Hybridapputveckling med Ionic Framework

Utveckling av en matbeställningsapplikation för Friends & Brgrs

Petter West

Examensarbete

Informations- och medieteknik

2016

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	5653
Författare:	Petter West
Arbetets namn:	Hybridapputveckling med Ionic Framework Utveckling av en matbeställningsapplikation för Friends & Brgrs
Handledare (Arcada):	Johnny Biström
Uppdragsgivare:	Genero

Sammandrag:

Det här arbetet beskriver planerandet, utvecklandet och testandet av en prototyp för en mobilapplikation (app) för hamburgerrestaurangen Friends & Brgrs. Olika teknologier för app-utveckling presenteras tillsammans med teorin för mobil appdesign. För utvecklingen användes ramverket Ionic Framework som genom hybridteknologin gör det möjligt att bygga en applikation med redan etablerade webstandarder. Som stöd under utvecklingen användes både det officiella hjälpmaterialet samt användargenerad exempelkod och externa kodbibliotek integrerades för att utöka funktionaliteten. Appen kompilerades för Android och konstaterades i testerna fungera enligt specifikationerna. Arbetet bevisar att det är tekniskt möjligt att bygga en app som ser ut och presterar som en nativ app med hjälp av ett ramverk. Med den färdiga prototypen kan användaren beställa måltider på förhand och få bekräftelse från restaurangen i realtid. Prototypen kommer att testas av en fokusgrupp och fortsätta utvecklas.

Nyckelord:	Angular, Friends & Brgrs, Genero, hybridteknologi, Ionic, mobila applikationer
Sidantal:	36
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information and Media Technlogy
Identification number:	5653
Author:	Petter West
Title:	Hybrid app development with Ionic Framework Development of a food ordering application for Friends & Brgrs
Supervisor (Arcada):	Johnny Biström
Commissioned by:	Genero

Abstract:

This work documents the planning, development and testing of a prototype of a mobile application (app) for the hamburger restaurant Friends & Brgrs. Different technologies for app development are presented together with theory for mobile app design. Ionic Framework was used in the development, which through hybrid technology allows apps to be built using existing web standards. During the development both official documentation and user generated sample code was used for support and external code libraries were integrated to extend the functionality of the app. The app was compiled for Android and during testing it was observed to be working according to the specifications. This work proves that it is technically possible to build an app that looks and performs similar to a native app using a framework. The finished prototype allows the user to order meals in advance and get a confirmation in real time. The prototype will be tested by a focus group and development continues.

Keywords:	Angular, Friends & Brgrs, Genero, hybrid technology, Ionic, mobile applications
Number of pages:	36
Language:	Swedish
Date of acceptance:	

INNEHÅLL

Figurer

Termer och förkortningar

AngularJS = ramverk för webbapplikationer, utvecklat av Google och utgivet som öppen källkod

API = applikationsprogrammeringsgränssnitt, från engelskans *application programming interface*; standardiserade kommandon som knyter samman funktioner mellan olika mjukvaror.

app = mobil applikation

cache = tillfällig lagringsplats där data sparas för att inte behöva läsas eller laddas ner på nytt

pushmeddelande = meddelande eller information som skickas till en applikation även när den inte används eller körs. Från engelskans *push notification*

ramverk = återanvändbar programvarumiljö

SASS = *Syntactically Awesome Stylesheets*, ett språk för stilmallar som utvecklats för att försnabba skrivandet av stilmallar

REST = *Representational State Transfer*, gemensamt gränssnitt för att för att överföra kommandon mellan klient och server

WebView = en systemkomponent i mobila operativsystem som kan visa webbsidor inne i applikationer

1 INLEDNING

De mobila användarna står för en allt större del av trafiken på Internet. Samtidigt som konkurrensen om synlighet på sociala medier och digitala marknadsföringskanaler blir allt större inser fler och fler företag vikten av att satsa resurser på att aktivera de som använder sig av mobila enheter. År 2014 spenderade vuxna amerikaner i genomsnitt 2,6 timmar om dagen på att använda mobila enheter, jämfört med 0,4 timmar 2010 (Meeker 2015). På marknaden finns numera en mängd olika så kallade ramverk som ska underlätta och försnabba processen att utveckla mobila applikationer (appar) för flera plattformar. Det här arbetet dokumenterar och utvärderar processen att utveckla en app för en hamburgerrestaurang med hjälp av ett sådant ramverk, *Ionic Framework*.

1.1 Bakgrund

Det här arbetet görs på uppdrag av den digitala marknadsföringsbyrån *Genero Ab* som erbjuder en mängd olika tjänster inom digitala reklamkampanjer och webbutveckling. Företaget sköter om marknadsföringen för hamburgerkedjan *Friends & Brgrs* som är intresserade av en mobil app för sina kunder. Samtidigt är Genero intresserat av att bredda sina erbjudna tjänster med apputveckling, som tidigare krävt särskilda kunskaper i programmering för alla olika operativsystemen, men som tack vare de nya ramverken nu kan göras med samma metoder som inom webbutveckling.

1.1.1 Varför bygga en mobil app?

På grund av dess stora genomslagskraft och distinkta egenskaper räknas mobilen som ett skilt massmedium som inte går att kategorisera tillsammans med några andra. Bland annat så är mobilen personlig, den är alltid ansluten, den är alltid tillgänglig då inspirationen infinner sig och har inbyggt betalningssystem. Dessutom kan den ge detaljerade uppgifter om sina användare och om sociala kontexter. Genom att utnyttja dessa egenskaper kan man bygga appar som vore omöjliga att genomföra på något annat massmedium. (Bruck 2013 s. 34-42)

1.2 Syfte

Syftet med det här arbetet är att bygga en prototyp för en app som kan användas av hamburgerkedjans kunder för att beställa måltider på förhand. Samtidigt utreds hur processen för apputveckling ser ut, hur tidskrävande den är och på vilket sätt användandet av ett ramverk förenklar arbetet. Jag kommer att utreda vilka problem man kan stöta på och vilka kompromisser som kan krävas.

Prototypen kommer att användas för testning bland en begränsad mängd stamkunder. Deras feedback kommer senare att användas för vidareutveckling av appen.

1.2.1 Specifikationer för produkten

Projektet kräver egentligen utvecklandet av två appar: en för konsumenterna att skicka beställningar och en för restaurangen att ta emot beställningarna. Konsumentappen kommer ha följande funktioner:

- Registrering och inloggning
- En meny med alla produkter som restaurangen erbjuder
- Ifall användaren är inloggad, möjlighet att beställa en viss produkt ur menyn, välja tillbehör och få en uppskattning på hur länge beställningen kommer att ta
- Ifall användaren inte är inloggad, hänvisning till inloggningsformuläret då en produkt läggs till
- Bekräftelse av beställningen och kvitto som visas på skärmen

Appen för restaurangpersonalen behöver följande funktionalitet:

- Se inkommande beställningar
- Bekräfta eller avslå en beställning
- Meddela att en beställning är redo

Dessutom kommer appen i framtiden att få flera funktioner i form av stamkundsförmåner, social integrering och stöd för flera olika restauranger, vilket kommer att tas i beaktande

vid planering av appen. Appen ska också kunna publiceras för alla de viktigaste mobilplattformerna (Android, iOS och Windows).

1.2.2 Frågeställningar

Genom det här arbetet hoppas jag få svar på bland annat följande frågor:

- Ger det valda ramverket tekniska möjligheter att utveckla den beskrivna produkten?
- Hur mycket extra kunskaper krävs av utvecklaren?
- Når den färdiga appen upp till de krav som ställs på Android-applikationer?

1.3 Avgränsningar

I arbetet kommer jag att kort presentera olika alternativa ramverk, men utvecklingsprocessen kommer att fokusera på Ionic Framework. En stor fördel med ramverk är att det förenklar utveckling för appar för olika operativsystem, men huvudfokusen för det här arbetet kommer att ligga på Android. Integrationen med betalningssystem behandlas på grund av tidsbrist inte, men processen torde inte skilja sig nämnvärt från integrationer med konventionella webbappar. Eftersom prototypen som byggs inte ännu kommer att publiceras för allmänheten kommer jag inte att ta upp processen för att publicera en app på App Store eller Play Store.

1.4 Struktur

I det första kapitlet *Inledning* presenterar jag bakgrunden till arbetet, mina frågeställningar och begränsningar för projektet. I det andra kapitlet *Teoretisk bakgrund* redogör jag för de förutsättningar och krav som är viktiga att ta i beaktande då man utvecklar en mobil app, både de tekniska och utseendemässiga. Dessutom presenteras de hjälpmaterial som använts vid utveckling. Det tredje kapitlet, *Utveckling av en app*, dokumenterar hela processen för planeringen, utvecklingen och testningen av en app med Ionic Framework. I det avslutande kapitlet *Utvärdering* analyserar jag processen från början till slut, besvarar arbetets frågeställningar och diskuterar resultatet.

2 TEORETISK BAKGRUND

2.1 Typer av appar

Mobilappar kan indelas i tre olika kategorier: nativa, webbaserade och hybrida appar. Varje teknologi har sina egna fördelar och nackdelar. **Nativa appar** har bäst prestanda, har full tillgång till enhetens hårdvara och kan lagra stora mängder data på enheten, men är ofta dyra att utveckla och måste programmeras skilt för varje operativsystem som ska stödas. (Natili 2013 s. 35)

Webbappar bygger på existerande webbteknologier som HTML, CSS och Javascript och fungerar därför likadant på de flesta enheter. Dessutom kan en webbutvecklare bygga en fungerande app utan att behöva lära sig nativ kod. Nackdelen är att appen bara fungerar online och möjligheterna att använda enhetens hårdvara är begränsade. (Charland & Leroux 2011)

En **hybridapp** är en kombination där en del av koden är nativ och resten är HTML-baserad (Cerf 2016). Hybridappar körs i ett mellanlager mellan operativsystemet och webbläsaren som kallas *WebView* vilket möjliggör tillgång till de flesta hårdvarufunktionerna. Apparna fungerar till skillnad från webbappar offline, men förlorar ändå i prestanda mot helt nativa appar (Charland & Leroux 2011).

2.2 Val av ramverk

Det finns ett flertal olika ramverk som underlättar utvecklandet av mobilappar för flera plattformar. Det första som använde sig av hybridteknologin och WebView var PhoneGap som redan 2008 gav ut en version för iOS (Charland & Leroux 2011). Den ursprungliga koden till PhoneGap heter numera Cordova och ägs av Apache, men är utgiven som öppen källkod och fritt att bygga vidare på. Många existerande ramverk, däribland *Ionic*, *PhoneGap* (Lynch 2014) och *appery.io* (Appery.io, u.å.) bygger på Cordovas källkod. Dessutom finns det flera andra ramverk, däribland *Appcelerator Titanium* som inte använder WebView men tolkar om JavaScript till nativ kod (Whinnery 2010) och

Framework 7, som ursprungligen bara hade stöd för iOS men som numera också stöder Android (Brown 2016).

De senaste åren har många nya verktyg dykt upp och de existerande utvecklas snabbt. Ionic är ett relativt nytt ramverk som innehåller flera funktioner som är avsedda att ytterligare underlätta apputveckling. Till skillnad från Cordova och PhoneGap innehåller Ionic färdiga komponenter för mobila användargränssnitt. De är byggda med SASS och AngularJS och påstås ha snarlik prestanda som nativa gränssnitt. Dessutom möjliggör Ionic integrering med betalningstjänster och pushmeddelanden. (Chalkley 2015)

Faktorer som är relevanta då man väljer ramverk är typen av applikation, vilken eller vilka plattformar man ämnar utveckla för och vilka förhandskunskaper man har. I fallet för det här projektet föll valet på Ionic på grund av dess stöd för flera plattformar och mångsidiga verktyg för både utveckling och publicering av appar.

2.3 Gränssnitt

2.3.1 Utformning för mobila användare

När man börjar utforma gränssnittet för en app finns det flera faktorer att ta i beaktande med tanke på att den kommer att användas i huvudsak på mobiltelefoner. McClure m.fl. (2012 s. 61-62) listar några riktlinjer att utgå från:

- Skärmstorleken är begränsad och därför borde bara några valmöjligheter i gången presenteras åt användaren.
- Användaren ska behöva skriva in så lite text som möjligt själv, eftersom det är tidskrävande att skriva på mobiltelefoner. Istället borde förhandsifyllda fält och rullgardinsmenyer prioriteras.
- Så mycket data som möjligt borde sparas på enheten istället för att varje gång laddas ner från nätet.
- Användarna är ofta i rörelse vilket betyder att de lättare trycker på fel knapp ifall de är för små.
- Texten måste vara tillräckligt stor så att den är läsbar också på små skärmar.

- Appen måste gå att använda oavsett ljusförhållande och vara läslig både i direkt solljus och i ett mörkt garage på natten.

För att tackla alla dessa förutsättningar har apputvecklarna tvingats vara kreativa och ta till olika designknep, som med tiden har blivit mer eller mindre standardiserade. Det här betyder att användarna förväntar sig att olika funktioner ska fungera enligt vissa mönster och genom att följa de här mönstren gör man det enklare för användaren att använda appen (Neil 2014 s. 2).

2.3.2 Conversion rate

Inom näthandel använder man begreppet *conversion rate* för att beskriva hur många besök på en sida som leder till ett köp (Moe & Fader 2001). En vanlig orsak till att kunder avbryter sina köp är att nätbutiken inte är tillräckligt användarvänlig, vilket dessutom kan leda till att kundens bild av företaget bakom försämras. Därför borde nätbutiker planeras så att köp är så enkla som möjligt att genomföra (Kuan m.fl. 2005).

Speciellt på mobilen finns det inga rum för misstag då olika formulär för inloggning, registrering och köp designas. Att användaren avbryter att fylla i ett formulär är ett stort problem eftersom det inte leder till något köp. Därför borde antalet inmatningar som krävs av användaren hållas på en minimal nivå. Att automatiskt markera det första fältet i ett formulär sparar ett tryck, medan dubbla fält för till exempel e-post och lösenord tar för mycket extra tid att fylla i på en mobilskärm. Ett bra alternativ är en knapp som tillfälligt visar lösenordet så att användaren kan kontrollera att det är rätt. Om ett användarnamn redan är upptaget eller användaren glömmer fylla i ett fält är det bättre om användaren meddelas direkt i formuläret istället för att få ett felmeddelande. (Neil 2014 s. 56-69)

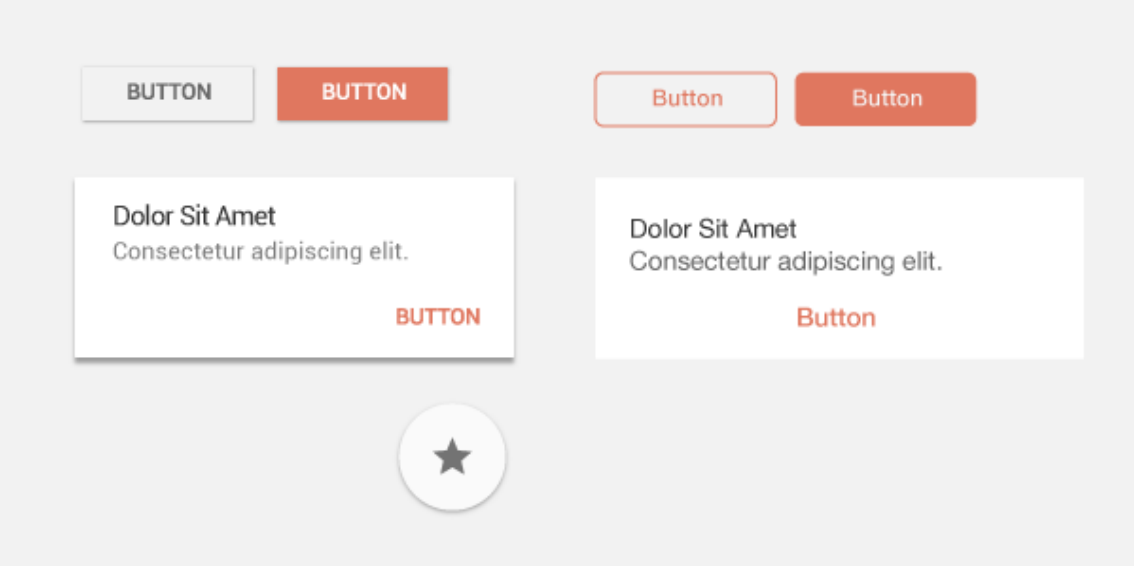
2.3.3 Plattformsspecifika krav

Utöver de generella designkraven så finns det också flera plattformsspecifika egenheter att ta i beaktande när gränssnittet utformas. Apple (2015) kräver att appen följer deras *Apple Human Interface Guidelines* för att den ska accepteras på App Store och ifall användargränssnittet är för komplicerat eller inte tillräckligt väl uttänkt så refuseras appen.

Apple begränsar även typen av innehåll och hur betalningar och pushmeddelanden får användas.

Även Play Store reglerar hurdant innehåll appen får ha men ställer inga krav på utseendet (Google 2016a). Däremot rekommenderar de att man följer deras principer för *Material Design* för att användarna ska få en enhetlig upplevelse.

Flera av riktlinjerna är snarlika, men i fråga om till exempel typografi, navigering, ikoner och knappar (se Figur 1) så har Apple och Google olika rekommendationer (O'Sullivan 2015). Ionic anpassar automatiskt en del av designen till plattformen men för att skräddarsy appen helt för varje plattform krävs det att man använder sig av antingen CSS-klasser, JavaScript eller dynamiska mallar (Drifty Co 2016a).



Figur 1: Jämförelse av utseendet på knappar mellan Android (t.v.) och iOS (t.h.) (O'Sullivan 2015)

2.4 Teknik

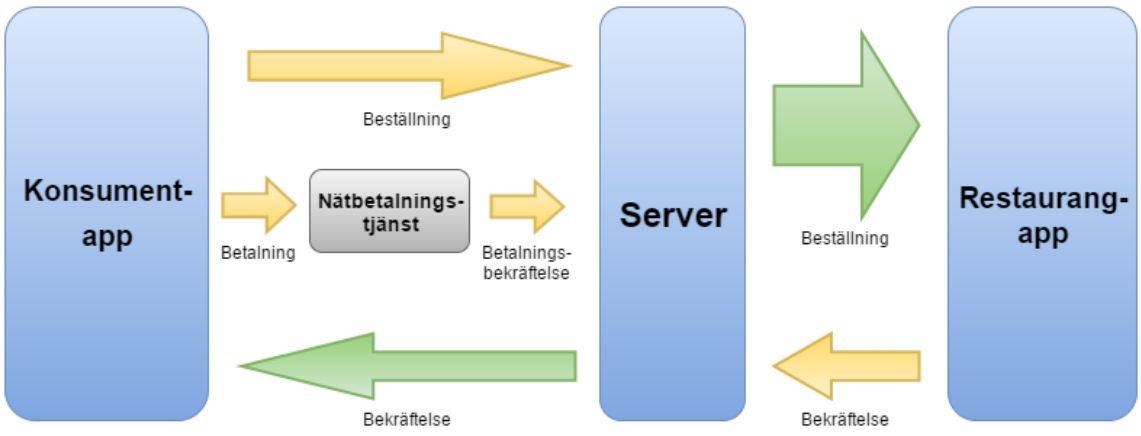
Applikationens gränssnitt byggs upp med HTML5, CSS och JavaScript. För inmatningar och animationer används AngularJS, som ingår i Ionic (Drifty Co 2016a). En server krävs för att förmedla beställningar från kunden till restaurangen. Dessutom behövs en skild

applikation för restaurangen, genom vilken personalen kan ta emot och bekräfta beställningar. Den här restaurangappen kommer att byggas upp på samma sätt som konsumentappen.

För kommunikationen mellan app och server så är API-baserade backend-tjänster (*BaaS = Backend as a Service*) populära. Tjänsterna erbjuder en färdig backend särskilt utformad för appar och data sparas i en molnserver. Det här besparar utvecklaren från att själv bygga upp serverstrukturer för datalagring, autentisering och integrationer med andra tjänster, dessutom med skilda konfigurationer för varje plattform. (Riggins 2015)

2.4.1 Beställningslogik

När kunden gör en beställning skickas den från konsumentappen till en server, som skickar den vidare till restaurangappen som meddelar personalen om en inkommande beställning (se Figur 2). När personalen bekräftat beställningen skickas den genom servern tillbaka till konsumentappen. Både då en beställning mottas av restaurangappen och konsumentappen tar emot bekräftelsen borde kommunikationen ske i realtid så att inga fördröjningar sker. En lösning för det här är *Socket.io* som är ett JavaScript-ramverk som gör det möjligt för en server att skicka data till klienten utan att klienten ber om det (Krill 2014).



Figur 2: Schematisk bild över hur en beställning kommer att fungera.

2.4.2 Caching

Eftersom tillgången till nätverk varierar måste tillräckligt mycket data sparas lokalt på enheten i en cache för att appen ska fungera utan uppkoppling. Ifall appen ständigt måste hämta data över nätverket påverkar det också batteritiden (Harlalka 2014). Eftersom Ionic använder sig av webbteknologier används webbläsarens cache och strängar kan sparas i Local Storage (Lynch, u.å.). Dessutom utförs caching på enskilda vyer för att förbättra prestandan (Drifty Co 2016a).

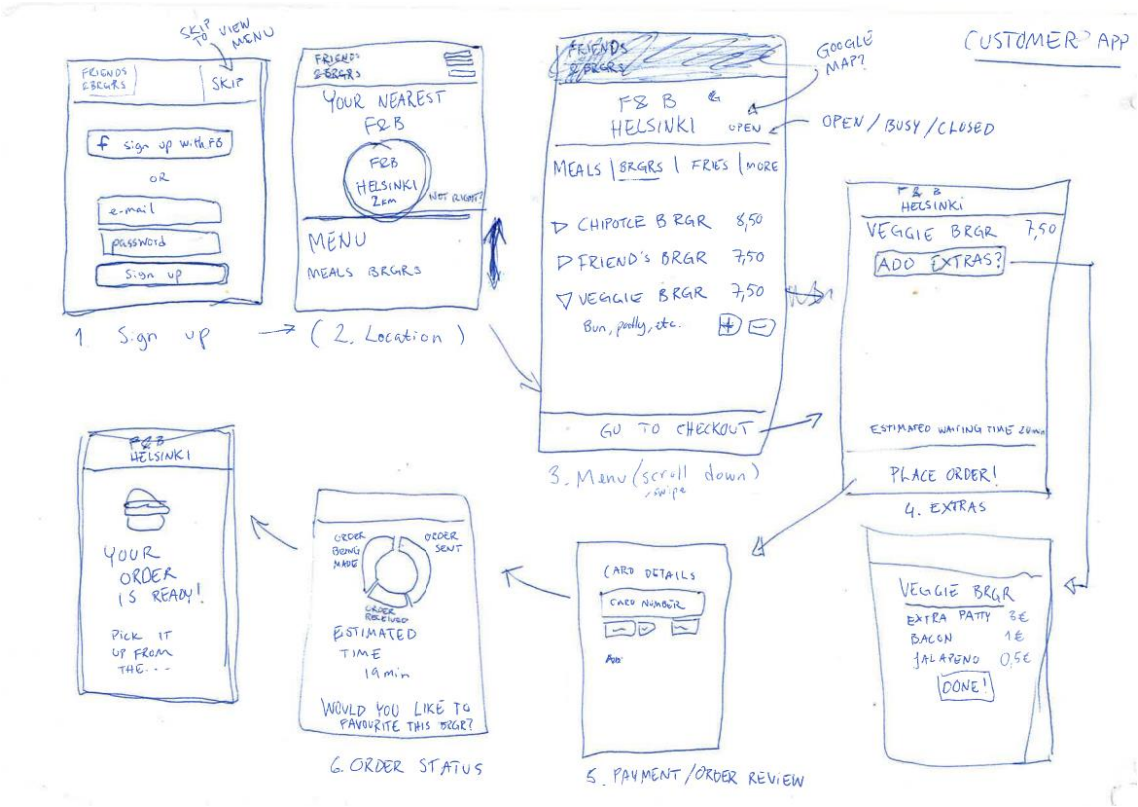
2.5 Hjälpmaterial

Ionic erbjuder ett omfattande material av instruktioner och guider på sin hemsida. Förutom instruktioner för installation på olika operativsystem finns en guide som beskriver utvecklingsprocessen för en exempelapp från början till slut. Alla CSS-komponenter och JavaScript-funktioner finns beskrivna, de flesta med fungerande exempelkoder bifogade. Dessutom finns ett supportforum där registrerade användare kan be om hjälp. (Drifty Co 2016a)

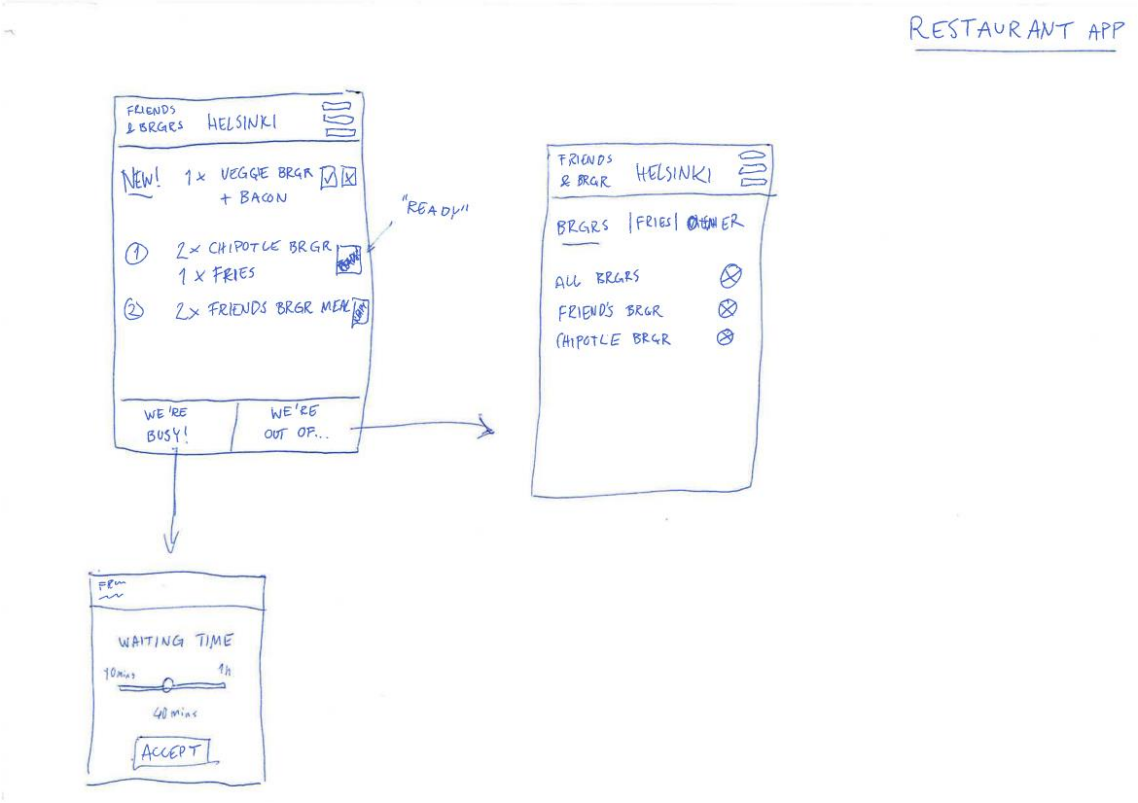
För funktioner som kodas i Angular kan också Angulars hemsida (Google 2016b) konsulteras för instruktioner, guider och annat hjälpmaterial. En annan betydande källa för hjälpmaterial är Stackoverflow (Stack Exchange Inc, 2016) som har över 12 000 frågor taggade med ”ionic-framework” och nästan 170 000 frågor taggade med ”angularjs”.

3 UTVECKLING AV EN APP

När specifikationerna för appen var klara så gjordes skisser upp för både användar- och restaurangappen (se Figur 3 respektive Figur 4). Skisserna visade alla vyer och interaktioner från inloggning till beställning och kommunikation med restaurangen. De faktorer gällande designstandarder, conversion rate och användarvänlighet som togs upp i kapitel 2.2 beaktades och skisserna kommenterades av en designer och godkändes av klienten innan själva utvecklingen av appen inleddes.



Figur 3: Skiss av konsumentappen



Figur 4: Skiss av restaurangappen

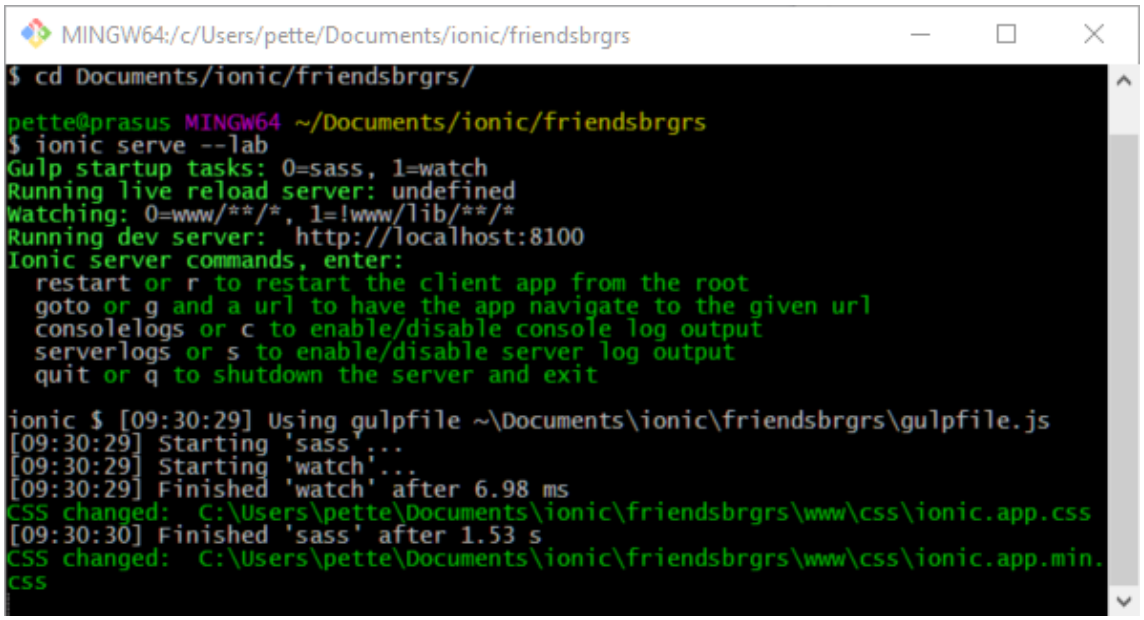
3.1 Verktyg

Det första steget i utvecklingen var att installera alla nödvändiga mjukvaruverktyg. För att installera Ionic krävdes *Node.js* (Node.js Foundation 2016). När det fanns installerat kunde Ionic installeras genom kommandot

```
$ npm install -g ionic
```

För Windows, som användes under det här arbetet, rekommenderar Ionics utvecklare att man installerar Visual Studio för att automatiskt få med alla verktyg som behövs. (Drifty Co 2016b)

Ionic inkluderar en kommandotolk, *Ionic CLI* (se Figur 5), genom vilken det går att starta, kompilera, testa och emulera appar (Drifty Co 2016a). För att köra kommandona användes *Git Bash*. Git användes även som versionshanteringssystem under projektets gång.



Figur 5: Ionics kommandotolk Ionic CLI.

3.1.1 Versioner

För det här arbetet användes den senaste stabila versionen av Ionic, version 1.3.0, som är byggt på AngularJS. När arbetet skrevs var Ionic 2 i betastadiet och en frågeställning som uppkom var om det skulle löna sig att använda den nya versionen som utlovade många förbättringar och nya funktioner (Bradley 2016). Ionic 2 bygger på Angular 2 som också innehåller många förändringar, men som också befann sig i betastadiet (Lynch 2015).

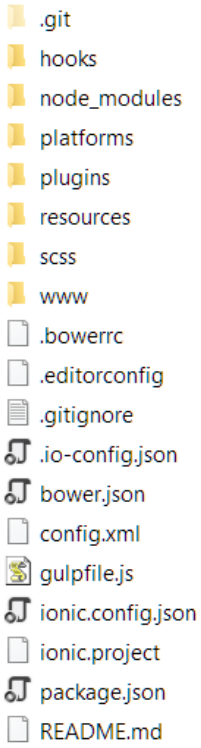
Att använda de nya betaversionerna skulle ha möjliggjort flera funktioner och modernare kod i appen, men kunde samtidigt ha inneburit problem i form av buggar och funktioner som ännu var halvfärdiga. Officiellt hjälpmaterial för både Ionic 2 och Angular 2 fanns redan, men största delen av det användargenererade hjälpmaterialet samt exempelkoder och integrationer med andra tjänster var endast tillgängligt för det äldre versionerna. Av de här orsakerna användes de äldre, stabila versionerna.

3.2 Start av projektet

När alla verktyg var installerade startades applikation genom kommandot

```
$ ionic start friendsbrgrs
```

Kommandot laddade ner alla filer som behövs och skapade filstrukturen (se Figur 6) med konfigurationsfiler och en baslayout att utgå ifrån. Det mesta av arbetet på appen skedde i mappen *www/*. All HTML-kod, JavaScript, bilder och CSS som byggde upp gränssnittet för appen placerades där i respektive filer. SASS-filer placerades i mappen *scss/*.



Figur 6: Filstrukturen för en Ionic-app.

För att aktivera appen för Android kördes sedan kommandot

```
$ ionic platform add android
```

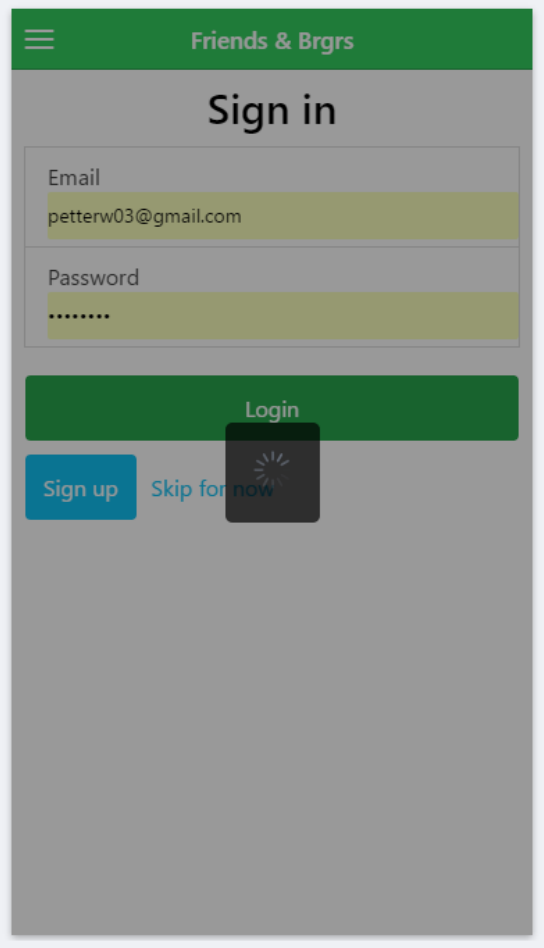
vilket skapade alla nödvändiga filer i mappen *platforms/* som krävdes för att senare kunna kompilera appen för Android.

3.3 Utveckling av frontend

Gränssnittet för appen byggdes till största delen upp med hjälp av Ionics färdiga, mobilanpassade komponenter och Angular-direktiv som sedan anpassades med SASS. Medan designelement för iOS var välrepresenterade så saknades många typiska navigeringsmönster för Android. Dessutom följde inte komponenterna de senaste riktlinjerna för

Googles Material Design, men bättre stöd utlovas i kommande versioner av Ionic (Bradley 2016).

Eftersom en del navigeringsmönster som hade skissats upp i designen inte fanns färdigt tillgängliga och visade sig vara antingen svåra eller omöjliga att koda, så gjordes en del avvikningar från skisserna och vissa element ersattes med andra, enklare lösningar. Andra fördefinierade funktioner, som mobilvänliga formulär, sidomeny och laddningsskärmar (se Figur 7), gick däremot smärtfritt att sätta till och anpassa efter behov.



Figur 7: Laddningsskärm under inloggning.

3.3.1 Konsumentappen

Först byggdes vyn med listan på tillgängliga produkter upp. Enligt den ursprungliga planen skulle de olika kategorierna delas in i flikar, men då de inte gick att implementera

enligt Androids navigeringsmönster byttes de ut mot en enkel, lång lista baserad på Ionics CSS-komponent *list* (se Figur 8). Produkterna läses in från en array och skrivs ut med Angulars *ngRepeat*. Angular användes också för att filtrera produkterna enligt kategori och för att koda en funktion som utvidgar den valda produkten med mera information.

```
<ion-list>
  <div ng-repeat="category in brgrs.categories">
    <div class="item item-divider">
      {{category.name}}
    </div>
    <div ng-repeat="item in brgrs.items | filter: { category: category.id}">
      <ion-item class="item-icon-left" ng-click="brgrs.toggleGroup(item)" ng-class="{active: isGroupShown(item)}">
        <i class="icon" ng-class="brgrs.isGroupShown(item) ? &amp;apos;ion-arrow-down-b&amp;apos; : &amp;apos;ion-arrow-right-b&amp;apos;"></i>
        {{item.name}}
        <span class="item-note">
          {{item.price}} &#x20AC;
        </span>
      </ion-item>
      <div class="item-accordion padding" ng-show="brgrs.isGroupShown(item)">
        <div class="row">
          <div class="col">
            {{item.description}}
          </div>
          <div class="col">
            <ngcart-addtocart id="{{ item.id }}" name="{{ item.name }}" price="{{ item.price }}" quantity="1"></ngcart-addtocart>
          </div>
        </div>
      </div>
    </div>
  </div>
</ion-list>
```

Figur 8: Kodsnutt som listar alla tillgängliga produkter enligt kategori.

```
1  <ion-view view-title="Friends & Brgrs">
2    <ion-content class="padding has-header">
3      <form ng-submit="login.signin()">
4
5        <h2 class="text-center">Sign in</h2>
6
7        <div class="list">
8          <label class="item item-input item-stacked-label">
9            <span class="input-label">Email</span>
10           <input id="email" name="email" type="email" placeholder="Your email" ng-model="login.email">
11         </label>
12         <label class="item item-input item-stacked-label">
13           <span class="input-label">Password</span>
14           <input id="password" name="password" type="password" placeholder="Your password" ng-model="login.password">
15         </label>
16       </div>
17       <div class="">
18         <button class="button button-block button-balanced" type="submit">
19           Login
20         </button>
21       </div>
22
23     </form>
24
25     <h4 class="assertive" ng-bind="login.error"></h4>
26
27     <button class="button button-calm" ui-sref="signup">Sign up</button>
28
29     <button class="button button-clear button-calm" ng-click="login.anonymousLogin()">Skip for now</button>
30   </ion-content>
31 </ion-view>
```

Figur 9: HTML-kod för inloggningsvyn.

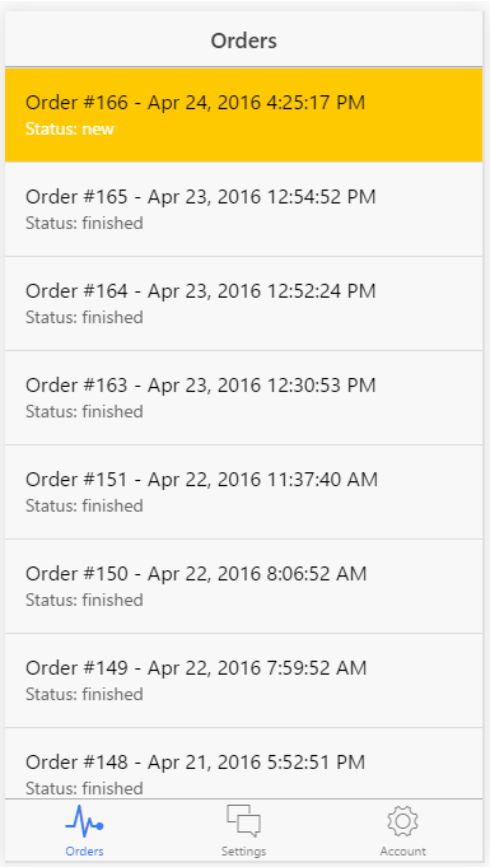
Angular-modulen *ngCart* (Github 2016) integrerades i koden för att hantera köpvagnen och uträkning av totala priset och momsen på en beställning. Efter det gjordes sidorna för beställningsbekräftelse och tilläggsförsäljning enligt skisserna. Till sist skapades sidor för

inloggning (se Figur 9) och användarregistrering med hjälp av Ionics färdiga formulärkomponenter. I registreringsformuläret programmerades en knapp som tillfälligt visar lösenordet så att användaren kan kontrollera att det matats in rätt.

3.3.2 Restaurangappen

Restaurangappen byggdes på samma sätt som konsumentappen. Istället för en meny med alla produkter så gjordes startvyn till en lista med alla inkommande beställningar (se Figur 10). En stor del av koden kunde återanvändas, med modifikationen att restaurangappen läser in alla beställningar och inte bara användarens egna beställningar.

Ingen sidomeny sattes till i restaurangappen, istället användes komponenten *tabs* för att skapa tre flikar för appens olika vyer. För inkommande beställningar skapades två olika valmöjligheter som visas då beställningen sveps åt sidan: bekräfta eller avslå. Ifall beställningen accepteras ersätts valen med en knapp för att meddela att beställningen är klar.



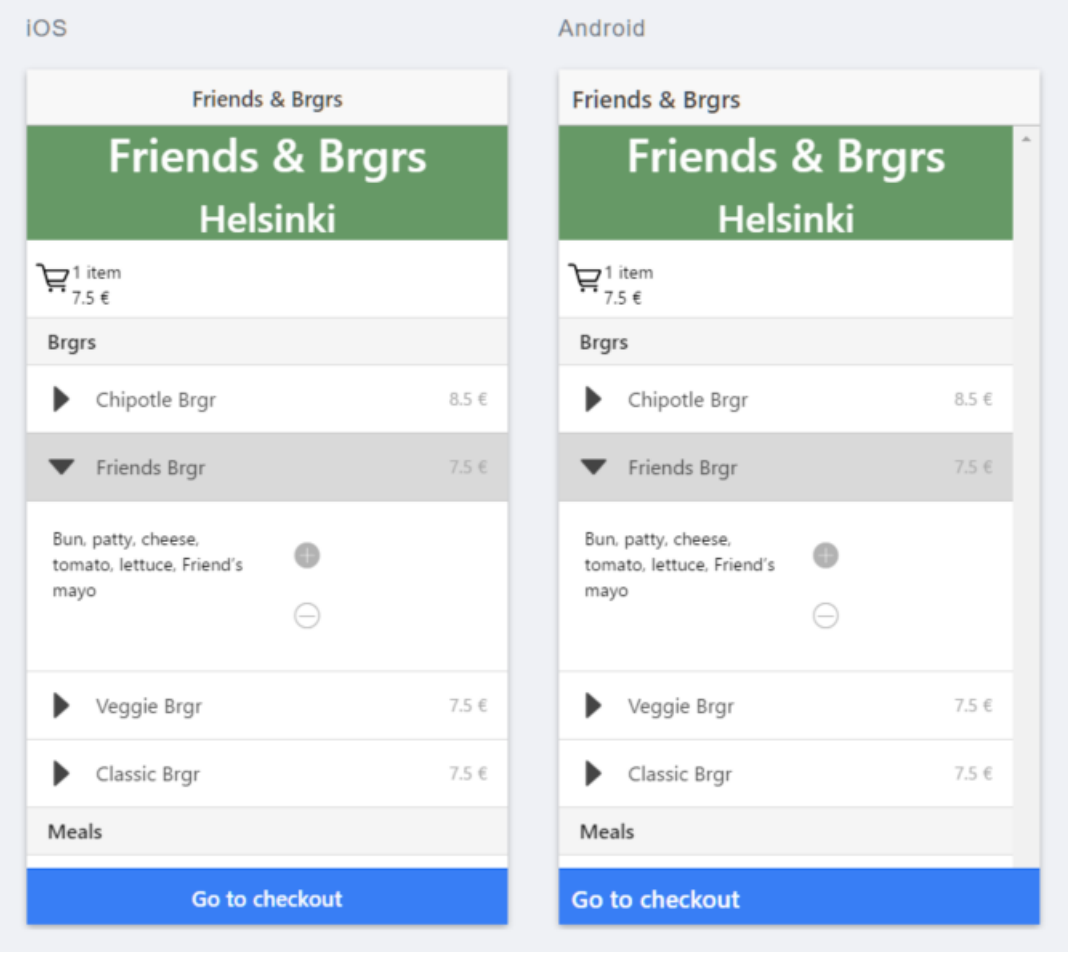
Figur 10: Inkommande beställningar visas i restaurangappen.

3.3.3 Förhandsvisning

Under arbetets gång kunde appen förhandsvisas i webbläsaren genom kommandot

```
$ ionic serve
```

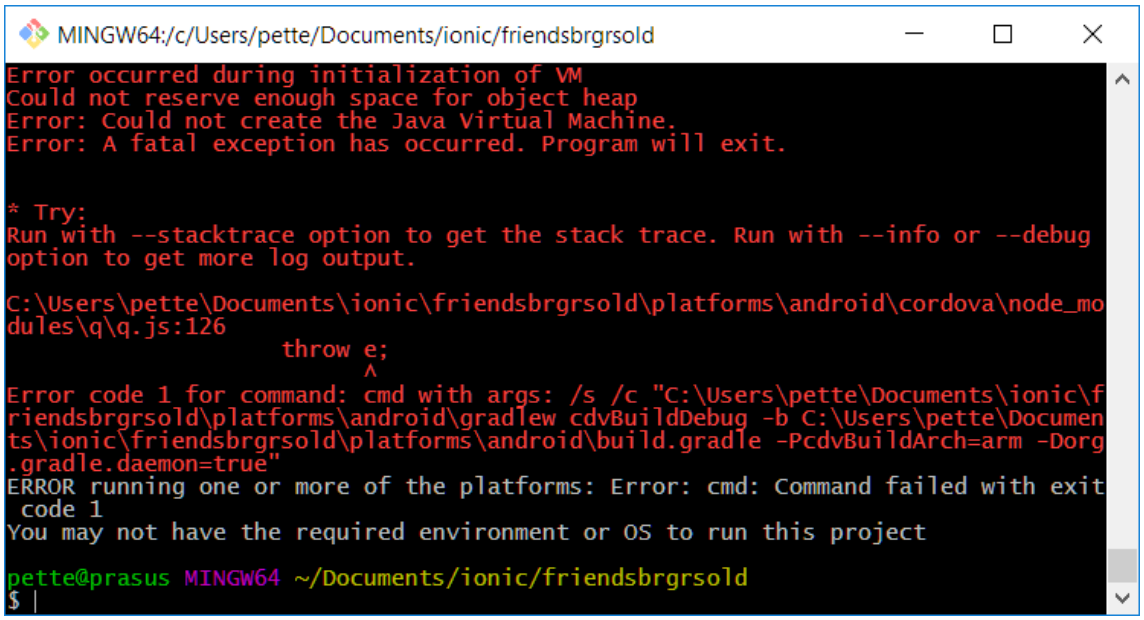
Genom att sätta till ändelsen *--lab* simulerades både iOS- och Androidversionerna av appen bredvid varandra (se Figur 11). Serve använder *LiveReload* vilket betyder att webbläsaren automatiskt laddas om varje gång en ändring görs i koden (Drifty Co 2016a). SASS-filer kompileras också automatiskt till CSS.



Figur 11: En tidig prototyp av mobilappen förhandsvisad i Ionic Lab.

För att testa appen på en Android-enhet fanns det två olika möjligheter, antingen att installera appen direkt på enheten med en USB-kabel eller genom appen *Ionic View* som kunde laddas ner på enheten för att sedan köra den egna appen. För att installera appen

direkt behövde först USB-felsökning aktiveras på enheten. Sedan kördes *ionic build android* för att bygga appen och sedan föra över den till enheten. Det här kommandot gav först felmeddelanden i kompileringsprocessen (se Figur 12), som löstes genom att höja det allokerade minnet i filen *gradle.properties*.



Figur 12: Felmeddelande vid byggprocessen för Android.

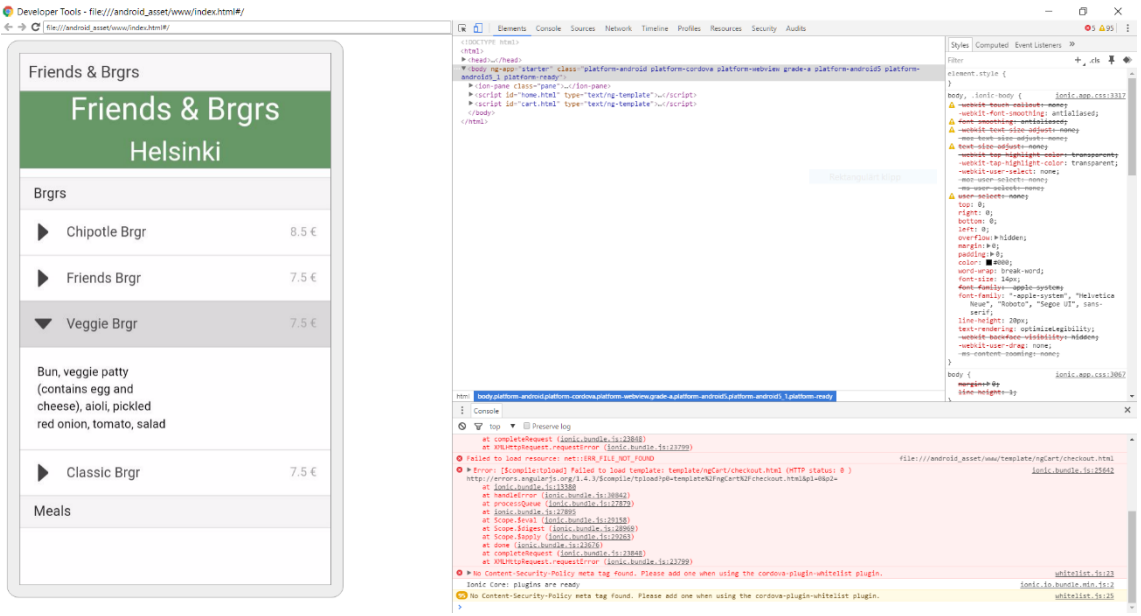
Android-appar kan också testas utan en fysisk enhet genom emulering, men det här rekommenderas inte eftersom emulatorn är långsam och inte representerar en riktig enhet (Drifty Co 2016a).

3.3.4 Felsökning

För att felsöka de problem och buggar som uppkom i utvecklingsprocessen användes till största delen inspektorn i Chrome. Buggar i JavaScript-koden skrev oftast ut felmeddelanden i konsolen som ledde till den felande raden i källkoden, och i andra fall kunde *console.log* sättas till i koden för att skriva ut extra data.

Fastän det ofta räckte att felsöka med Ionic Serve visade det sig att den metoden inte alltid var pålitlig, eftersom vissa buggar endast uppdagade sig på enheten. I ett fall där en sökväg skrivits med liten begynnelsebokstav i misstag, fungerande appen perfekt i Ionic

Serve eftersom Windows inte skiljer på stora och små bokstäver, men i Android hittades inte den rätta filen. Det här problemet hittades också med hjälp av inspektorn i Chrome, som kan användas för att felsöka direkt på enheten då den är inkopplad genom USB (se Figur 13). Då Ionic View användes fanns inte den här möjligheten.



Figur 13: Felsökning på en Android-telefon med Chromes inspektor.

Andra problem berodde på Ionics inbyggda caching. Ifall användaren först använde appen i gästläge och sedan loggade in så visade appen fortfarande de gamla vyerna. För att säkerställa att alla vyer uppdaterades så sattes en funktion till som laddar om appen då in- och utloggning sker.

3.4 Integration med backend

För att kunna fokusera på utvecklingen av front-end användes tjänsten *Backand* (2016) för back-enden och för alla serverfunktioner. Datatabeller för produkter, produktkategorier, användare och beställningar skapades och kopplades till appen genom en REST API. Genom HTTP-kommandona POST och GET skapas respektive hämtas data i databasen (se Figur 14). Autentisering och inloggning sköttes också genom Backand, som erbjöd exempelkod för Ionic-applikationer.

```
service.getOrders = function() {
  return $http({
    method: 'GET',
    url: service.getUrl('orders'),
    params: {
      sort: JSON.stringify([{
        fieldName: "id",
        order: "desc"
      }])
    }
  });
};
```

Figur 14: Kodsnutt som hämtar beställningar från servern.

Realtidsfunktionerna integrerades snabbt med hjälp av Socket.io. Det räckte att sätta till en hänvisning till Socket-biblioteket i appen och sätta till en kodsnutt som lyssnar efter utannonseringar från servern och läser in fälten på nytt då en ändring registrerats.

3.5 Kompilering för Android

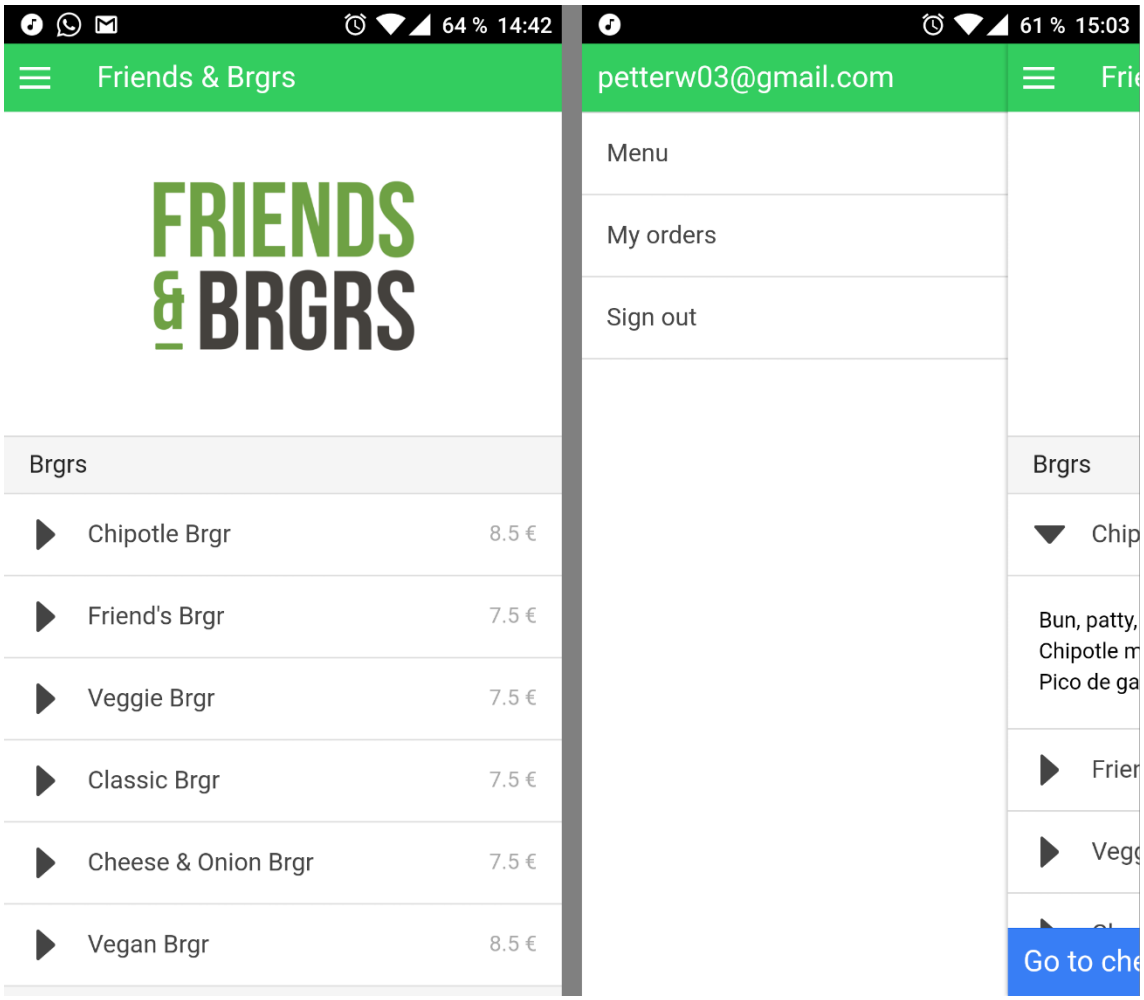
Då en prototyp av appen färdigställts och konstaterats fungera så var det sista steget av utvecklingen att förbereda appen för lansering och kompilera den för Android. Bildfiler för ikon och startskärm sattes till i mappen *resources/*, utifrån vilka Ionic automatiskt skötte generandet av alla nödvändiga storlekar. Innan kompileringen togs överflödiga tilllägg bort och nödvändig information sattes till i *config.xml* enligt Ionics guide (Drifty Co 2016c). Dessutom kördes verktygen *lint* och *uglify* på JavaScript-koden för att förhindra att den kan dekompileras (Raboy 2015). Sedan kördes kommandot

```
$ cordova build --release android
```

Processen kördes utan problem och skapade en fil vid namn *android-release-unsigned.apk*. För att kunna installera appen krävdes det också att den signerades, vilket gjordes genom att köra verktygen *keytool* och *jarsigner*. Till sist kördes verktyget *zipalign* som optimerar appen för att minska RAM-användningen. (Android Developers u.å. a)

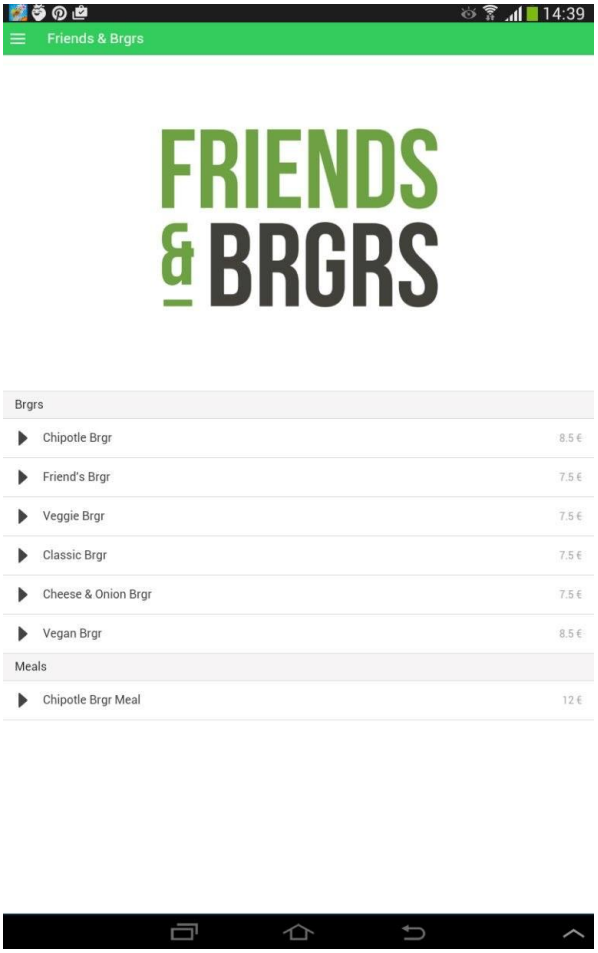
3.6 Testning

Den färdiga appen installerades och testades på tre olika enheter, två smarttelefoner och en surfplatta. De testade mobiltelefonerna var en OnePlus X (se Figur 15) med Android version 5.1.1 och en Huawei Honor 7 med samma version. Surfplattan var en Samsung Galaxy Pad 2 (se Figur 16) med Android version 4.2.2. I testerna så följdes alla steg i den rekommenderade proceduren för kvalitetstestning av Android-applikationer, som innebär att alla vyer testas i både stående och liggande läge, att minimering av appen och byte till en annan app, bakåtknappens funktion, låsning av skärmen, inmatning av text och andra funktioner kontrolleras (Android Developers u.å. b).



Figur 15: Den färdiga appen körs på en OnePlus X. På den högra bilden är sidormenyn öppen.

I de första testerna på OnePlus One-enheten så betedde sig inte bakåt-knappen som den skulle; ibland gick den framåt i historiken istället för bakåt eller stängde oväntat ner appen. Det här löstes efter att Ionic uppdaterades till den senaste stabila versionen.



Figur 16: Appen körs på en Samsung Galaxy Pad.

I testerna kontrollerades också att inte appen utförde några onödiga bakgrundsaktiviteter eller krävde fler behörigheter än nödvändigt. Flera versioner av appen kompilerades och installerades för att testa att appen smidigt kan uppdateras med nya funktioner.

På de båda smarttelefonerna fungerade appen enligt förväntningarna. På Samung Galaxy Pad-surfplattan fungerade inte alltid inloggningen. Problemet var troligen relaterat till den föråldrade Android-versionen (4.2.2) på enheten, trots att Ionic officiellt (Drifty Co 2016d) stöder Android från version 4.1 uppåt.

4 UTVÄRDERING

Den färdiga appen (se Bilaga 1 och Bilaga 2) innehåller all nödvändig funktionalitet och alla vyer som specificerades i planeringen, med några undantag. Tilläggsförsäljningen är endast halvfärdig och registreras inte. Dessutom förekommer en del buggar vid in- och utloggning.

Efter planering och installering av verktygen tog den aktiva fasen av utvecklingen ungefär fyra hela veckor i anspråk. Utvecklingen av appen fortsätter och planen är att den ska testas bland kunder sommaren 2016. Innan dess kommer alla produkter att matas in och ett betalningssystem ska integreras. Pushmeddelanden borde också sättas till för att kunna meddela användaren om att en beställning är klar också då appen inte är öppen. Dessutom borde appens utseende förbättras och texter ses över.

Överlag gick processen som väntat och jag stötte inte på några oöverkomliga problem. Eftersom jag inte tidigare använt AngularJS hade jag inte tänkt använda det för appen mer än nödvändigt men vartefter arbetet fortskred lärde jag mig Angular och insåg vilka möjligheter det tillförde.

De flesta av Ionics komponenter, framförallt sidomenyn, dialogrutor och formulär var enkla att sätta till och anpassa efter behov. Sveprörelser och det inbyggda tangentbordet fungerade direkt utan någon extra konfiguration. Dock var de inbyggda komponenterna något föråldrade och till exempel de svepbara flikarna jag planerat för min app visade sig vara väldigt svåra att göra och därför beslöt jag att kompromissa och inte ta med dem i den här versionen.

En stor fördel med att appen baseras på JavaScript och Angular är den stora mängd tillgängliga bibliotek som kan integreras. För det här projektet använde jag mig av både Angular-biblioteket ngCart och JavaScript-biblioteket Socket.io.

Felsökning och testning gick enkelt tack vare det inbyggda verktyget Ionic Serve och appen Ionic View. Eftersom Ionic är ett populärt ramverk fanns det många användargenerade guider att vända sig till då det officiella materialet inte räckte till och vid problem

hittades lösningen oftast genom att googla på olika hjälpforum. Att endast felsöka appen i Ionic Serve var inte alltid helt pålitligt och då var USB-felsökningen på Android till stor hjälp.

Kompileringen av appen var inte så enkel som jag hade hoppats och det krävdes flera manuella steg för signering och optimering innan appen var redo att installeras på en Android-enhet. Förhoppningsvis kan de här stegen automatiseras i kommande versioner av Ionic.

5 SLUTSATSER

Målet med det här arbetet var att med hjälp av ett ramverk bygga en fungerande app för hamburgerrestaurangen Friends & Brgrs, vilket utfördes inom utsatt tid. I utvecklingen användes ramverket Ionic Framework genom vilket redan etablerade webbtekniker kunde användas för att bygga en så kallad hybridapp. Fastän appen kompilerades och testades endast för Android, ger ramverket möjlighet att i framtiden kompilera versioner också för andra plattformar utan ändringar i koden.

I praktiken byggdes två skilda appar som kan kommunicera med varandra och båda apparna konstaterades fungera som önskat. Med konsumentappen kan en kund logga in på sitt eget konto, se på menyn med produkter och göra en beställning. Restaurangappen låter personalen ta emot och bekräfta inkommande beställningar i realtid. Med hjälp av externa, fritt tillgängliga kodbibliotek kunde de önskade funktionerna förverkligas. Appen följer tillräckligt långt Androids rekommendationer för gränssnitt och klarade på nyare telefonmodeller de tester som utfördes. På äldre versioner av Android förekom en del buggar vars orsak ännu är oklar.

Arbetet bevisade att det är fullt tekniskt möjligt att göra en relativt avancerad applikation med hjälp av ett ramverk. Att som traditionell webbutvecklare direkt bygga mobilappar utan någon erfarenhet eller förhandsplaneringen är ändå inte realistiskt, eftersom plattformen ställer nya krav och kräver ett annat tänkesätt. Ionic innehåller färdiga visuella komponenter som överbygger klyftan till mobilvänlig design, men har samtidigt en del begränsningar som gör att kompromisser kan krävas. Webbteknikerna utvecklas ändå snabbt och nya versioner av Ionic utlovar förbättrad funktionalitet. Det här betyder förhoppningsvis att steget mellan webbutveckling och apputveckling i framtiden kommer att vara ännu mindre.

KÄLLOR

Android Developers. (u.å. a). *Signing Your Applications.* Tillgänglig: http://developer.android.com/tools/publishing/app-signing.html Hämtad 22.4.2016.

Android Developers. (u.å. b). *Core App Quality.* Tillgänglig: http://developer.android.com/distribute/essentials/quality/core.html Hämtad 24.4.2016.

Appery.io. (u.å.). *Datasheet (white paper).* Tillgänglig: https://appery.io/wp-content/uploads/ApperyioDataSheet.pdf Hämtad 15.3.2016.

Apple. 2015. *App Store Review Guidelines*. Tillgänglig: https://developer.apple.com/app-store/review/guidelines/ Hämtad 25.2.2016.

Backand. 2016. *Backand Documentation*. Tillgänglig: http://docs.backand.com/en/latest/index.html Hämtad 11.4.2016.

Bradley, Adam. 2016. *Announcing Ionic Framework 2 Beta*. Tillgänglig: http://blog.ionic.io/announcing-ionic-framework-2-beta/ Hämtad 5.4.2016.

Brown, Eric. 2016. *10 Best Free Mobile Application Development Frameworks That Support Android.* Tillgänglig: https://www.linux.com/news/embedded-mobile/mobile-linux/882122-10-best-open-source-mobile-frameworks-that-support-android Hämtad 21.3.2016.

Bruck, Peter & Rao, Madanmohan. 2013, *Global Mobile: Applications and Innovations for the Worldwide Mobile Ecosystem*, Medford, NJ: Information Today, Inc, 633 s.

Cache. (u.å.). I *Svenska Datatermgruppen*. Tillgänglig: http://www.datatermgruppen.se/ Hämtad 20.4.2016.

Cerf, Vinton. 2016, Apps and the Web, *Communications of the ACM,* Volume 59 Issue 2, s. 7. Tillgänglig: http://cacm.acm.org/magazines/2016/2/197413-apps-and-the-web/fulltext

Chalkley, Andrew. 2015. *Why Ionic is Reigniting the Native vs HTML5 Debate*. Tillgänglig: http://blog.teamtreehouse.com/ionic-reigniting-native-vs-html5-debate Hämtad 18.1.2016.

Charland, Andre & Leroux, Brian. 2011. Mobile Application Development: Web vs. Native. *Communications of the ACM,* Volume 54 Issue 5, s. 49-53. Tillgänglig: http://cacm.acm.org/magazines/2011/5/107700-mobile-application-development/fulltext

Drifty Co. 2016a. *Ionic Documentation*. Tillgänglig: http://ionicframework.com/docs/. Hämtad 25.2.2016.

Drifty Co. 2016b. *Getting Started with Ionic*. Tillgänglig: http://ionicframework.com/getting-started/ Hämtad 21.3.2016.

Drifty Co. 2016c. *Chapter 6: Publishing your app.* Tillgänglig: http://ionicframework.com/docs/guide/publishing.html Hämtad 22.4.2016.

Drifty Co. 2016d. *Chapter 2: Installation*. Tillgänglig: http://ionicframework.com/docs/guide/installation.html Hämtad 24.4.2016.

Github. 2016. *ngCart*. Tillgänglig: https://github.com/snapjay/ngcart Hämtad 23.4.2016.

Google. 2016a. *Policy guidelines & practices.* Tillgänglig: https://support.google.com/googleplay/android-developer/answer/113474?hl=en. Hämtad 25.2.2016.

Google. 2016b. *Angular Docs.* Tillgänglig: https://angular.io/docs/ts/latest/ Hämtad 28.3.2016.

Harlalka, Rajat. 2014. *How to stop your mobile app from being a serious battery drain.* Tillgänglig: http://thenextweb.com/dd/2013/08/05/how-to-minimize-your-mobile-apps-power-consumption/#gref Hämtad 28.3.2016.

Krill, Paul. 2014. *Socket.IO JavaScript framework ready for real-time apps*. Tillgänglig: http://www.infoworld.com/article/2607757/javascript/socket-io-java-script-framework-ready-for-real-time-apps.html Hämtad 10.4.2016.

Kuan, Huei Huang, Bock, Gee-Woo & Vathanophas, Vichita. 2005. Comparing the effects of usability on customer conversion and retention at e-commerce websites, *Proceedings of the 38th Hawaii International Conference on System Sciences*.

Lynch, Max. 2014. *The Last Word on Cordova and PhoneGap.* Tillgänglig: http://blog.ionic.io/what-is-cordova-phonegap/ Hämtad: 15.3.2016.

Lynch, Max. 2015. *Learn Angular 2.* Tillgänglig: http://learnangular2.com/why-angular2 Hämtad 9.4.2016.

Lynch, Max. (u.å.). *Using Local Storage.* Tillgänglig: http://learn.ionicframework.com/formulas/localstorage/ Hämtad 22.3.2016.

McClure, Wallace B., Blevins, Nathan & Croft, John J. 2012. *Professional Android Programming with Mono for Android and .NET/C#,* Hoboken, NJ, USA: Wrox, 556 s.

Meeker, Mary. 2015. *2015 Internet Trends*. Tillgänglig: http://www.kpcb.com/blog/2015-internet-trends Hämtad 14.1.2016.

Moe, Wendy & Peter S. Fader. 2001. Which visits lead to purchases? Dynamic conversion behavior at e-commerce sites. T*he Wharton School, Working Paper 00 23 (2000)* s. 7-18.

Natili, Giorgio. 2013. *PhoneGap 3 Beginner's Guide*, Birmingham: Packt Publishing Ltd, 289 s.

Neil, Theresa. 2014, *Mobile design pattern gallery: UI patterns for smartphone apps,* 2nd ed. edn, O'Reilly, Sebastopol, CA.

Node.js Foundation. 2016. *Node.js*. Tillgänglig: https://nodejs.org/en/ Hämtad 24.4.2016.

O'Sullivan, Chris. 2015. *A Tale of Two Platforms: Designing for Both Android and iOS.* Tillgänglig: http://webdesign.tutsplus.com/articles/a-tale-of-two-platforms-designing-for-both-android-and-ios--cms-23616 Hämtad 25.2.2016.

Raboy, Nick. 2015. *Minifying Your App's Source Code*. Tillgänglig: http://blog.ionic.io/minifying-your-source-code/ Hämtad 24.4.2016.

Riggins, Jennifer. 2015. *Why You Should Build Apps With An API Backend – BaaS.* Tillgänglig: http://nordicapis.com/why-you-should-build-apps-with-an-api-backend-baas/ Hämtad 10.4.2016.

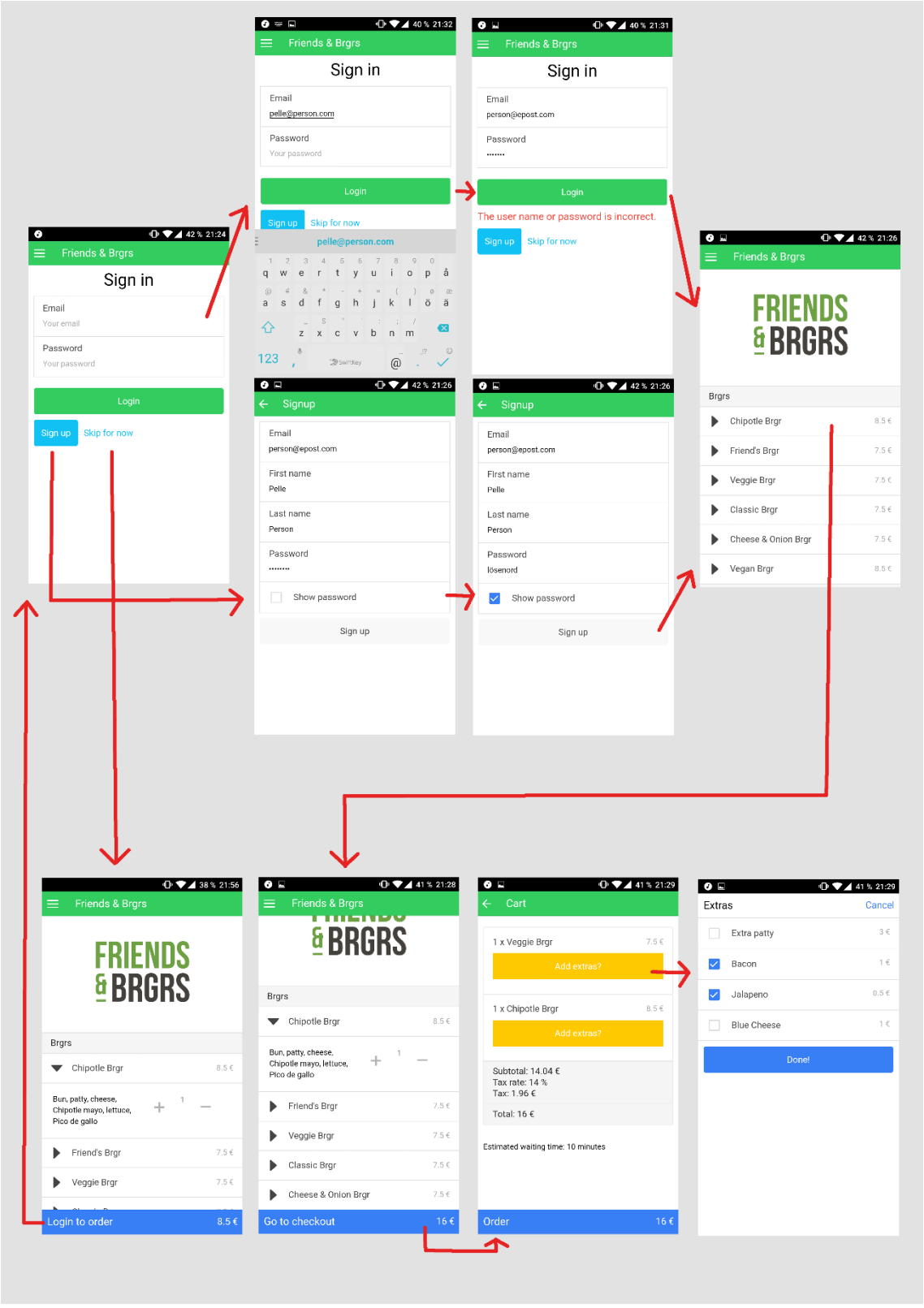
Stack Exchange Inc. 2016. *Stackoverflow*. Tillgänglig: http://stackoverflow.com/ Hämtad 25.4.2016.

Whinnery, Kevin. 2010. *Titanium Guides Project: JS Environment*. Tillgänglig: http://www.appcelerator.com/blog/2010/12/titanium-guides-project-js-environment/ Hämtad 17.3.2016.

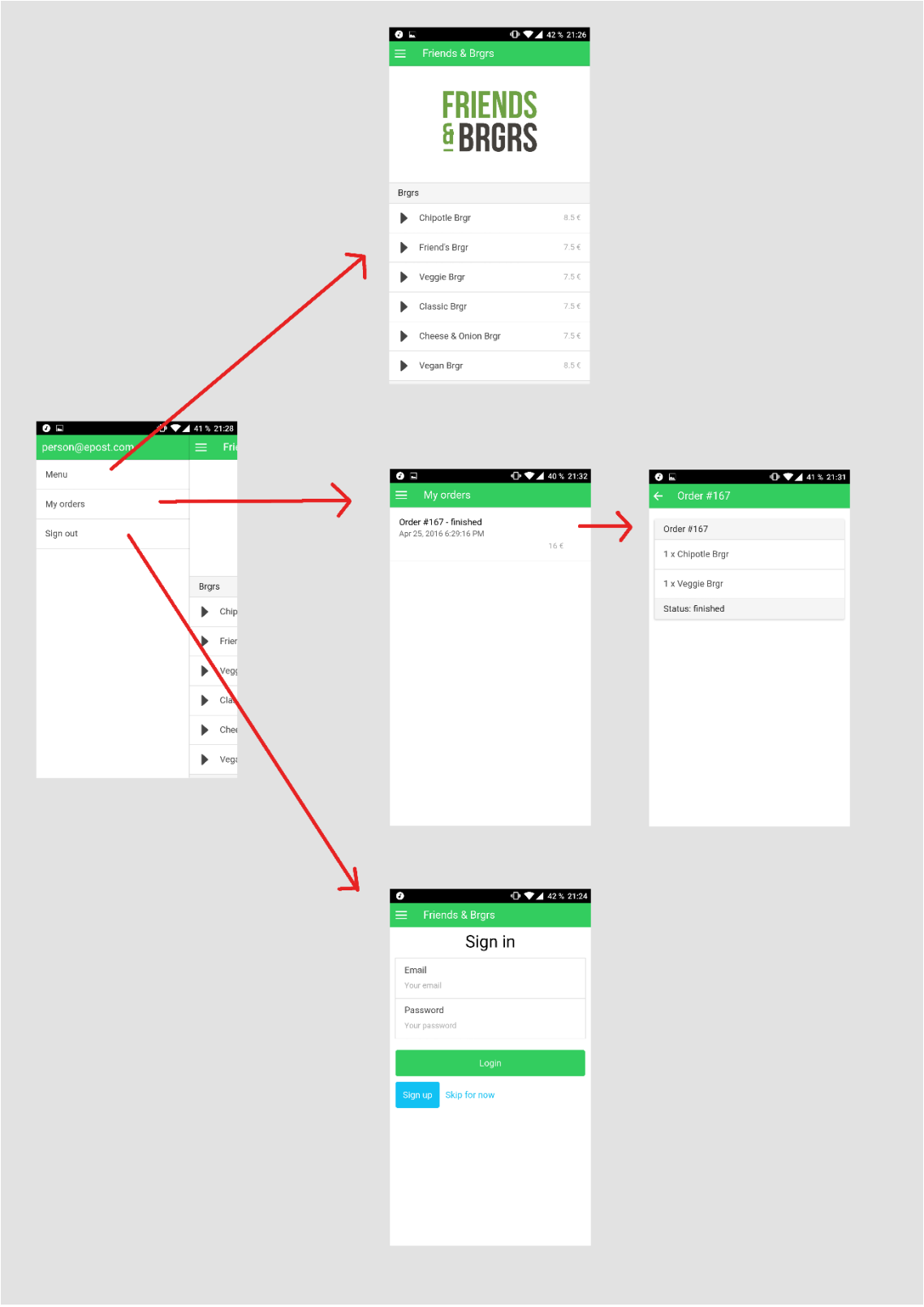
BILAGOR

Bilaga 1: Flödesschema för registrering, inloggning och beställning i den färdiga appen

Bilaga 2: Flödesschema för alla menyval i den färdiga appen



Bilaga 1: Flödesschema för registrering, inloggning och beställning i den färdiga appen



Bilaga 2: Flödesschema för alla menyval i den färdiga appen