via SIM för att skapa en återkoppling till leverantör.

Alla mottagare har inte tillgång till skanningsenheter och blir då tvungna att manuellt, via SIM, fylla i information om mottaget gods. Detta kan leda till att mottagare väljer bort återkoppling till leverantören, vilket bland annat innebär att:

- Leverantörens fakturaunderlag kommer att baseras på information från skeppningstillfället istället för mottagningstillfället.

- Leverantören får ingen återkoppling om när godset har anlänt och eventuellt ägande (beroende på leveransvillkor) övergår till mottagare.

1.2 Syfte

Syftet med arbetet är att förenkla godsmottagning och återkoppling till leverantör via SIM för mottagare som i dagsläget saknar skanningenheter.

1.3 Mål

Målet är att utveckla en plattformsoberoende applikation för mobila enheter. Denna skall kunna ersätta skanningsenheter. Applikationen skall kommunicera med SIM för att hämta information om leveranser och göra återkoppling. Det ska i applikationen dessutom finnas funktionalitet för att skanna godsmarkering med hjälp av enhetens kamera.

1.4 Avgränsningar

Utvecklingen kommer endast innefatta en klientapplikation, server sidan tillhandahålls av Apper Systems.

2 Metod

Det finns många olika alternativ för utveckling av plattformsoberoende applikationer. För att avgöra vilket som ska användas delas projektet upp i två delar. Den första delen består av en kortare utvärdering av olika utvecklingsalternativ. Detta kommer att ligga till grund för hur applikationen utvecklas. Den andra delen omfattar utvecklingen av en applikation med ett av alternativen.

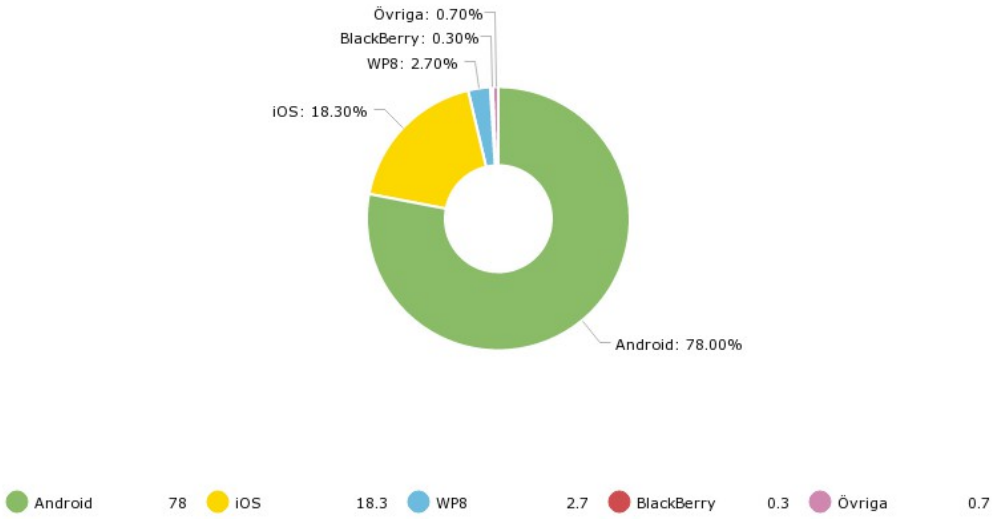
Utvärderingen kommer enbart ske genom information från artiklar, rapporter och tidigare arbeten. Inga testapplikationer kommer att utvecklas. Utvärderingen kommer begränsas genom att ett fåtal av de populäraste alternativen utvärderas. Detta är för att utvärderingen inte ska ta upp för mycket tid då fokus för projektet är utvecklingen av applikationen.

Som utvecklingsprocess kommer scrum att tillämpas. Utifrån användarfall skapas en lista med uppgifter. Uppgifterna representerar funktionalitet som ska implementeras dessa inkluderar design, dokumentation, implementation och testning. Uppgifterna kommer bearbetas i veckosprintar. Innan första sprinten påbörjas kommer en prototyp av applikationens grafiska gränssnitt skapas i form av wireframes. Denna kommer utgöra en startpunkt och en tidig representation av det grafiska gränssnittet.

3 Teknisk bakgrund

3.1 Mobilplattformar

En plattform består av ett operativsystem, hårdvara och diverse utvecklingsverktyg. Med mobilplattform menas en plattform specifikt anpassad för mobilenheter som smarttelefoner. Idag används primärt Android eller iOS. Tillsammans utgör dessa mer än 95% av alla smarttelefoner som finns på marknaden. Andra alternativ är Windows Phone och BlackBerry, som tillsammans utgör ca 3% av den globala marknaden [4].



Figur 3.1: Marknadsandelar för mobila plattformar [4].

3.1.1 Android

Android är ett Linuxbaserat operativsystem med öppen källkod utvecklat av Android Inc. Android Inc. är sedan 2005 en del av Google, som sköter vidareutvecklingen av operativsystemet [5]. Android används i främst i smarttelefoner och läsplattor. Utvecklingsspråket för Android-applikationer är Java och XML. Applikationer finns tillgängliga via Google Play, och 2015 uppgick antalet Android-applikationer till 1,4 miljoner [6]. Android kännetecknas som ett mer öppet system.

3.1.2 iOS

iOS är ett operativsystem utvecklat av Apple Inc för användning på iPod, iPhone och iPad. Det är baserat på Apples OS X, vilket i sin tur är baserat på Darwin som är en Unix variant. iOS-applikationer utvecklas i Objective-C. Applikationer finns tillgängliga genom Apple App Store, och 2015 uppgick antalet till 1,4 miljoner [7].

3.1.3 Windows Phone

Windows Phone är ett operativsystem för Windows-telefoner skapat av Microsoft. Windows Phone har stängd källkod och är baserat på Windows-NT-kärnan. Windows Phone applikationer kan utvecklas med JavaScript, C#, Visual Basic eller C++. Applikationer finns tillgängliga genom Windows Phone Store [8].

3.1.4 BlackBerry OS

BlackBerry OS är utvecklat av BlackBerry Limited och används i stor utsträckning av BlackBerrys smarttelefoner. Applikationer för BlackBerry kan utvecklas i ett flertal språk, exempelvis HTML5/JavaScript/CSS, C/C++/Qt, ActionScript/AIR eller Java [9]. Applikationerna finns tillgängliga på BlackBerry World. Med den senaste version av operativsystemet, OS 10, finns det även stöd för Android-applikationer genom ompaketering av befintliga Android-applikationer [10].

3.2 Mobilapplikation

3.2.1 Plattformsspecifik

Plattformsspecifika, även kallade nativa applikationer är utvecklade för att användas på endast en plattform. Utvecklare har en målplattform och använder dess utvecklingsverktyg. En viktig fördel är att det ger en skräddarsydd upplevelse som fullt utnyttjar plattformens egenskaper och prestanda. Några nackdelar är att det vid utveckling krävs att en applikation skapas för varje plattform. Detta är mycket tids- och resurskrävande. Man blir även tvungen att för varje plattform återskapa samma grafiska utseende och funktionalitet för att få en enhetlig upplevelse [11]. Plattformsspecifika applikationer kan även ge kompetensproblem, eftersom enskilda utvecklare ofta specialiserar sig mot en plattform.

3.2.2 Plattformsoberoende

Plattformsoberoende applikationer är utvecklade för att kunna användas på flera plattformar. Fördelar är att utvecklaren kan nå flera plattformar och kan använda ett och samma utvecklingsverktyg. Utveckling och underhåll är mer effektivt. Nackdelar är att applikationerna kan vara begränsade ur utvecklingssynvinkel, funktioner som implementeras måste stödjas av varje plattform. Utöver det kan även prestandan vara sämre.

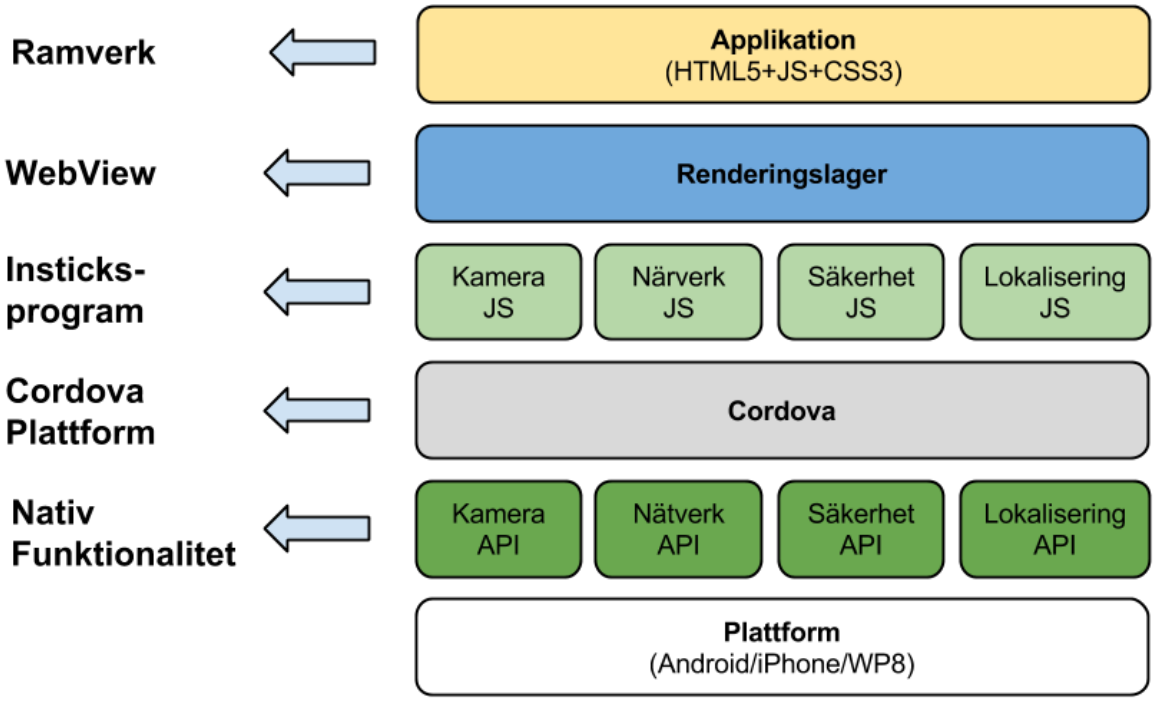
Den här rapporten kommer att gå igenom två olika sätt att utveckla plattformsoberoende applikationer.

- Hybridapplikationer använder webbtekniker och exekveras i en Webview. Fördelar med hybridapplikationer är att man kan uppnå 100% delad kodbas mellan plattformar och att de utvecklas med väletablerade tekniker. Nackdelar är sämre prestanda och att det grafiska gränssnittet är begränsat [11].

- Kompilerade hybridapplikationer översätts innan kompilering till plattformens nativa språk. Detta gör att hela applikationen översätts till maskinkod vid kompilering. Fördelar med kompilerade applikationer är bättre prestanda och att det grafiska gränssnittet rättas efter det nativa gränssnittet. Nackdelar är att det grafiska gränssnittet kan kräva anpassning för varje plattform vilket betyder att den delade kodbasen minskar [12][13].

3.3 Hybridapplikationer

Hybridapplikationer utvecklas med webbtekniker som HTML5, CSS3 och JavaScript. Det är samma tekniker som används för vanliga webbsidor. På grund av detta behövs mobilanpassade ramverk för att ge ett mobilanpassat utseende. Mobilanpassade ramverk fokuserar på prestanda och ett mobilanpassat grafisk användargränssnitt. Många anpassar även gränssnittet utefter målplattform för att ge samma utseende som nativa applikationer. Figur 3.2 visar mjukvarustacken för hybrid-applikationer. Det översta lagret av applikationen består av en webbapplikation som exekveras i enhetens Webview, som uppgör renderingslagret [14][15].



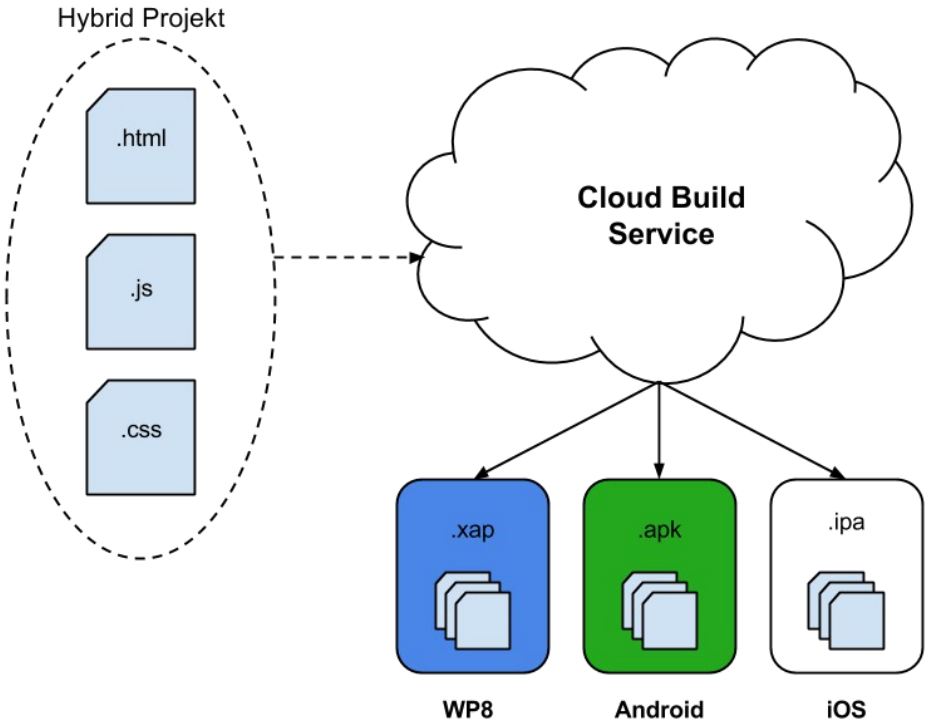
Figur 3.2. Mjukvarustack för hybridapplikationer.

3.3.1 Apache Cordova

Cordova möjliggör åtkomst till nativ funktionalitet för hybridapplikationer. Cordova är baserat på öppen källkod och underhålls av Apache. Cordova är en samling av API:er som möjliggör tillgång till plattformars inbyggda funktioner som till exempel kamera. Detta görs genom insticksprogram, där varje insticksprogram ger tillgång till någon specifik funktionalitet hos enheten. Insticksprogrammen är skrivna med plattformsberoende kod, men för åtkomst genom Cordova API:n används JavaScript. Plattformar som stöds av Cordova är bland annat: iOS, Android, Blackberry och Windows Phone [16]. Cordova har utöver ett rikt bibliotek för standard insticksprogram, även tillgång till tredje-part-insticksprogram med öppen källkod. Dessa ökar användningsområdet för applikationen genom förbättrade verktyg och utökad funktionalitet.

3.3.2 PhoneGap

PhoneGap är en distribution av Cordova som utvecklas av Adobe. När Adobe köpte PhoneGap donerades den dåvarande kärnan som öppen källkod till Apache och döptes till Cordova. PhoneGap innehåller även funktionalitet och tjänster utöver de som finns i Cordova. PhoneGap erbjuder en samling av godkända och verifierade instickspro gram, men det finns även stöd för instickspro gram utvecklade av tredjepart. PhoneGap erbjuder tjänsten PhoneGap Build som är en molnbaserad tjänst. Den kan användas för kompilering av projekt och paketering av applikationer, vilket till exempel möjliggör utveckling av iOS-applikationer från en Windows dator [17].



Figur 3.3: PhoneGap Build paketerar en hybridapplikation som en nativapplikation för varje plattform.

3.3.3 Ionic

Ionic är ett ramverk med öppen källkod som är optimerat för hybridapplikationsutveckling. Ionic bygger på AngularJS vilket ger en bra grund för struktur och funktionalitet. Responsivitet är viktigt i mobilapplikationer därför ligger prestanda i fokus. Detta uppnås bland annat genom minimal manipulation av DOM-trädet samt hårdvaruaccelererad grafik [18]. Ionic innehåller även mobil-anpassade gränssnittselement, såsom knappar och menyer. Dessa anpassas även efter mål-plattformens designdirektiv. Ionic har även en CLI för att bistå i utvecklingen. Den har bland annat funktionalitet för hantering Cordova instickspro gram, kompilering och paketering [19]. Ionic View är en mobilapplikation som kan användas för att snabbt testa applikationer under utveckling [20].Vid utveckling av Ionic applikationer kan valfri IDE med stöd för webbtekniker användas.

3.3.4 Intel XDK

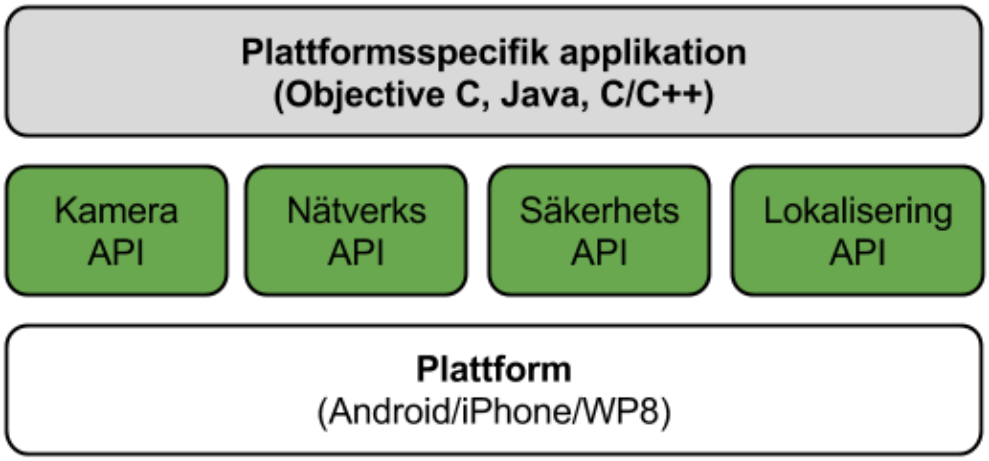
Intel XDK är ett utvecklingsverktyg för hybridapplikationer. Tanken är att verktyget skall klara hela utvecklingskedjan. Intel XDK består av en IDE med inbyggt stöd för utveckling, testning, visning och molnkompilering av applikationer. Det finns även inbyggt stöd för hantering av Cordova, startbilder och ikoner. Det finns en tillhörande mobilapplikation med funktionalitet för visning och testning på en mobilenhet. Detta gör det enkelt att testa hur applikationen ser ut och beter sig i verkligheten. Intel XDK har även verktyg för konstruktion av grafisk gränssnitt med drag-och-släpp funktionalitet. Detta gör det möjligt att snabbt konstruera gränssnitt [21]. Det finns även stöd för en mängd olika ramverk, till exempel Intel App Framework, ett ramverk utvecklat av Intel för användning inom hybrid projekt. Dess UI-komponenter är designade för att anpassas till den nativa enheternas grafiska utseende [22].

3.3.5 Telerik AppBuilder

Telerik AppBuilder är en molnbaserad utvecklingsmiljö för hybridapplikationer. AppBuilder förser användaren med ett ekosystem av tjänster för alla steg i utvecklingsprocessen. Fördelen med en molnbaserad tjänst är att den finns tillgänglig, och fungerar på samma sätt, oavsett användarens hårdvara. Appbuilder fungerar även bra på mindre kraftfulla enheter. Andra fördelar är att det blir lättare att samarbeta då det finns inbyggd versionshantering, så att alla ändringar för alla användare går att se direkt. Eftersom hela miljön är molnbaserad stöds även molnbaserad kompilering. Det finns även användbara verktyg inbyggda för statistik och testning. Telerik har ett komponentramverk för grafiska gränssnitt som heter Kendo UI. Ramverket innehåller en mängd olika komponenter anpassade för mobila enheter [23].

3.4 Kompilerade hybridapplikationer

Kompilerade hybridapplikationer utvecklas med samma språk för flera plattformar. Innan kompilering översätts koden till plattformens specifika språk, och applikationen kan efter kompilering därmed exekveras som en nativapplikation. Figur 3.4 visar mjukvarustacken för en kompilerad hybridapplikation. Stacken består av den kompilerade applikationen som körs direkt på plattformen med direkt åtkomst till plattformsspecifika funktioner.

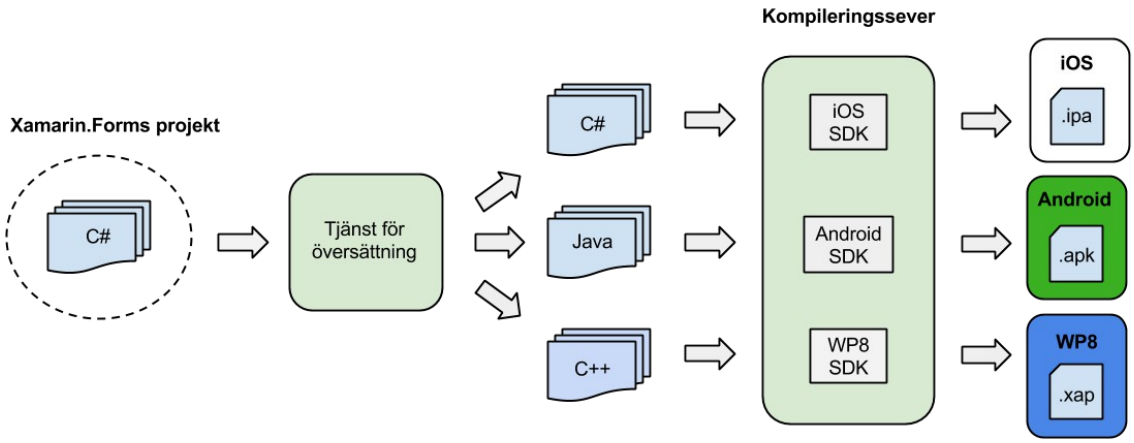


Figur 3.4. Mjukvarustack för kompilerade hybridapplikationer.

3.4.1 Xamarin

Xamarin är ett utvecklingsverktyg med stöd för plattformsoberoende utveckling. Xamarin stödjer utveckling i C# för Windows, iOS och Android. Xamarin gör det möjligt att dela samma logik mellan flera plattformar och till stor del även det grafiska gränssnitt. Utnyttjas detta fullt ut leder det till ett genomsnitt av 75% delad kodbas [24]. Xamarin har även stöd för insticksprogram med färdig funktionalitet för platformsspecifika funktioner [25].

För att uppnå ännu större delad kodbas finns Xamarin Forms. Xamarin Forms är ett grafiskt ramverk med funktionalitet för att utveckla gränssnitt som helt eller till stor del delar kod över ett flertal plattformar (Figur 3.5) [26]. Om detta används möjliggörs nästan fullständig delning av kodbas, inklusive GUI. Dock begränsar Xamarin Forms utvecklingsprocessen något då bara de mest grundläggande funktionerna finns tillgängliga.



Figur 3.5: En illustration av kompileringsprocessen för ett Xamarin.Forms projekt.

3.4.2 Appcelerator

Appcelerator består av utvecklingsmiljön Titanium Studio tillsammans med Titanium SDK [26]. Med dessa verktyg kan 60-90% av kodbasen delas mellan applikationer. JavaScript används som utvecklingsspråk. I utvecklingsmiljön finns det tillgång till emulatorer och avlusare. Det officiella testningsverktyget är Appcelerator Test [27], men det finns även stöd för tredjeparts-tillägg. Vid kompilering kopplas JavaScript-koden ihop med ett API skrivet i den specifika enhetens programmeringsspråk. När applikationen körs kommer exekvering av JavaScript kod hanteras av enhetens JavaScriptmotor. API:t kommer däremot hanteras som en nativ applikation. JavaScript koden kan därmed anropa API:t för åtkomst till nativ funktionalitet [28]. Appcelerator har till skillnad från hybridapplikationer tillgång till plattformens specifika UI-komponenter och funktioner [29].

3.5 Web Storage

Web Storage API:et lanserades tillsammans med HTML5. API:et möjliggör förvaring av nyckel/värde par lokalt på datorn. Web Storage har två format för dataförvaring, Local Storage och Session Storage. Local Storage sparar all data persistent, vilket innebär att all data kommer att finnas kvar även vid nedstängning av webbläsaren. All sparad data är tillgänglig för alla webbsidor inom samma domän. Session Storage sparar data under HTTP-sessionen och finns endast tillgängligt för den aktuella webbsidan [31].

3.6 Testning

3.6.1 Testmetoder

Testning för mjukvara delas ofta in i olika typer utifrån vilka delar av applikationen som testas. De mest standardiserade typerna inom testning är enhets-, integrations-, system-, och acceptanstester. Det finns flera olika utgångspunkter för testning, till exempel Black-box-testning och White-box-testning. Vid Black-box-testning saknas kännedom om den interna strukturen för programmet. Testerna byggs då upp efter förväntade resultat. Motsatsen till Black-box-testning är White-box-testning, där testerna är skapade med vetskap om implementation [32].

- Enhetstestning är testning av individuella enheter eller moduler [33]. Normalt testas funktioner och metoder från enhetens publika API. Testerna verifierar att funktioner returnerar förväntat resultat och felaktig indata hanteras på rätt sätt. Med hjälp av enhetstestning kan brister i algoritmer eller logik upptäckas på ett tidigt stadium.

- Integrationstestning testar interaktionen mellan olika moduler [34]. Exempelvis testas hur olika klasser fungerar tillsammans. Detta verifierar att de samspel som sker mellan de olika delarna av mjukvaran utförs på ett korrekt sätt.

- Systemtestning testar systemet som helhet, det vill säga om det sammansatta systemet fungerar som förväntat [35]. Systemtester sker utan att gå ner på detaljnivå inom mjukvaran.

- Acceptanstestning testar om produkten uppnått de krav, specifikationer och kontrakt som ställts [36]. Inom acceptanstestning särskiljs ofta acceptanstestning ur kundperspektiv från acceptanstestning ur utvecklingssynvinkel, då dessa kan avvika från varandra.

3.6.2 Testverktyg

3.6.2.1 Jasmine

Jasmine är ett testverktyg för JavaScript [37]. Med hjälp av Jasmines API kan implementerade funktioner testas på ett strukturerat och logisk sätt. Jasmine är oberoende av andra JavaScript-ramverk och kan därför användas inom de flesta JavaScript-projekt.

3.6.2.2 Karma

Karma är ett testverktyg för webbsidor. Karma använder en lokal webbserver för att exekvera testfiler för ett valt HTML/JavaScript projekt via vald webbläsare [38]. Projektet testas och servern fångar upp resultaten som sedan presenteras i CLI [33].

4 Kravspecifikation

4.1 Tekniska Krav

De tekniska kraven på applikationen sattes av Apper Systems. Applikationen ska vara en plattformsoberoende applikation med stöd för tre plattformar: Android, iOS och Windows Phone 8. Minst 80% gemensam kodbas, mätt i rader kod, skall uppnås.

4.2 Användarfall

För att beskriva funktionaliteten i systemet skapades ett att antal användarfall. Följande fall identifierades;

1. Användare ska kunna autentisera sig. Autentisering sker genom inloggning med användarnamn och lösenord.

2. Efter första inloggning skall användarnamn och lösenord sparas för att på så sätt automatisera framtida inloggning.

3. Användare ska få en överblick över inkommande leveranser och innehåll för att kunna förbereda godsmottagningar.

4. Användare ska kunna skanna en godsmarkering för att få relevant information. Skannas ett pall-ID skall information om pallen presenteras.

5. Användare ska kunna göra justeringar av antal angivna artiklar en pall innehåller.

6. Användare ska kunna återkoppla till SIM med en mottagningsbekräftelse.

7. Data ska lagras lokalt så att applikationen kan användas även om uppkoppling saknas.

4.3 Användarflöden

Applikationen kommer användas för två syften: förberedande för inkommande leveranser och för att hantera godsmottagningen. Båda dessa kräver autentisering.

Användarflödet för förberedandet består i granskning av inkommande leveranser och eventuellt vidare granskning av leveransens innehåll. För att användare i detta skede ska kunna hitta en specifik leverans eller visst innehåll ska sök- och filterfunktion finnas.

Användarflödet för godsmottagning består av att skanna godsmarkeringar. När en godsmarkering skannas tas användare till korrekt vy i användargränssnittet. Om det var ett pall-ID som skannades visas pallens innehåll. Skannades däremot ett följesedel-ID visas alla pallar som tillhör den aktuella följesedeln. Därefter justerar användaren eventuella diskrepanser och markerar pallen som granskad. Detta utförs för varje pall i en leverans. När alla pallar i en leverans har blivit granskade markeras även hela leveransen som granskad. Därefter kan synkronisering utföras, vilket innebär att lokal data på enheten skickas till servern och en återkoppling till SIM upprättas.

5 Utvecklingsalternativ

Under den inledande fasen av projektet granskades och jämfördes ett antal utvecklingsalternativ för plattformsoberoende applikationer. För att begränsa antalet möjliga alternativ har vissa ramverk och verktyg kombinerats, särskilt i de fall då verktyg och ramverk har samma utvecklare.

Utvärderingen baseras på kostnad, möjlighet att uppfylla kravspecifikation, utvecklingsspråk, kompilering och verktyg för testning.

5.1 Utvärdering

5.1.1 Hybridlösningar

Hybridlösningar använder Cordova för att åtkomst till plattformens funktionalitet.
Webbtekniker används för utvecklingen och kravet på 80% delad kodbas kan uppnås då hela även gränssnitt kan delas mellan plattformar. Det är möjligt att uppfylla kravspecifikationen fullt ut.

5.1.1.1 Cordova+Ionic

Både Cordova och Ionic är baserade på öppen källkod och är kostnadsfria. För kompilering krävs tillgång till varje plattforms SDK och det går därmed inte att utveckla för iOS från en Windows PC. Gränssnittet byggs med Ionic-komponenter. Inbyggda verktyg för att underlätta byggandet av gränssnittet saknas. Ionic erbjuder Ionic View, en applikation som gör det möjligt att på ett enkelt sätt provköra applikationen på en mobilenhet. Valfri utvecklingsmiljö med stöd för webbutveckling kan användas.

5.1.1.2 PhoneGap+Ionic

PhoneGap och Ionic är baserade på öppen källkod och är kostnadsfria. Ytterligare PhoneGap-tjänster såsom Build ligger på olika prisnivåer. Molnkompilering av en applikation är gratis, för högre antal startar priset på 12 dollar per månad. Build-tjänsten erbjuder molnbaserad kompilering för iOS, Android och Windows Phone 8. Gränssnittet byggs med Ionic-komponenter. Inbyggda verktyg för att underlätta byggandet av gränssnittet saknas. Ionic erbjuder Ionic View, en applikation som gör det möjligt att på ett enkelt sätt provköra applikationen på en mobilenhet. Valfri utvecklingsmiljö med stöd för webbutveckling kan användas.

5.1.1.3 Intel XDK+Intel App Framework

Intel XDK och dess tjänster är kostnadsfria. Detta inkluderar obegränsat användande av moln-kompilering. Gränssnittet byggs med Intel App Framework komponenter och det finns verktyg för att underlätta byggandet. Intel XDK har en inbyggd simulator och en mobilapplikation för enkel och snabb testning av applikationen.

5.1.1.4 Telerik Appbuilder+Kendo UI

Telerik Appbuilder har olika prisnivåer beroende på mängden aktiva projekt med lägsta nivån på 29 dollar per månad. Gränssnittet byggs med Kendo UI komponenter och det finns verktyg för att underlätta byggandet. Molnbaserad tjänst vilket medför många fördelar som inbyggda test- och statistik verktyg. Det finns även en mobilapplikation för testning på en mobilenhet.

5.1.2 Kompilerad hybridlösning

5.1.2.1 Appcelerator

Appcelerator har olika prisnivåer där den lägsta börjar på 39 dollar i månaden. Det finns möjlighet att uppfylla kravspecifikationen men kräver att separat gränssnitt utvecklas för varje plattform detta kan leda till svårighet att nå 80% delad kodbas. JavaScript och XML används för utveckling. För kompilering krävs tillgång till varje plattforms SDK. Det går därmed inte att kompilera för iOS från Windows. Det finns avancerade verktyg för testning och analysering. Appcelerator tillhandahåller utvecklingsmiljön Titanium Studio tillsammans med Titanium SDK. I denna miljö finns det tillgång till emulatorer och avlusare.

5.1.2.2 Xamarin

Xamarin har olika prisnivåer där den lägsta börjar på 25 dollar i månaden. För att uppfylla kravspecifikationen på 80% delad kodbas behöver Xamarin Forms användas. C# används som programmeringsspråk. För kompilering krävs tillgång till varje plattforms SDK, det går där med inte att kompilera för iOS från Windows. Antingen kan ett specifikt gränssnitt byggas för varje plattform, eller också kan Xamarin Forms användas. Det finns verktyg för testning, bland annat en molnbaserad tjänst.

Produkt	Kostnad	Möter Kravspec	Kompilering	Språk	Mobil Testapp
Cordova	0	Ja	Kräver SDK	Webbtekniker	Ja
PhoneGap	0	Ja	Molnbaserad	Webbtekniker	Ja
Intel XDK	0	Ja	Molnbaserad	Webbtekniker	Ja
Appbuilder	$29/m	Ja	Molnbaserad	Webbtekniker	Ja
Appcelerator	$39/m	Ja*	Kräver SDK	JavaScript/XML	Nej
Xamarin	$25/m	Ja*	Kräver SDK	C#	Nej

Tabell 5.1: Tabell över olika plattformsoberoende alternativ.
**Möter kravspec om vissa val görs*

5.2 Val

En hybridlösning valdes som utvecklingsalternativ. Gruppens tidigare erfarenheter av webb-applikationsutveckling hade stor inverkan då det förkortar inlärningsprocessen. Kravet på 80% delad kodbas påverkade även då det betyder att även de grafiska gränssnittet måste delas mellan plattformar. Med en hybridapplikation kan nära 100% delad kodbas uppnås. Tillgänglighet av molnbaserad kompilering påverkade även valet då gruppen inte hade tillgång till någon Mac. Potentiella nackdelarna hos hybridapplikationer påverkade inte valet. Applikationen utför inga prestandakritiska uppgifter så eventuella prestandaförluster är därför inget problem. Applikationen kräver heller inte ett avancerad grafiskt gränssnitt då funktionalitet prioriteras.

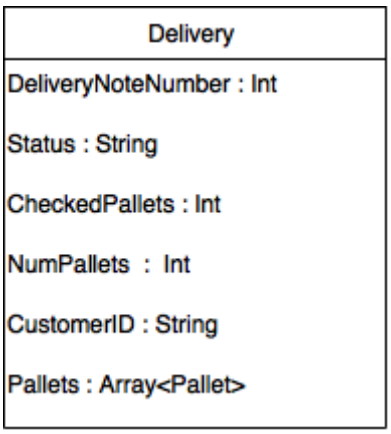
Som hybridlösning valdes Ionic och Cordova. Ionic valdes eftersom den kan uppfylla krav-specifikationen till fullo. Utöver det är utvecklingen mycket aktiv och det nätgemenskap med mycket information om potentiella problem vilket underlättar under utvecklingen. Cordova valdes för att det rekommenderas av Ionic, då det ger mer frihet jämfört med PhoneGap. Med Ionic kan valfri IDE användas; gruppen valde att använda Intel XDK på grund av dess inbyggda simulator, hantering av insticksprogram och molnkompilering. Eftersom Ionic inte erbjuder någon molnkompileringstjänst var detta en stor anledning till att Intel XDK valdes som IDE. Med dessa val uppnås nära 100% delad kodbas över de tre målplattformarna. Att alla verktyg är kostnadsfria bidrog även till valen.

6 Analys

6.1 Domänmodell

I SIM baseras modellen främst på artiklar. Denna modell passar arbetsmetodiken för leverantörer som utför skeppningar och för mottagare som gör beställningar.

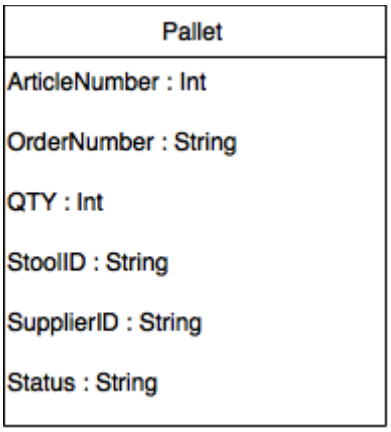
Modellen som används inom projektet är en begränsad version där en pall bara kan innehålla en typ av artikel. Vid godsmottagning är det viktigaste pallar. Om alla pallar i en leverans är mottagna och godkända så är hela leveransen mottagen. Då har alla artiklar anlänt och leverans gått enligt planering. För att anpassa data som mottages från servern för denna arbetsmetodiken omstruktureras den och ytterligare information läggs till. Detta för att underlätta hantering i applikationen.



Figur 6.1: Struktur av delivery-objekt.

Specifikation av fälten i figur 6.1:

DeliveryNoteNumber	Används för att identifiera en leverans, är unikt för varje leverans.
Status	Används för att avgöra status för en leverans. Kan vara granskad eller ogranskad.
CheckedPallets	Används för att visa antal granskade pallar i en leverans.
NumPallets	Används för att visa totalt antal pallar som ingår i en leverans.
CustomerId	Används för att identifiera mottagare mot leverantörer. En mottagare kan vara kopplad mot flera leverantörer och har då ett CustomerId för varje.
Pallets	Innehåller en lista av alla Pallet-objekt som tillhör leveransen



Figur 6.2: Struktur av pallet-objekt.

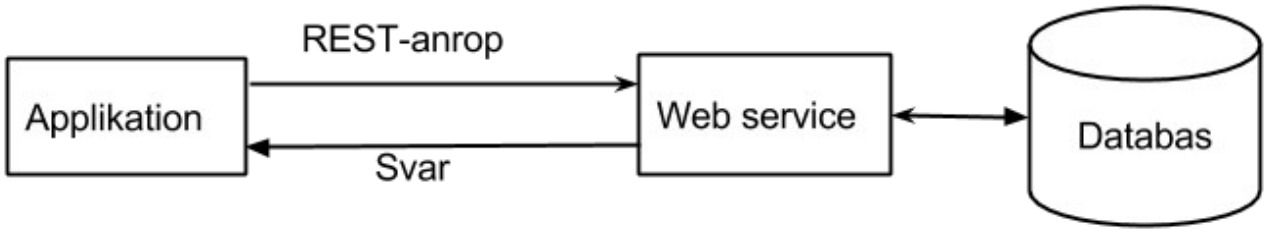
Specifikation av fälten i figur 6.2:

ArticleNumber	Används för att identifiera en artikel, är unikt för varje artikel.
OrderNumber	Används för att identifiera en order. Alla artiklar som är del av en order har samma ordernumber.
QTY	Kvantitet av den artikel som ingår i pallen
StoolID	Används för att identifiera en pall. Det kan bara finnas en typ av artikel på varje pall.
SupplierID	Används för att identifiera leverantören.
Status	Används för att avgöra status för en pall. Kan vara ogranskad, förlorad, granskad med fel eller granskad.

7 Design

7.1 System

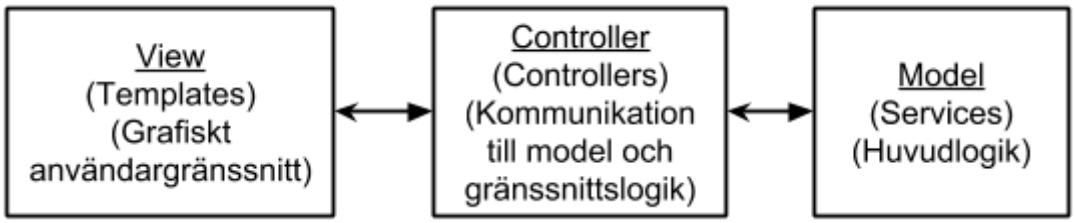
Systemet i sin helhet visas i Figur 7.1. Systemet består av en applikation som kommunicerar med en RESTful webbtjänst som i sin tur är kopplad mot en databas. Webbtjänstens API har stöd för autentisering, hämtning av data och återkoppling. Anrop görs via HTTP-anrop och retur data skickas i form av JSON. Projektets syfte är att utveckla applikationen av detta systemet, resterande delar levereras av Apper Systems.



Figur 7.1: Övergripande bild av systemet.

7.2 Applikation Struktur

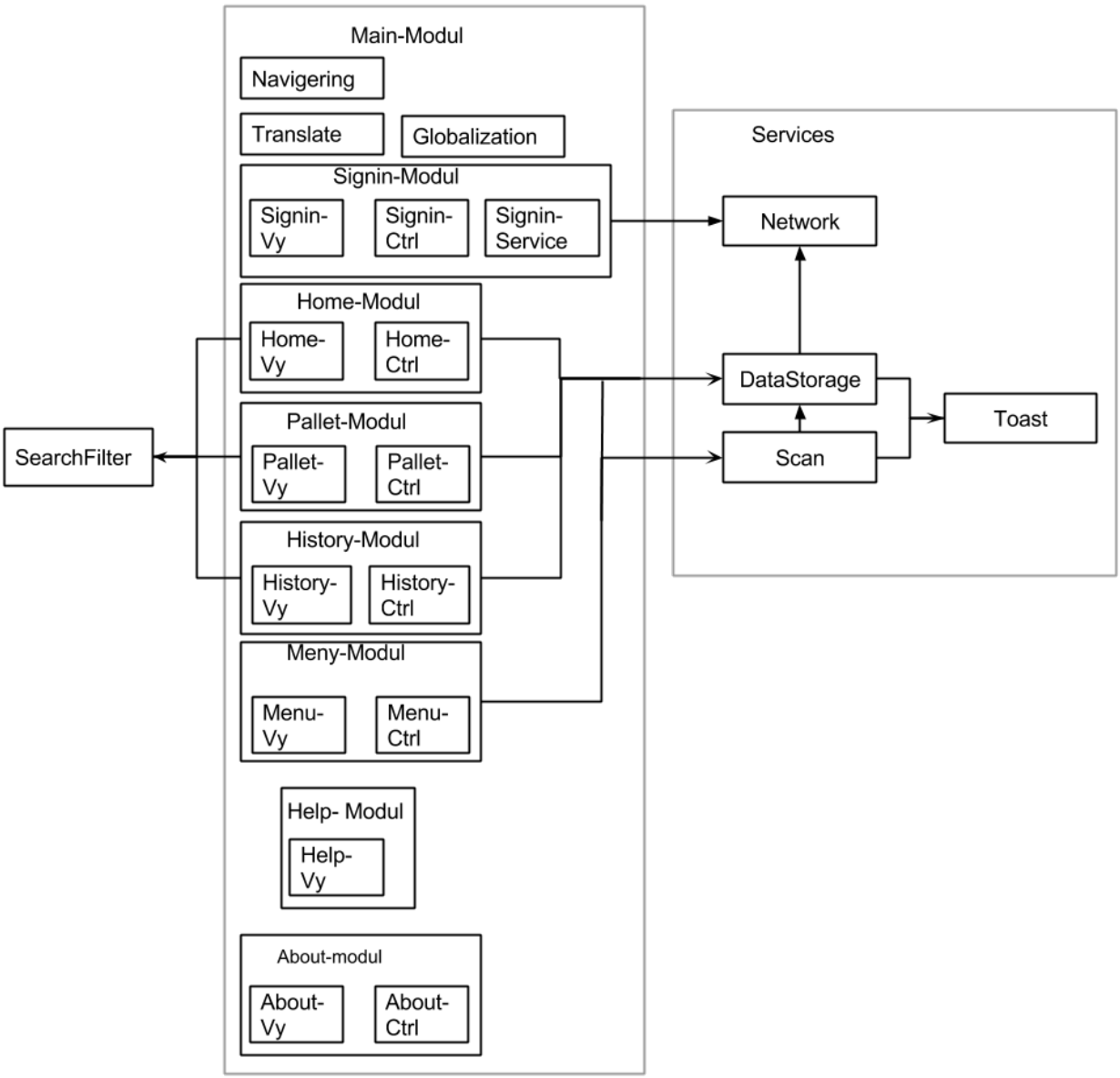
Applikationen använder en MVC-struktur. Med AngularJS delas applikationen upp i templates (View), controllers (Controller) och services (Model) (Figur 7.2). Templates är HTML-filer som innehåller GUI komponenter och instruktioner för hur sidan ska visas. Dessa är kopplade till controllers som hanterar dynamiskt innehåll som visas i templates, lättare logik samt kommunikation med services. Services innehåller mer robust logik, såsom datahantering och kommunikation till webbserver.



Figur 7.2: AngularJS MVC-model.

Applikationen är uppdelad i moduler, vilket ger en så löst kopplad applikation som möjligt. Detta gör det enklare att utföra ändringar eller flytta moduler till nya projekt. Modulerna togs fram utifrån allt som behövdes för att implementera en vy i applikationen. Vyerna baserades på wireframe prototyper av gränssnittet.

Figur 7.3 visar hur applikationen är strukturerad. Det finns en modul för varje vy i applikationen. Varje modul innehar ett grafiskt gränssnitt förutom main-modulen. De vyer som använder någon form av dynamiskt innehåll består även av en controller. Signin Service är en del av signin-modulen för att den endast används där. Resterande services används av flera moduler och är därför inte del av någon specifik modul.



Figur 7.3: Illustration över applikationens struktur.

7.3 Moduler

7.3.1 Main

Main-modulen kopplar ihop applikationens olika delar och konfigurerar applikationen vid uppstart. Den ansvarar även för navigation mellan de olika vyerna inom applikationen. Ionic använder Angular-UI-Router, en AngularJS modul, vilket är ett navigationssystem baserat på vyer [39]. Det finns stöd för nästlade vyer, vilket används i menyn. Det går även att implementera mer avancerad funktionalitet vid navigation till en vy, som inläsning av data eller anrop till tjänster. Detta leder till en bättre kodstruktur då denna typ av funktionalitet samlas.

Globalization används för att avgöra vilket språk mobilenheten använder. Denna information används sedan av Translate för att ställa in språket i applikationen. Innehåller även funktionalitet för inhämtning av tidszon.

Translate är en AngularJS modul. Användningsområde inom applikationen är stöd för språkbyte. Laddning av språk sker asynkront för att minska upplevd prestandapåverkan. Det finns stöd för svenska och engelska.

7.3.2 Menu

Menu-modul ansvarar för att menyn uppträder på ett enhetligt sätt genom hela applikationen. Inläsning av data till controllern utförs samtidigt som denna vy laddas. Detta gör att inga undervyer kommer laddas innan data finns. I menyn finns funktionalitet för att initiera skanning; för detta används Scan service. Det finns även funktionalitet för att initiera synkronisering som använder DataStorage service.

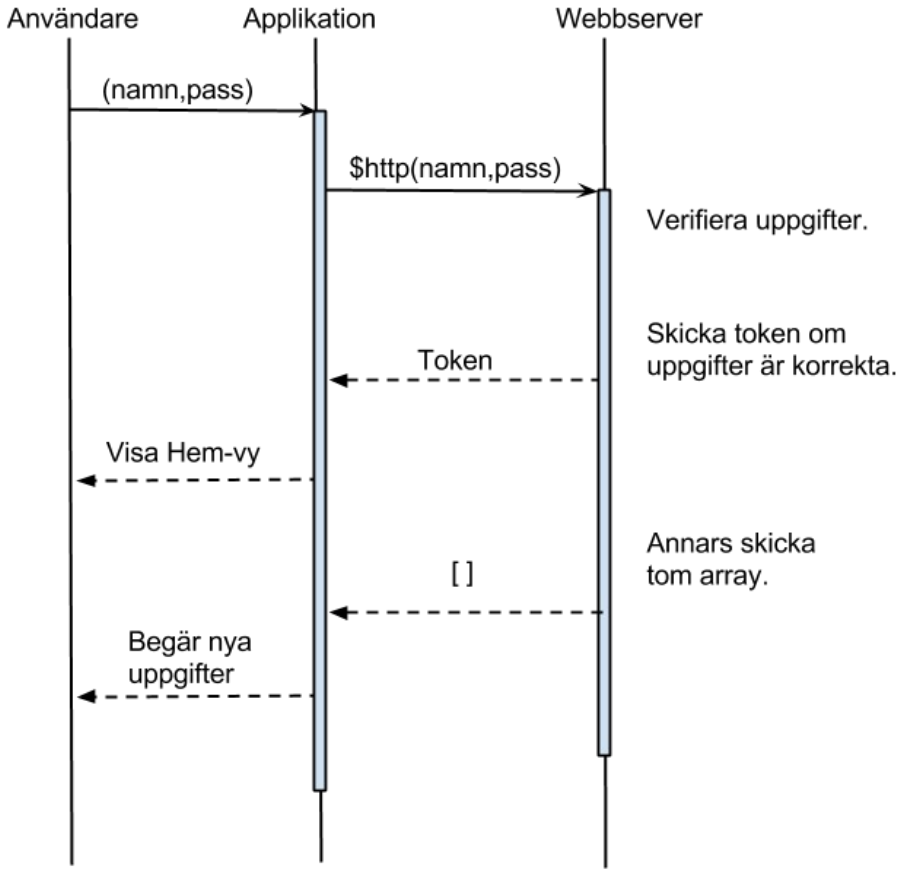
Menu-vyn är en abstrakt vy vilket betyder att den kan inhysa undervyer. Den består av en sidomeny, ett sidhuvud och en ram för att visa undervyer. Huvuddelen av vyn består av området där undervyer visas. Från sidomenyn nås navigation till första nivå av undervyer, som visas inom ramen. Sidhuvudet innehåller titel och knappar för streckkodskanning och synkronisering av data. Sidhuvudet visas på alla undervyer.

7.3.3 Signin

Signin-modulen ansvarar för inloggnings- och utloggningsprocessen samt lagring av användardata.

Signin-vyn visas för användare första gången applikationen startas. Den är separat från resterande vyer i applikationen och delar inte sidhuvud eller sidomeny. Från denna vy kan användare logga in med sitt användarnamn och lösenord. Om informationen är korrekt navigeras användaren vidare till hemskärmen. Om informationen är felaktig visas lämpligt felmeddelande och användaren nekas inträde.

Signin är den enda modulen som använder en dedikerad service. Signin service ansvarar för att skicka vidare användardata vid inloggning och lagra användaruppgifter samt autentiseringsvariabel under sessionen. Vid inloggning skickas användarnamn och lösenord till en RESTful server. Om uppgifterna är korrekta svarar servern med kundnummer och en autentiseringsvariabel. Autentiseringvariabeln kommer sedan att användas vid varje kontakt med servern under sessionen för att validera användaren. Efter lyckad inloggning sparas användarinformation lokalt för automatiskt inloggning vid senare tillfällen. Signin använder Network service vid inloggning. När användaren väljer att logga ut rensas all data, både session och persistent. Sekvensdiagramet i Figur 7.4 visar inloggningsprocessen.



Figur 7.4: Sekvensdiagram för inloggning.

7.3.4 Home

Home-modulen ansvarar för visning av leveranser, för initiering av synkronisering samt för navigering. Den består av en HTML-sida och en controller. Inläsning av data till controllern görs genom navigeringssystemet. Det finns funktionalitet för att initiera synkronisering, men ansvaret för synkroniseringen ligger hos DataStorage.

Home-vyn visar en lista av kort som innehåller information om leveranser. Kort är ett vanligt sätt att presentera information inom mobila gränssnitt. Användare ska snabbt kunna få en överblick av dagens inkommande leveranser och deras status. Filtrering och sök alternativ finns tillgängligt i en subheader för att underlätta planering av inkommande leveranser. Trycker användaren på ett kort visas Pallet-vyn med mer detaljerad information om leveransens pallar.

7.3.5 Pallet

Pallet-modulen ansvarar för visningen och granskningen av pallar. Det är den enda vyn i applikationen som är en andra nivå undervy. Detta innebär att sidomenyn är oåtkomlig och att navigationen är begränsad till en bakåt knapp.

Pallet-vyn listar innehållet i en leverans uppdelat efter pall. Användare ska få en överblick av pallar som ingår i en leverans och hur många som har blivit godkända. När alla pallar inom en leverans granskats markeras hela leveransen som granskad. Pallstatus visas med färgmarkering för snabb avläsning. Filter och sök funktionalitet finns även i subheader.

7.3.6 History

History-modulen ansvarar för visning av synkroniserade pallar. Den innehåller en HTML-sida och en controller. Den använder DataStorage för att hämta synkroniserade pallar.

History-vyn visar leveranser som blivit godkända och synkroniserade till servern av användaren. Dessa leveranser kan ej längre justeras. De är sorterade efter datum. Filter och sökfunktionalitet finns även i subheader.

7.3.7 Help

Help-modulen ansvarar för visning av en hjälpguide. Den består endast av en HTML sida, det vill säga endast grafiska komponenter. Ingen logik utförs här.

Vyn innehåller information om hur man hanterar applikationens funktioner. Hit ska en användare vända sig om något är oklart och kunna få den information som behövs för att lösa problem.

7.3.8 About

About-modulen ansvarar för visning av licenser och språkbyte. About består av en HTML sida och en controller.

About-vyn innehåller information om de licenser applikationen brukar och har funktionalitet för att byta språk. I nuläget finns det stöd för svenska eller engelska.

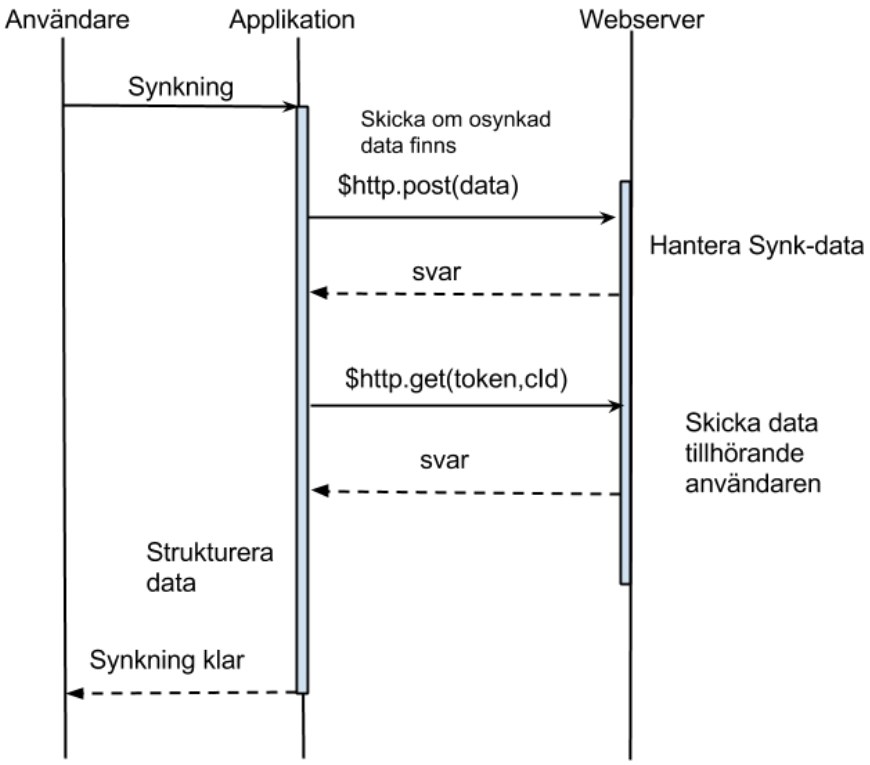
7.3.9 SearchFilter

Filter är en del av AngularJS, de fungerar genom att ta in en mängd element och ge tillbaka en delmängd beroende på villkor. Filter används i applikationen för att filtrera vid sökningen, villkoret är då söksträngen. Användare ska kunna söka efter leveranser och pallar beroende på deras innehåll. Vid vanlig sökning används de filter som är inbyggda i AngularJS. Men problem uppstår när olika fält kan innehålla samma eller delar av samma teckenkombinationer. För att lösa detta används ett filter som ger funktionalitet att söka efter ett specifikt fält.

7.4 Services

7.4.1 DataStorage

DataStorage ansvarar för synkronisering, sortering och lagring av data. Synkronisering används för att samordna data som finns lokalt på enheten med servern. Synkroniseringen utförs i två steg (Figur 7.5). I det första steget skickas osynkroniserad data till servern om sådan data existerar. I det andra hämtas data. DataStorage använder sig av Network service för att kommunicera med servern. Data som hämtas sorteras efter leverans och behandlas genom tillägg av information som underlättar bearbetning i applikationen. För lagring av data används Web Storage i form av Local Storage. Local Storage används för att lagra leverans-, synkroniserings- och historikdata mellan sessioner. Användarnamn och lösenord sparas även för automatisk inloggning. Local Storage valdes för att det stöds av alla plattformarna.



Figur 7.5: Sekvensdiagram av synkning.

7.4.2 Network

Network sköter kommunikation till servern genom AngularJS $http tjänst. $http tjänsten förenklar REST-anrop och möjliggör asynkron kommunikation. Den utför endast enkla HTTP-anrop till en RESTful webbtjänst och lämnar behandling av svaren till andra tjänster. Funktionalitet för inloggning, hämtning av fraktsedlar och återkoppling finns.

7.4.3 Scan

Scan sköter hanteringen av streckkodsskannning. Ett insticksprogram till Cordova används för att skanning ska fungera enhetligt över alla plattformar. Insticksprogrammet har stöd för de mest standardiserade streckkoderna och specifikt Code_39 som används inom SIM. Det finns stöd för att skanna leverans-ID och pall-ID, detta för att de är unika fält som används för identifiering. Resultatet av skanningen jämförs med lokal data. Om en matchande leverans eller pall hittas navigeras användare till rätt vy, annars visas felmeddelande.

7.4.4 Toast

Toast används för att kunna visa enhetliga meddelanden i hela applikationen. Meddelanden kommer se ut på samma sätt och visas på samma position oavsett vart dom skickas ifrån. Detta gör att services kan skicka meddelanden direkt till användaren, något som vanliga fall görs genom templates. Ett insticksprogram till Cordova används för att visa enhetliga meddelanden över alla plattformar.

7.5 Testning

7.5.1 Enhetstestning

Enhetstestning består av att applikationen bryts isär till små testbara enheter, såsom moduler och i vissa fall enskilda komponenter ur modulerna. Detta genomfördes med hjälp av Jasmine och Karma, två testverktyg.

7.5.2 Integrationstestning

Integrationstestning användes för att kontrollera kopplingen mellan olika delar av applikationen, såsom controllers och services. Syftet var att verifiera funktionaliteten mellan enheter. Detta genomfördes med hjälp av Jasmine och Karma.

7.5.3 Systemtestning

Systemtestning är test som utförs på systemet som helhet. Syftet med testerna är att verifiera funktionalitet hos systemet som en enhet. Detta genomfördes till stor del på mobila enheter med hjälp av Ionic View. Tester utfördes även i emulatorer för samtliga plattformar. För testning på verkliga enheter kompilerades applikationen till Android och testades på en Samsung Galaxy S4.

8 Implementation

8.1 Utvecklingsmiljö och process

För versionshantering inom projektet har Git använts, eftersom samtliga medlemmar i gruppen har god erfarenhet av och kunskap om systemet. Källkoden har distribuerats bland gruppmedlemmarna med Github.

Filstrukturen i applikationen delades upp efter vyer där varje vy behandlas som en modul. Modulerna implementerades i början som tomma skal. Dessa fylldes sedan ut med logik allt eftersom funktionalitet implementerades. Vyernas grafiska gränssnitt består av Ionic-komponenter. Översättning implementerades tidigt i applikationen för att inte skapa onödigt arbete senare. Cordova instickspogram implementerades med hjälp av Intel XDK.

Under projektet användes 3 olika sätt att testa funktionalitet, grafiskt utseende och prestanda. Jasmine och Karma användes för att utföra tester på moduler och mindre enheter inom kodstrukturen. Jasmine-tester har lagts till löpande under projektets gång för att testa nya funktioner och implementeringar. För att få en överblick av det grafiska utseendet har simulatorer för iOS, Android och Windows Phone 8 används. Användningen av simulatorer snabbade upp utvecklingen markant då ändringar snabbt kunde testas på de olika enheterna. En nackdel med att testa i simulator är att prestandan inte är representativ för en riktig enhet. För att bedöma prestandan användes Ionic View. Testning utfördes även på riktiga mobila enheter.

8.2 Inloggning

Inloggning sker med användarnamn och lösenord. Dessa skickas i nuläget okrypterat i form av en GET-anrop. Om användaruppgifterna stämmer överens med SIMs databas returneras ett JSON objekt med kund-ID samt en autentiseringsvariabel (Se nedan). Denna variabel används sedan vid varje kontakt med servern för att identifiera användaren.

[{"CustomerID":["12323","54324","76431"],"Token":"XXX"}]

Om användaruppgifterna skulle vara felaktiga returneras ett tomt JSON objekt.

```
//Auto login
DataStorage.getUserInfo().then(function(success) {
    if(states[networkState] != 'No network connection'){
        //Internet connection, login the user
        Signin.login(success.username, success.password)
    }
    else{
        //No internet but saved user! Go to Home.
        $rootScope.$broadcast('event:auth-loginConfirmed', status);
    }
},
function(error){
    //No saved user, do nothing.
});
```

Figur 8.1: Kodimplementation för automatisk inloggning.

Vid inloggning sparas användarens uppgifter lokalt på enheten. Dessa uppgifter bevaras och tas endast bort när användaren själv väljer att logga ut. Koden i Figur 8.1 exekveras vid uppstart av applikationen för att möjliggöra automatisk inloggning. Två olika metoder används vid inloggning beroende på om enheten har uppkoppling eller ej. Finns uppkoppling kommer inloggningen ske som beskrivet ovan. Om uppkoppling ej finns, kommer användaren ändå har tillgång till lokal data.

8.3 Skanning

Vid skanning utnyttjas instickskprogrammet *barcodeScanner* för att läsa av godsmarkeringen. I nuläget finns det stöd för skanning av följesedel-ID eller pall-ID. Följesedel-ID har bokstaven “N” som prefix, medan pall-ID har bokstaven “S”. Detta används för att avgöra vilken streckkod som skannades. Den information som erhålls används sedan för att lokalisera en viss leverans eller pall inom applikationen. Informationen presenteras för användaren för granskning. Figur 8.2 visar hur Scan service fungerar.

```
cordova.plugins.barcodeScanner.scan(
    function (success) {
        if(!success.cancelled){
            var scanId = success.text;
            switch(scanId.charAt(0)) {
                    case 'N' :
                        scanId = scanId.replace('N','');
                        if(DataStorage.dispatchExist(scanId)){
                            $ionicViewSwitcher.nextDirection("forward");
                            $state.go('menu.pallets', {dispatch : scanId});
                        }
                        else
                            Toast.toast("no dispatch found");
                        break;
                    case 'S' :
                        scanId = scanId.replace('S','');
                        var result = DataStorage.palletExist(scanId);
                        if(result){
                            $ionicViewSwitcher.nextDirection("forward");
                            $state.go('menu.pallets',{dispatch: result, pallet: scanId})
                        }
                        else
                            Toast.toast("no pallet found");
                        break;
                    default :
                        Toast.toast("invalid scan");
                        break;
            }
        }
        else{
            Toast.toast("Scan cancelled");
        }
    },
    function (fail) {
        console.log("Scan failed:"+fail)
    }
);
```

Figur 8.2: Implementation av skanning av godsenheter.

8.4 Synkronisering

Synkronisering används för att skapa konsistent data mellan lokal enhet och server. Vid ändring av data lokalt sparas den förutom i minnet även separat i Local Storage under nyckelvärdet *syncData*. Vid synkronisering kontrolleras om *syncData* existerar. Om *syncData* finns så skickas datan till servern för återkoppling och placeras sedan i historiken. Därpå efterfrågas ny data från servern. Figur 8.3 demonstrerar hur synkroniseringen arbetar. Denna funktion hanterar även de fall där autentiseringsvariabel har blivit ogiltig genom att logga in användaren igen. På så vis återfås en giltig autentiseringsvariabel och synkroniseringen kan återupptas.

```
if(window.localStorage['syncData']){
    //Unsynced data in local storage.
    console.log("unsynced data")
    Network.post().then(function(succes){
        moveToSynced();
        window.localStorage.removeItem('syncData');
        sync().then(function(success){
            deferred.resolve();
        });
    })
}
else{
    //No unsynced data, get new data.
    Network.dbTestData().then(function(success){
            if(success[0].data[0].DeliveryNoteNumber == "Invalid token"){
                //If token is unvalid, login the user and try again.
                getUserInfo().then(function(success){
                    Network.login(success.username, success.password).then(function(data){
                        window.localStorage.setItem("token", data[0].Token)
                        sync().then(function(success){
                            deferred.resolve();
                        });
                    }, function(fail){
                        console.log(fail);
                    })
                })
            }
            else{
                //If token is valid, structure the data
                for(var i =0; i < success.length; i++)
                    for(var j =0; j < success[i].data.length; j++)
                        syncData.push(success[i].data[j]);
                structure(syncData);
                deferred.resolve();
            }
        },function(fail){
            console.log("fail in datastorage");
            deferred.resolve();
        })
    }
```

Figur 8.3: Kodimplementation för synkronisering av data.

8.4.1 Hämtning av data

Efter inloggning hämtas data kopplat till de kund-ID som erhölls vid lyckad inloggning. GET-anropet upprepas för alla kund-ID som finns lagrade (Figur 8.4), och returneras sedan för lagring i Local Storage. Anropen är asynkrona vilket betyder att applikationen fortsätter köras medan anropen utförs.

```
if(window.localStorage['syncData']){
    //Unsynced data in local storage.
    console.log("unsynced data")
    Network.post().then(function(succes){
        moveToSynced();
        window.localStorage.removeItem('syncData');
        sync().then(function(success){
            deferred.resolve();
        });
    })
}
else{
    //No unsynced data, get new data.
    Network.dbTestData().then(function(success){
            if(success[0].data[0].DeliveryNoteNumber == "Invalid token"){
                //If token is unvalid, login the user and try again.
                getUserInfo().then(function(success){
                    Network.login(success.username, success.password).then(function(data){
                        window.localStorage.setItem("token", data[0].Token)
                        sync().then(function(success){
                            deferred.resolve();
                        });
                    }, function(fail){
                        console.log(fail);
                    })
                })
            }
            else{
                //If token is valid, structure the data
                for(var i =0; i < success.length; i++)
                    for(var j =0; j < success[i].data.length; j++)
                        syncData.push(success[i].data[j]);
                structure(syncData);
                deferred.resolve();
            }
        },function(fail){
            console.log("fail in datastorage");
            deferred.resolve();
        })
    }
```

Figur 8.4: Hämtning av data.

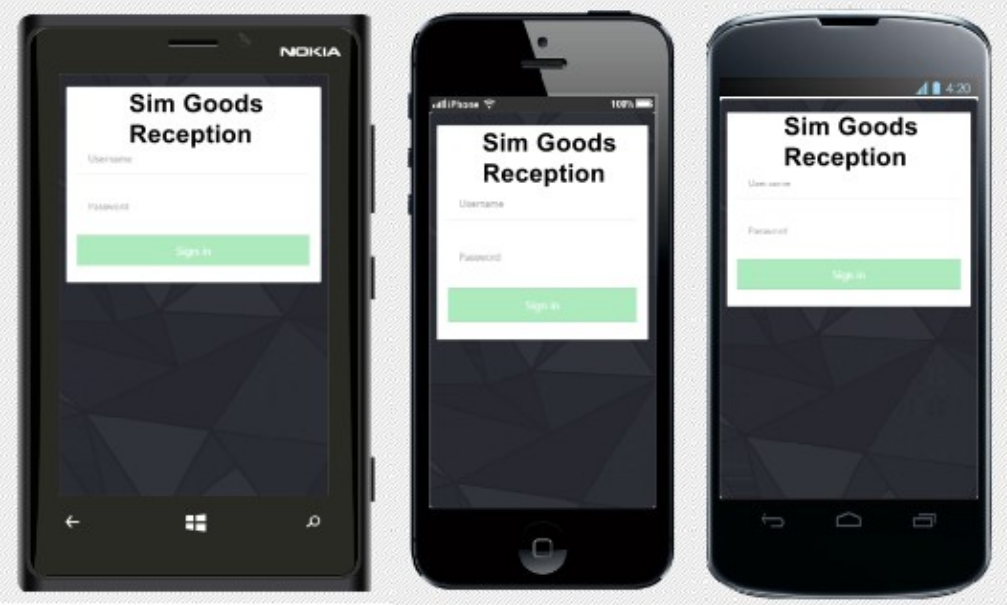
9 Resultat

Resultatet är en applikation som följer ovanstående kravspecifikation.

9.1 Gränssnitt och funktionalitet

Vid utformning av gränssnittet hade användarflödet och presentation av relevant information hög prioritet. Då skanning och godsmottagning är huvudsyftet för applikationen utformades vyerna med detta som främsta syfte. Endast information som är relevant för mottagningsprocessen presenteras.

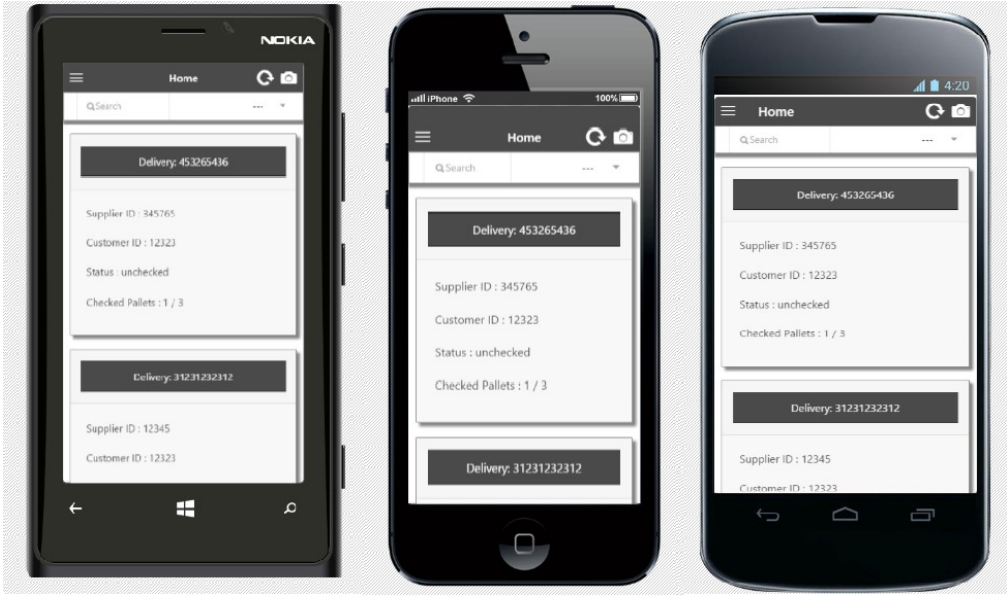
9.1.1 Signin-vy



Figur 8.5: Grafiskt gränssnitt för signin-vyn.

Signin-vyn är startvyn för applikationen. Användare begärs på användaruppgifter och kan därefter logga in. Om uppkoppling saknas eller om felaktig information anges visas felmeddelande. Vid inloggning visas en laddningsruta tills inloggningsprocessen är slutförd.

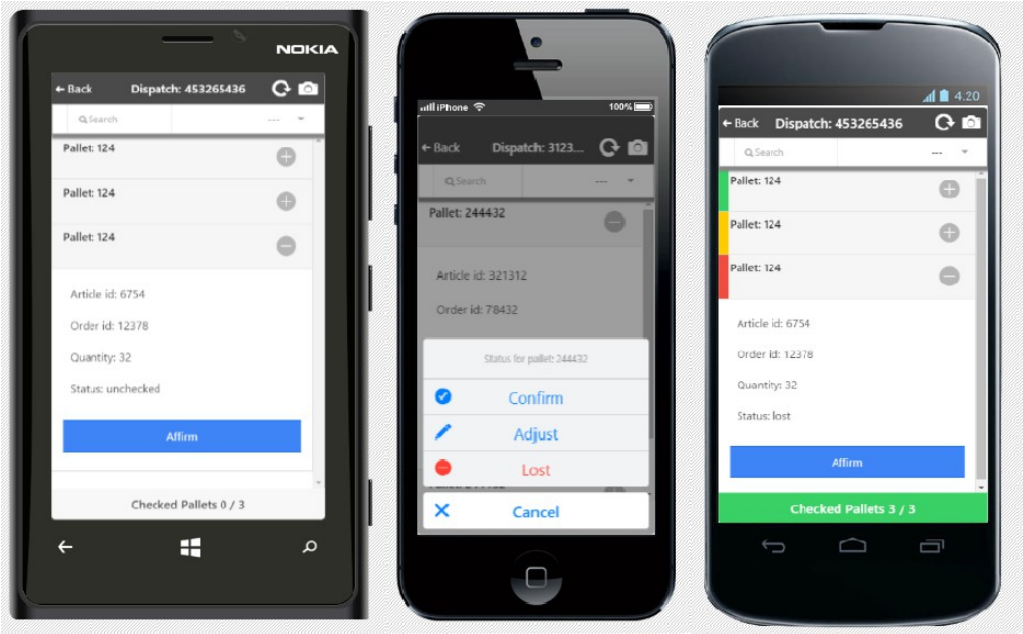
9.1.2 Home-vy



Figur 8.6: Grafiskt gränssnitt för home-vyn.

I home-vyn presenteras dagens följesedlar i form av kort. Endast relevant information om varje följesedel visas. För att skanna gods används kameraknappen i det övre högra hörnet. Synkronisering med servern sker antingen via synk knappen, som är positionerad till vänster om kameraknappen, eller genom att dra ner innehållet inom vyn. För att navigera till ytterligare vyer används en sidomeny, som öppnas med hjälp av menyknappen i det övre vänstra hörnet.

9.1.3 Pallet-vy



Figur 8.7: Grafiskt gränssnitt för pallet-vyn.

I Pallet-vyn kan användaren se antalet pallar som tillhör varje följesedel. Inom varje pall visas relevant information för godsmottagning. Användaren kan sedan godkänna, justera antal artiklar eller markera pallen som borttappad. En granskad pall får en färgmarkering för snabb igenkänning. Färgkoden är grön för godkänd, gul för ändrad och röd för borttappad.

10 Slutsats

Applikationen är en prototyp som ännu inte testad i drift. Dock är all funktionalitet för att den skall kunna ersätta en skanningsenhet implementerad. Vi anser därmed att vi uppnått projektets syfte och mål.

10.1 Resumé

Sex olika utvecklingsalternativ för plattformsoberoende applikationer har utvärderats. Av dessa klassificeras fyra som hybridapplikationer och två som kompilerade hybridapplikationer. Det är stora skillnader mellan de olika typerna av plattformsoberoende utveckling; båda har sina för- och nackdelar. Hybridapplikationer utvecklas med webbtekniker och kompileras vid körtid. Kompilerade hybridapplikationer kodas i ett gemensamt språk men översätts innan kompilering till plattformsspecifikt språk. Valet mellan de två alternativen beror till stor del på tidigare kunskap och kraven på applikationen som ska utvecklas.

För utveckling av applikationen valdes en hybridlösning. En kombination av Cordova, Ionic och AngluarJS användes. Denna lösning passade bäst baserat på kravspecifikation för applikationen och gruppens tidigare erfarenhet. Som IDE valdes Intel XDK då den innehåller den funktionalitet som ansågs krävas inom projektet. Funktioner för testning och felsökning med simulator har i stor omfattning underlättat arbetet.

De valda utvecklingsverktygen fungerade bra. Den resulterade applikationen är en hybrid-applikation utvecklad med webbtekniker. Att webbtekniker används ledde till att det finns stora mängder information som underlättade utveckling. Det fanns insticksprogram till Cordova för all funktionalitet som krävdes. Det fanns mycket gedigen dokumentation om Ionic som underlättade arbetet. Även Intel XDK fungerade bra och simulatorn blev en viktig del av utvecklingsprocessen. Den molnbaserade kompileringstjänsten användes vid testning av applikationen på riktiga mobilenheter och Ionic View användes för att snabb testa hur applikationen såg ut. Applikationen är en prototyp, som fungerar enligt krav på alla målplattformar med full funktionalitet. Applikationen är inte satt i bruk men det arbete och tester som utförts leder till tron att den kommer fungera som avsett.

10.2 Kritisk diskussion

Utvärdering av utvecklingsverktyg gav en insikt i hur många olika alternativ det finns. I rapporten tar vi upp sex olika alternativ där vissa har en del överlappande komponenter, men det finns betydligt fler. Utvärderingen av verktygen inkluderade ingen utveckling av testapplikationer, vilket möjligen skulle kunna lett till en bättre utvärdering. Det hade dock varit en tidskrävande process och låg utanför tidsramen för vårt projekt där större vikt låg på utvecklingen av applikationen.

Utvecklingen av applikationen fungerade bra. Gruppens tidigare erfarenhet av webbappliktions-utveckling gjorde att arbetet kunde komma igång snabbt. Enligt planeringen skulle arbetet utföras under veckosprintar bestående av planering, implementering och testning. Detta höll inte och arbetet delades inte upp i sprintar. Istället arbetade vi mot en backlog med funktionalitet som skulle implementeras. Detta gjorde utvecklingen mindre strukturerad men mer flexibel.

Valet att utveckla en hybridapplikation betydde att webbtekniker användes. Det finns mycket information och dokumentation om utveckling inom detta då metoden använts länge. Detta har för- och nackdelar; två fördelar är att det finns mycket hjälp om problem uppstår och att det finns väldefinierade standarder. Några nackdelar är att mycket av informationen och funktionaliteten hos webbtekniker inte avsedd för mobila plattformar och att funktionalitet man försöker implementera kanske inte helt stöds av plattformen.

Plattformsoberoende applikationer är en marknad som troligen kommer fortsätta växa, då propositionen att endast behöva utveckla och underhålla en applikation är tilltalande för företag. Den mesta funktionalitet kan idag uppnås med en plattformsoberoende applikation; skillnaden mot plattformsspecifika ligger snarare i det grafiska gränssnittet och i prestandan. Med utvecklings-verktygen under konstant utveckling och ökande prestanda hos mobila enheter minskar skillnaderna. Jämfört med nativ utveckling där varje plattform erbjuder ett enda utvecklingssätt finns här ett tiotal. Det är svårt att veta vilket man ska använda och om det kommer fortsätta vara relevant i framtiden.

10.3 Samhällsaspekter

När utvecklare väljer vilka plattformar man ska nå med en applikation vägs andel användare som nås mot kostnaden av utveckling och underhåll. När marknaden domineras av två plattformar blir dessa ofta de enda som nås. Det är helt enkelt inte värt att utveckla en separat applikation för 3% av marknaden. Detta leder till att mindre plattformar helt saknar vissa applikationer och därmed blir mindre attraktiva för konsumenter. Målet med plattformsoberoende applikationer är att kunna nå till så många plattformar som möjligt med en applikation. Användandet av plattformsoberoende utveckling skulle i förlängningen leda till att mindre plattformar inte väljes bort under utveckling och inte heller av konsumenter på grund av saknade applikationer. Sett från en marknadsekonomiskt synvinkel skulle detta förbättra konkurrens på marknaden.

Plattformsoberoende utveckling är mycket resurseffektivt, då utvecklingstiden kan minskas markant jämfört med nativ applikationsutveckling. Detta leder mer effektiv användning av resurser hos företag. Applikationen blir även tillgänglig tidigare för alla plattformar vilket kan leda till snabbare acceptans hos användare. På sikt kommer detta troligen leda till ett ökande av plattformsoberoende utveckling.

Nativa applikationer använder ofta senaste tekniken som kräver senaste operativsystemet. Detta kan leda till att vissa äldre mobiltelefoner och läsplattor inte får tillgång till applikationen. Hybridapplikationer utvecklas med webbtekniker som inte genomgår stora förändringar lika ofta och ofta är bakåtkompatibla. Detta förlänger äldre enheters livstid och betyder att äldre enheter kan få tillgång till applikationen. Detta kan leda till mindre konsumtion av tekniska produkter, vilket är gynnsamt för miljön.

Applikationen som utvecklas i projektet kan användas som alternativ till skanningsenheter. Detta skulle leda till minskad efterfråga av skanningsenheter då de flesta har tillgång mobiltelefoner. På sikt skulle detta kunna innebära en miljövinst då färre skanningsenheter behöver tillverkas och distribueras. Även hos SIM användare som idag inte använder skanningsenheter skulle applikationen kunna ge en positiv påverkan på miljön; användaren kan via applikationen ha direkt återkoppling till leverantör/transportör, som vid eventuell reklamation kan planera en effektiv upphämtning/avlämning av det reklamerade godset. På så vis kan onödiga extrakörningar och omvägar undvikas, vilket minskar påfrestningen på miljön.

10.4 Framtida utveckling

Push notifications
Push notifications används för att skicka meddelanden till användare av en mobil applikation. Det skulle kunna användas för att informera användare om att en ny leverans har blivit inlagd eller att en leverans är på ingång. Ionic har precis börjat implementera stöd för detta.

Databas
Databas för lagring och synkronisering: För förvaring av större datamängder i strukturerad form skulle en bättre lagringslösning behövs i form av en databas. Det finns lösningar som inte bara ger bättre lagringsmöjligheter men som även kan underlätta synkronisering mot servern och mot andra mobila enheter. Detta skulle kunna användas för att skapa ett system där flera skanningsenheter kan användas. Synkronisering skulle då ske både direkt mellan mobilenheter och via servern. I dagsläget finns det inte heller stöd för att synkronisera enskilda pallar.

Automatisk reklamation
I SIM finns funktionalitet för reklamation om det uppstått problem med ett gods. Mobil-applikationen skulle kunna byggas ut med automatiskt stöd för ifyllande och skickande av reklamation. Detta skulle ytterligare förenkla mottagningsproceduren.

Grafiskt Gränssnitt
För att förbättra användarupplevelsen kan gränssnittet förstärkas med mer användarvänliga funktioner. Animeringar för att göra synkroniseringen mer tydlig, visandet av status på leveranser är två exempel på hur detta kan göras.

Kryptering
I nuläget används ingen kryptering. Kryptering av kommunikation med webbservern kan implementeras med SSL/TSL över HTTPS. Kryptering av lagrad användarinformation kan krypteras med Cordova insticksprogram som använder plattformens inbyggda säkerhet. Vid utförandet av detta projekt fanns inget insticksprogram med stöd för alla tre målplattformar.

11 Referenser

[1] SitePoint, 'What Is CSS? - SitePoint', 2014. Tillgänglig från: http://www.sitepoint.com/web-foundations/css/. [Citerad: 09- Jun- 2015].

[2] Mozilla Developer Network, 'HTML5', 2015. Tillgänglig från: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. [Citerad: 09- Jun- 2015].

[3] Mozilla Developer Network, 'Introduction', 2015. Tillgänglig från: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. [Citerad: 09- Jun- 2015].

[4] www.idc.com, 'IDC: Smartphone OS Market Share', 2015. Tillgänglig från: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Citerad: 09- Jun- 2015].

[5] R. Goodwin, 'What Is Android? Google's Mobile OS Explained', *Know Your Mobile*, 2015. Tillgänglig från: http://www.knowyourmobile.com/google/18749/what-android-googles-mobile-os-explained. [Citerad: 09- Jun- 2015].

[6] Statista, 'Number of available apps in the Google Play Store 2015 | Statistic', 2015. Tillgänglig från: http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. [Citerad: 09- Jun- 2015].

[7] Statista, 'Apple App Store: number of available apps 2015 | Statistic', 2015. Tillgänglig från: http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/. [Citerad: 09- Jun- 2015].

[8] Msdn.microsoft.com, 'Languages, tools and frameworks - Windows app development', 2015. Tillgänglig från: https://msdn.microsoft.com/en-us/library/windows/apps/dn465799.aspx. [Citerad: 09- Jun- 2015].

[9] Developer.blackberry.com, 'Platform Choice - BlackBerry Developer', 2015. Tillgänglig från: https://developer.blackberry.com/devzone/develop/platform_choice/index.html. [Citerad: 09- Jun- 2015].

[10] Developer.blackberry.com, 'Download for Windows - Runtime for Android apps - BlackBerry Developer', 2015. Tillgänglig från: http://developer.blackberry.com/android/tools/. [Citerad: 09- Jun- 2015].

[11] P. Rudolph, 'Hybrid Mobile Apps: Providing A Native Experience With Web Technologies – Smashing Magazine', *Smashing Magazine*, 2014. Tillgänglig från: http://www.smashingmagazine.com/2014/10/21/providing-a-native-experience-with-web-technologies/. [Citerad: 09- Jun- 2015].

[12] R. Khanna, 'Evolution of Hybrid App's WebView in iOS & Android & its Performance',*Airpair.com*, 2015. Tillgänglig från: https://www.airpair.com/android-webview/posts/evolution-of-webview-ios-and-android. [Citerad: 09- Jun- 2015].

[13] M. Kort and E. Oksman, 'Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options - developer.force.com', *Developer.salesforce.com*, 2015. Tillgänglig från: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options. [Citerad: 09- Jun- 2015].

[14] J. Bristowe, 'What is a Hybrid Mobile App? -', *Telerik Developer Network*, 2015. Tillgänglig från: http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/. [Citerad: 09- Jun- 2015].

[15] S. Souders, 'do u webview? | High Performance Web Sites', *Stevesouders.com*, 2015. Tillgänglig från: http://www.stevesouders.com/blog/2014/10/09/do-u-webview/. [Citerad: 09- Jun- 2015].

[16] Cordova.apache.org, 'Apache Cordova', 2015. Tillgänglig från: https://cordova.apache.org/. [Citerad: 09- Jun- 2015].

[17] Phonegap.com, 'PhoneGap | About', 2015. Tillgänglig från: http://phonegap.com/about/. [Citerad: 09- Jun- 2015].

[18] Ionicframework.com, 'Ionic Documentation Overview - Ionic Framework', 2015. Tillgänglig från: http://ionicframework.com/docs/overview/. [Citerad: 09- Jun- 2015].

[19] Ionicframework.com, 'Ionic CLI - Ionic Framework', 2015. Tillgänglig från: http://ionicframework.com/docs/cli/. [Citerad: 09- Jun- 2015].

[20] View.ionic.io, 'The Ionic View App | Share your apps with the world', 2015. Tillgänglig från: http://view.ionic.io/. [Citerad: 09- Jun- 2015].

[21] Software.intel.com, 'Introducing Intel® XDK Development Tools | Intel® Developer Zone', 2015. Tillgänglig från: https://software.intel.com/en-us/xdk/docs/intel-xdk-introduction. [Citerad: 09- Jun- 2015].

[22] App-framework-software.intel.com, 'App Framework', 2015. Tillgänglig från: http://app-framework-software.intel.com/. [Citerad: 09- Jun- 2015].

[23] Docs.telerik.com, 'First Look at Cross-Platform Mobile Development with AppBuilder', 2015. Tillgänglig från: http://docs.telerik.com/platform/appbuilder/. [Citerad: 09- Jun- 2015].

[24] Xamarin.com, 'Mobile Application Development to Build Apps in C# - Xamarin', 2015. Tillgänglig från: http://xamarin.com/platform. [Citerad: 09- Jun- 2015].

[25] Appcelerator, 'Products | Appcelerator Inc.', 2012. Tillgänglig från: http://www.appcelerator.com/product/. [Citerad: 09- Jun- 2015].

[26] Xamarin.com, 'Build a Native Android UI & iOS UI with Xamarin.Forms - Xamarin', 2015. Tillgänglig från: http://xamarin.com/forms. [Citerad: 09- Jun- 2015].

[27] Appcelerator, 'Mobile App Testing | Enterprise Mobile Development | The Appcelerator Platform', 2012. Tillgänglig från: http://www.appcelerator.com/functionaltest/. [Citerad: 09- Jun- 2015].

[28] Docs.appcelerator.com, 'Appcelerator Platform - Appcelerator Docs', 2015. Tillgänglig från: http://docs.appcelerator.com/platform/latest/#!/guide/Titanium_Platform_Overview. [Citerad: 09- Jun- 2015].

[29] Appcelerator, 'Comparing Titanium and PhoneGap', 2012. Tillgänglig från: http://www.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/. [Citerad: 09- Jun- 2015].

[30] Software.intel.com, 'App Security API | Intel® Developer Zone', 2015. Tillgänglig från: https://software.intel.com/en-us/app-security-api/api. [Citerad: 09- Jun- 2015].

[31] C. Mills, 'Web Storage API', *Mozilla Developer Network*, 2014. Tillgänglig från: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API. [Citerad: 09- Jun- 2015].

[32] C. Carlsson, *Testning*, 1st ed. Göteborg: Christer Carlsson, 2012, pp. 2-4, 8-10. Tillgänglig från: http://www.cse.chalmers.se/edu/year/2010/course/TDA143/Lectures/Testning.pdf. [Citerad: 09- Jun- 2015].

[33] Istqbexamcertification.com, 'What is Unit testing?', 2015. Tillgänglig från: http://istqbexamcertification.com/what-is-unit-testing/. [Citerad: 09- Jun- 2015].

[34] Istqbexamcertification.com, 'What is Integration testing?', 2015. Tillgänglig från: http://istqbexamcertification.com/what-is-integration-testing/. [Citerad: 09- Jun- 2015].

[35] Istqbexamcertification.com, 'What is System testing?', 2015. Tillgänglig från: http://istqbexamcertification.com/what-is-system-testing/. [Citerad: 09- Jun- 2015].

[36] Istqbexamcertification.com, 'What is Acceptance testing?', 2015. Tillgänglig från: http://istqbexamcertification.com/what-is-acceptance-testing/. [Citerad: 09- Jun- 2015].

[37] Jasmine.github.io, 'introduction.js', 2015. Tillgänglig från: http://jasmine.github.io/edge/introduction.html. [Citerad: 09- Jun- 2015].

[38] Karma-runner.github.io, 'Karma - How It Works', 2015. Tillgänglig från: http://karma-runner.github.io/0.12/intro/how-it-works.html. [Citerad: 09- Jun- 2015].

[39] Ionic, ‘ion-nav-view - Directive in module ionic’, 2015. Tillgänglig från: http://ionicframework.com/docs/api/directive/ionNavView/