Web and Cross-Platform Mobile application sharing same code base using modern web technologies

Ishwor Thapaliya

Master’s thesis
November 2016
Degree Programme In Information Technology
Software Engineering
ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
Software Engineering

THAPALIYA, ISHWOR:
Web and Cross-Platform Mobile application sharing same code base using modern web technologies

Master's thesis 55 pages, appendices 5 pages
November 2016

With the raise in amount of technologies with which we can build web applications and mobile applications, it is usually a tough task to decide which technology is the best for the project to save time and effort and increase velocity of the project in hand.

Web technology has seen a huge amount of transformation in the recent years and with the open source projects like Apache Cordova, it is now possible to build Cross Platform mobile applications using the web technologies. With the server-side platform Node.js, it is now possible to build server side application in JavaScript.

This piece discusses and demonstrates the use of a JavaScript in both server-side and client-side application, how the client-side JavaScript code can be shared between both web and mobile applications, and different tools and frameworks which will help in the process of making the whole application including the topics like integration and end-to-end testing in a short amount of time.

Key words: web, mobile, integration, e2e, client, server, shared code
CONTENTS

1 INTRODUCTION ................................................................................................................. 6
2 REST ARCHITECTURE ........................................................................................................ 8
  2.1 Client-server .................................................................................................................. 9
  2.2 Stateless ....................................................................................................................... 9
  2.3 Cacheable ..................................................................................................................... 10
  2.4 Layered System ........................................................................................................... 10
  2.5 Code on demand .......................................................................................................... 11
  2.6 Uniform interface ....................................................................................................... 11
    2.6.1 Identification of resource ...................................................................................... 11
    2.6.2 Manipulation of resources through representations ............................................. 13
    2.6.3 Self-descriptive messages .................................................................................... 13
    2.6.4 Hypermedia as the engine of application state (HATEOAS) ............................. 14
3 BRIEF OVERVIEW OF WEB TECHNOLOGIES ................................................................. 15
  3.1 HTML ........................................................................................................................ 15
  3.2 CSS ........................................................................................................................ 16
  3.3 JavaScript .................................................................................................................. 16
4 THE DEVELOPMENT STACK .............................................................................................. 17
  4.1 AngularJS .................................................................................................................. 17
  4.2 Node.js ...................................................................................................................... 19
    4.2.1 Sails.js ................................................................................................................ 20
  4.3 MongoDB .................................................................................................................. 22
  4.4 Apache Cordova ......................................................................................................... 23
    4.4.1 Ionic Framework ................................................................................................. 23
  4.5 Yeoman ..................................................................................................................... 25
  4.6 Gulp .......................................................................................................................... 27
  4.7 Bower ......................................................................................................................... 30
  4.8 Bootstrap .................................................................................................................... 32
5 THE DEVELOPMENT WORK .............................................................................................. 33
  5.1 Objective .................................................................................................................... 33
  5.2 Creating custom development workflow using Yeoman, Gulp and Bower ..... 34
  5.3 Property Management App in brief .......................................................................... 36
  5.4 Server-side application – creating REST APIs ......................................................... 37
    5.4.1 Models ................................................................................................................. 37
    5.4.2 Blueprint API and Controllers .......................................................................... 38
    5.4.3 Routes .................................................................................................................. 38
    5.4.4 Development and Production environment configurations .............................. 39
5.4.5 Policies ................................................................................................................. 39
5.4.6 Custom Responses ............................................................................................... 40
5.4.7 Testing APIs manually using Postman ............................................................... 41
5.4.8 Automated integration testing using Mocha test framework ......................... 42
5.5 Client-side application – creating the User Interface ....................................... 43
  5.5.1 The JavaScript part (AngularJS) ........................................................................ 44
  5.5.2 Views and Styles for the Web Application ....................................................... 47
  5.5.3 Views and Styles for the Mobile Application .................................................... 47
  5.5.4 Running Web Application in development environment ............................... 47
  5.5.5 Emulating/Running Cross platform Mobile Applications ............................. 48
  5.5.6 End-to-end testing ............................................................................................. 50
6 DEPLOYMENT ............................................................................................................... 52
7 DISCUSSION ................................................................................................................ 55
REFERENCES .................................................................................................................. 56
APPENDICES .................................................................................................................. 58
  Appendix 1. Different views of the web application ................................................. 58
  Appendix 2. Different views of the mobile application ............................................. 61
ABBREVIATIONS AND TERMS

REST  Representational State Transfer
JSON  JavaScript Object Notation
RDMBS Relational Database Management System
NoSQL Not Only SQL
npm  Node Package Manager
e2e testing End-to-end testing
API  Application Programming Interface
HTML Hypertext Markup Language
JS  JavaScript
UI  User Interface
ORM  Object-relational mapper
CRUD Create Read Update Delete
MVC  Model View Controller
HTTP Hypertext Transfer Protocol
WWW  World Wide Web
MIME Multipurpose Internet Mail Extensions
SPA Single Page Application
1 INTRODUCTION

With the vast amount of platforms, technologies, programming languages, frameworks and tools that are available, sometimes it is a very daunting task for a developer to choose between them or it might be so that a developer is already familiar with a programming language, framework or set of tools that he/she is comfortable with and end up choosing them even though it might not be the optimum language, tool or framework for the project on hand.

With the transformations seen in the area of web technologies and the raise of platform like Node.js and frameworks like Apache Cordova and AngularJS, we will dive into the world of JavaScript and try to build an application using JavaScript in both back-end and the front-end. We will also look into on how is it possible to share the same front-end JavaScript code in between web and mobile application.

For the mobile application development part, since we are going to share the same code base in between web and mobile application, hybrid cross platform mobile application development framework called Ionic (which uses the Apache Cordova behind the scene) will be used. The other reason behind choosing hybrid framework is because the application is going to be light in terms of performance requirement and we would also like to save time using the hybrid cross platform framework.

In terms of architecture, a decoupled architecture will be used meaning the back-end, front-end (mobile and web) are independent of each other because of the nature of the application we are developing being “Write once, publish everywhere” type. These independent applications then talk to each other using REST (Representational State Transfer ) APIs (Application Programming Interfaces).

We will also make use of widely used tools like yeoman, bower, gulp, and npm in order to make our development process faster. We will also use test frameworks like Mocha and Protractor to automate the application testing.

The development part will focus on developing an application, which will be used to manage properties and will be used by property managers. The main idea behind the
application is that the property managers, who are in the different property sites throughout the day, will use the mobile application version through which they can add different tasks like inspecting renovation jobs, picture of the renovation processes while they are on the renovation site. The web application is used in the office where office secretary can inspect the tasks that are registered and can bill the tasks.

Also using the web application, the people with different roles can follow what is happening in the properties that is managed by their company, assign price to the different tasks performed in the site, add/remove new condominiums, create users with different roles and permissions etc.

In summary, during the development work, we will be experimenting with the development stack and frameworks chosen and utilize them to use a web application and hybrid mobile application which will be sharing same code base.
2 REST ARCHITECTURE

REST stands for Representational State Transfer, which is an architecture style for designing networked applications. The term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. (Mike Amundsen 2008).

REST is a simple way to organize interactions between independent systems (Ludovico Fischer, 2013). It has been increasingly popular since 2005 because of its simplicity, minimal overhead and ability to interact with clients like mobile phones and other website compared to alternative mechanisms like SOAP (Simple Object Access Protocol), RPC (Remote Procedure Calls) etc.

REST is not only used in web but since it is inspired by HTTP (Hypertext Transfer Protocol), almost always HTTP protocol is used. The REST architecture consists of clients, servers, resources and a vocabulary of HTTP operations known as request methods.

PICTURE 1: REST and some terminologies surrounding it
For an application to be “RESTful”, it must follow the architectural constraints that are described below.


2.1 Client-server

Client and Server are separated. The clients are not concerned with the data storage but request it from the server. Server is not concerned with the user interface or user state. This separation of concerns makes both client and server scalable and simpler.

PICTURE 2: Client and Server separated (Separation of concerns)

2.2 Stateless

REST is stateless, meaning that the server does not stores any state about the client between the requests. Each request from any client contains all the information necessary to service the request and session state is held in the client. Since, the server does not store the state information, the server uses less resource. Even though the sessions are not stored in the server, it can be stored on database upon requirement.
2.3 Cacheable

Cache mechanism plays vital role on REST architecture style. WWW (World Wide Web) clients can cache responses that are received from the server so in order to avoid the reuse of old or inappropriate data; responses must define themselves implicitly or explicitly cacheable or non-cacheable.

2.4 Layered System

Client does not know if it is connecting directly with the end server or just intermediary server. Servers can be in a hierarchical order where each layer sees only the layer, which is connected to it. Intermediary servers can improve scalability by enabling load balancing and providing shared caches.
2.5 Code on demand

Server can send codes to the clients to temporarily extend the functionality of client. Typical example of this can be sending of JavaScript code to the client. Code on demand is the only optional architectural constraint of REST architecture.

2.6 Uniform interface

This is the most fundamental REST architectural constraint. The uniform interface simplifies and decouples the architecture. The uniform interface has four constrains and they are explained below.

2.6.1 Identification of resource

In REST, client interact with resource through a fix set of verbs i.e. GET (read), POST (create), PUT and DELETE using URI (Uniform Resource Identifier).
PICTURE 6: Interacting with resource using HTTP verbs

URI's should follow a predictable and hierarchical structure to enhance understandability. For example, if there is a resource of type “article”, the URI might look something like below:

Create an article using POST verb:
http://www.domain.com/articles

Reading a particular article (with unique ID ex. 123) using GET verb:
http://www.domain.com/articles/123

Updating a particular article (with unique ID ex. 123) using PUT verb:
http://www.domain.com/articles/123

Deleting a particular article (with unique ID ex. 123) using DELETE verb:
http://www.domain.com/articles/123

From the above example we can clearly see that the URI's for Reading, Updating and Deleting for the resources are exactly the same and the HTTP request determines what kind of operation to perform with the resource in question.
2.6.2 Manipulation of resources through representations

HTTP standards are used to describe the communication. Example: with a GET request and URI client can retrieve a representation of resource from the server and that representation of resource contains all the/enough information needed for the resource in question to be modified or deleted.

2.6.3 Self-descriptive messages

Each message includes enough information to describe how to process the message. Internet media types (previously known as MIME (Multipurpose Internet Mail Extensions) types) are used to make the messages self-descriptive.

Example:
Content-Type: application/xml
Content Type: application/json
2.6.4 Hypermedia as the engine of application state (HATEOAS)

In REST, there is a single endpoint for a resource. When client gets the representation of a resource, all the other actions that are available are described in the representation of the resource obtained. Client does not assume that any other actions are available beyond what has been received from the server previously.
3 BRIEF OVERVIEW OF WEB TECHNOLOGIES

World Wide Web consists of vast amount of resources, which are identified by URI’s and interlinked by hypertext links. These resources are accessed via the Internet using web browsers like Internet Explorer, Mozilla Firefox, Google Chrome or Apple Safari to name a few. These web resources are mainly text documents formatted with Hypertext Markup Language (HTML), styled with Cascading Style Sheet (CSS) and programmed with JavaScript to enhance the user experience. Besides HTML, CSS and JavaScript which are often referred to as Front-End technologies, there are Back-End technologies also involved in the process which can consist of programming languages like PHP, Java, Ruby, Python and databases like MySQL, MSSQL, PostgresSQL, Oracle to name a few.

![Diagram of web technologies]

PICTURE 9: General overview of web technologies that are commonly used

With the introduction of new technologies like Node.js and application frameworks like AngularJS, web development is leaning more towards the concept of Single Page Applications (SPA) where the contents of the application are dynamically updated as the user interacts with the app without having to reload the entire HTML page and real-time web applications where the contents are pushed to the clients and the client side technology which is aware of the new content then updates the content and displays it without any kind of user interaction at all.

3.1 HTML

HTML (HyperText Markup Language) is a markup language, which provides structure to the websites and web applications so that the browsers can read them and display to
the end user in proper format. HTML5, the fifth version of HTML standard was finalized on 28\textsuperscript{th} October 2014 aimed to improve the language with the support for latest multimedia, new element / input element types, Local storage to name a few. (W3C, [www.w3.org](http://www.w3.org), Read 2016)

### 3.2 CSS

CSS (Cascading Style Sheet) is a language, which is used for describing the look of the HTML documents. The main purpose of CSS is to be able to separate the document content from document presentation i.e. to separate the style definition from the HTML documents. (W3C, [www.w3.org](http://www.w3.org), Read 2016)

### 3.3 JavaScript

JavaScript is the programming language of HTML and the Web ([http://www.w3schools.com](http://www.w3schools.com), Read 2016) and is mainly used in web pages and is supported by all the modern browsers. Earlier, JavaScript was considered just as a part of front end technology and was used mainly to add interactivity to the web pages but since the introduction of Node.js in 2009, JavaScript is raising as a good option for the server side language specially for the front-end developers who are already familiar with the JavaScript and want to jump to the server side programming and/or full-stack programming.

![JavaScript in the server side using Node.js](image)

Picture 10: JavaScript in the server side using Node.js
4 THE DEVELOPMENT STACK

Since the development work is mainly focused on building web and mobile applications sharing same code base, after trying number of frameworks and tools, the decision was made to go with the following tools and technologies described below based on the facts like popularity, community and contributor activities (open source projects), learning curve and time and effort that is required to build a fully functional web and mobile application.

4.1 AngularJS

AngularJS is a structural framework for dynamic web apps (http://docs.angularjs.org, read 2016). It is a popular JavaScript UI framework widely used to build dynamic single page applications and is being maintained by Google and community of developers. There are many advantages of using AngularJS like being able to structure the client side codes systematically using MVC (Model View Controller) design pattern, Data binding, dependency injection, HTML extension via directives etc.

![Diagram of MVC pattern and data binding in AngularJS](Picture 11)

One of main advantage of using AngularJS is being able to organize the codes in smaller modules. This enables developer to maintain the code easily and reuse the same module in multiple applications by injecting the modules in an application where it is
needed. An example of this can be seen in picture 12 below where a module named ‘restApp.articleData’ is being created. This module is then injected to the angular application named restApp. After injecting the ‘restApp.articleData’ to the ‘restApp’, we can now access the service provided by the ‘restApp.articleData’ in the angular controller named ‘articleDetailCtrl’.

Picture 12: An example on creating module and dependency injection in AngularJS
4.2 Node.js

Node.js is a runtime that lets JavaScript to run in a computers or servers rather than only in browsers which enables developers to do various tasks like accessing file system, listening http request, sending http response, access database etc. using JavaScript. Node.js is gaining popularity because of its active community, lots of reusable packages, which can be downloaded easily using node package manager (commonly known as npm). Node.js makes it easy to build server side applications and utility applications, which help in daily development tasks (ex. Gulp.js, Yeoman).

Node.js is a JavaScript runtime built on Chrome’s vV8 JavaScript engine. Node.js uses an event driven, non-blocking I/O model that makes it lightweight and efficient (https://nodejs.org, 2015).

In the picture 13 below, a node.js application is being created using the node modules like express, node-restful, body-parser and method-override. The application will listen to the environment variable Port if available or 3000 port when started.

PICTURE 13: Node.js app starting server at port 3000 and creating RESTful API for resource ‘articles’
Picture 14 below shows the process of starting the node application created above in the picture 13, which starts the node server in port number 3000 using terminal window.

![Starting the server using terminal in Mac OS X](Picture 14)

Picture 15 below illustrates the process of interaction with the node server using postman tool without us having to build user interface for testing.

![Interaction with resources using Postman tool](Picture 15)

4.2.1 **Sails.js**

Sails.js is a framework that is built on top of Express (a minimal and flexible Node.js web application framework), which is designed to emulate the MVC (Model View Controller) pattern like Ruby on Rails, Laravel etc. Sails makes it easy to build custom, enterprise-grade Node.js apps ([www.sailsjs.com](http://www.sailsjs.com) 2015). Sails provides a lot of out of the box features like automatically serve basic CRUD (Create Read Update Delete) JSON (JavaScript Object Notation) API’s (Application Programming Interface) without hav-
ing to write a single line of code using something called blueprints, built-in integration with the cross browser websocket for real-time applications and database agnostic ORM. Sails.js is also designed to be compatible with any front-end technology like AngularJS, Backbone.js, and Apache Cordova etc.

In the picture 16 below, the first part shows the process of installing sails.js globally using the command “npm install –g sails”. In the second part, a new application called “thesisdemo” is created using the command “sails new thesisdemo”. After the app has been created, we go inside the “thesisdemo” folder and create an API named “thesis” using the command “sails generate api thesis”. Once the API generation was successful, the sails server was started using the command “sails lift”. The sails framework starts the server in the default port 1337. Manual testing of the API was done using the postman tool as seen in the last part of the picture.

Picture 16: Installing Sails.js, creating new application and blueprint API and interacting with API using Postman tool
4.3 MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling (https://docs.mongodb.com/manual/introduction, Read 2016). MongoDB is a cross-platform document oriented database. Documents are the basic unit of data in MongoDB and are comprised of key-value pairs. A group of MongoDB documents are called collection, which is equivalent to a table in relational database management systems (RDBMS). Unlike traditional RDBMS, MongoDB supports dynamic schema design allowing the documents to have different fields and structures. MongoDB is classified as NoSQL (Not only SQL) database, which simply means that they are an alternative to SQL database but they can apply SQL like concepts also.

Once the MongoDB has been installed in the system, it can be started using the command “mongo”, the databases can be listed using the command “show dbs”. A particular database can be selected for use using the command “use database_name”. These processes are illustrated in the picture 17 below.

Picture 17: Example of selecting database and finding documents in MongoDB
4.4 Apache Cordova

Apache Cordova is a set of device APIs that allow a mobile app developer to access native device functions such as camera, network information, geo location etc. from JavaScript applications which allows developers to develop mobile applications using the web technologies like HTML, CSS and JavaScript also known as hybrid mobile app. Since the device APIs is consistent across multiple device platforms, the apps are portable to other device with almost no change.

4.4.1 Ionic Framework

Ionic is an HTML5 mobile app development framework targeted at building hybrid mobile apps ([http://ionicframework.com/docs/guide/preface.html](http://ionicframework.com/docs/guide/preface.html), read 2016). It is an open-source SDK for hybrid mobile app built on top of AngularJS and Apache Cordova. Since we are already using AngularJS for the front-end development, using hybrid mobile app framework that also uses AngularJS seemed a good reasonable choice for the development of hybrid mobile app.

Picture 18 below shows the process of installing ionic framework using command “npm install –g ionic” and creating a project using ionic using the command “ionic start projectname”.

PICTURE 18: Installing ionic and creating a new project
Picture 19 shows the process of adding android platform to the ionic project using the command “ionic platform android”

![Picture 19: Adding android platform to the project](image)

Android project can be emulated in an android emulator using command “ionic emulate android” as shown in the picture 20. The project is opened in an android emulator as shown in the picture 21.

![Picture 20: Emulating the demo project in android device](image)

![Picture 21: Demo running in an android emulator](image)
4.5 Yeoman

Yeoman is a workflow tool that provides a set of development tools. Yeoman helps you to kickstart new projects, prescribing best practices and tools to help you stay productive (http://yeoman.io/, Read 2016). With the use of Yeoman generators, one can quickly create a development environment with the desired file and folder structure along with the build configuration using Gulp or Grunt and package management using Bower and/or npm just by using a command line tool. There are thousands of yeoman generators available and one can install the generators using node package manager (npm). If there is no generator that fits own workflow, one can always roll out own generator according to their requirement.

To install yeoman, one must install its dependencies, which are Bower, Grunt and Gulp. Yeoman, Bower, Grunt and Gulp are installed globally using the command “npm install –g yo bower grunt-cli gulp” as illustrated in the picture 22 below. Picture 23 shows the process of installing angular generator using the command “npm install –g generator-angular”.

PICTURE 22: Installing yeoman and its dependencies

PICTURE 23: Installing angular generator

Once the angular generator has been installed, an angular application can be created using the command “yo angular application_name” as shown in the picture 24. This process will generate the angular application with the files and folders as illustrated in picture 25. The app folder is where all the development code will reside, app/scripts will have AngularJS codes, app/views will have all the HTML templates and app/styles will have all the CSS files.
As shown in the picture 26, running command ‘grunt’ will generate the dist folder where all the JavaScript library files are minified, concatenated/bundled and put in vendor.xxxx.js file, all the project codes are minified, concatenated and put in script.xxxx.js
file. Similarly the style folder consists of main.xxxx.css and vendor.xxxx.css and all the HTML files are also minified. The contents of the dist folder are the materials, which will be deployed to the server when the project is finished.

**PICTURE 26: Generating distribution (dist) folder and a look at the ‘dist’ folder**

Since no any readymade yeoman generator, which would be suitable for the development work for this thesis project, a generator that suits our development workflow will be created from scratch.

### 4.6 Gulp

Gulp is task runner, which automates the tedious and repetitive tasks during the development such as watching the development files and when a file changes, run some task (ex. reloading the browser which lets the developer see the changes immediately without having to go and press the reload icon by themselves), bundling and minifying libraries and stylesheets etc. The main purpose of using build system like Gulp is to reduce the repetitive tasks during development and create more productivity ([http://brandonclapp.com/what-is-gulp-js-and-why-use-it/](http://brandonclapp.com/what-is-gulp-js-and-why-use-it/), Blog, Read 2016).
In order to use Gulp, it must be installed in the system using node package manager (npm) using the command “npm install –global gulp” as shown in picture 27. After installing gulp, one can create ‘gulpfile.js’ file in the project and start creating tasks that is required for the development.

![Picture 27: Installing gulp](image)

The command “npm init” shown in picture 28 will initialize a project by asking some questions and creating “package.json” file in the root of the project. Finally we can add gulp as our development dependency for our project as shown in picture 29 which will add the gulp in “node_modules” folder and add gulp under “devDependencies” object in package.json file as shown in picture 30.

![Picture 28: Creating ‘gulp-demo’ directory and creating a ‘package.json’ file using npm init.](image)

![Picture 29: Adding gulp as our development dependency](image)

![Picture 30: ‘package.json’ file after adding devDependency](image)
After adding the dependency, we can now finally create “gulpfile.js” file in our project root and start to add the tasks that we would like to automate. Explained below is a simple gulp task on how to serve an index.html in a development environment. This process is illustrated in picture 31, which also shows the process of installing gulp modules like “gulp-connect” and “opn”.

PICTURE 31: creating a “gulpfile.js” and installing gulp dependencies like ‘gulp-connect’ and ‘opn’

In picture 32, we are utilizing gulp, gulp-connect and opn to create a gulp task named “start” which simply starts the application on port 9000. The task “start” is then assigned as default gulp task so that it starts with the command “gulp”.

PICTURE 32: Creating tasks in “gulpfile.js” to start server on port 9000 and serve index.html file from app folder.
We can see in the picture 33 that the command gulp starts the “start” task defined in the gulpfile.js file because “gulp” command starts the “default” task where the ‘start’ is also defined. The default task can be bundled with many other tasks along with the “start” task upon requirement. Similarly we also could start the server using “gulp start” which would just start the server only. Picture 34 shows the result of the command “gulp” which starts the app in browser in port 9000.

PICTURE 34: The result of “gulp” command where index.html from app directory is served on localhost:9000

4.7 Bower

Bower is used for managing packages like frameworks, libraries for the front-end or client side development. Bower is very popular package management system with more than 30 thousand packages as of 15.09.2015.

Bower works by fetching and installing packages from all over, taking care of hunting, finding, downloading and saving the stuff you are looking for. Bower keeps track of these files in manifest file, bower.json (http://bower.io, 15.09.2015).

As described in bower.io, the main advantage of using bower is being able to download required libraries using command line and not having to hunt them over the internet and
download the files separately and move them to the development environment which saves a lot of developers time.

Bower can be installed globally using the command “npm install –g bower”. To create a bower.json file in the project, we can use command “bower init” once it has been installed as shown in the picture 35 and 36 respectively. Once the bower has been initialized, we can now install dependencies to our project using command “bower install” as shown in the picture 37 and 38.

PICTURE 35: Installing bower in local machine

PICTURE 36: Initializing bower, which will create bower.json file

PICTURE 37: Installing bootstrap (bower will install jQuery automatically as it is a dependency for Bootstrap)

PICTURE 38: Structure of “bower.json” file after initialization and installing Bootstrap
4.8 Bootstrap

Bootstrap is the most popular HTML, CSS and JS framework for developing responsive, mobile first projects on the web (http://getbootstrap.com, 15.09.2015).

Bootstrap is very good tool when it comes to faster and easier web development as it includes many reusable components like buttons, tables, navigations modals etc. It is responsive meaning uses grid systems for creating page layouts through series of rows and columns. It’s ease of use another advantage as anyone with basic HTML and CSS knowledge can start using it.
5 \hspace{1em} \textbf{THE DEVELOPMENT WORK}

5.1 \hspace{1em} \textbf{Objective}

The main objective of the development work part of thesis is to be able to leverage all the technologies discussed in development stack part (4.1 to 4.8) and use them to create a web and mobile application which will be used to manage properties and will be used by property managers. The main idea behind the application is that the property managers, who are in the different property sites throughout the day, will use the mobile application version through which they can add different tasks like inspecting renovation jobs, picture of the renovation processes while they are on the renovation site. The web application is used in the office where office secretary can inspect the tasks that are registered and can bill the tasks. It also can be used to keep track of the projects that are active and so on.

The objective with the front-end is to share the AngularJS code between web and mobile application. In order for this to be done, a custom yeoman generator is needed where we can have our own development workflow.

The back-end will use Sails.js and MongoDB and provide a simple REST API, which is used by the front-end and mobile apps to interact with the data residing in the database.

The application will implement the concept of decoupled architecture where the back-end and front-end are completely detached and unaware with each other. They will communicate with each other using REST API as mentioned in the previous paragraph.

By the end of the development work we should be able to

- Utilize the modern tools and technologies discussed in 5.1 to 5.8
- To be able to share / use presentation logic codes for multiple devices and platforms
- To be able to build sellable products targeted to different devices and platforms without needing to have an army of developers for different devices and platforms
- To be able to develop end-to-end application rapidly
5.2 Creating custom development workflow using Yeoman, Gulp and Bower

Since the workflow required for the front-end is unique, development of custom generator was required so that it fits our development workflow. The generator then could be used to scaffold/generating starting point for the similar kinds of projects later on. To create a custom generator, yeoman’s “generator-generator” was used as a starting point. The files and folder structure needed in desktop and mobile app is similar when deployed but during the development phases both mobile and web application will have own styles and views folders. Similarly, the entry point to the application i.e. index.html file will be separate during development. JavaScript files, which are shared in the both applications, will be in a shared folder.

Keeping the requirements mentioned above in mind, a generator named “generator-webmobileapp” was created and separate index files for web and mobile app was added. Also the dependencies, which are needed during development and deployment, were added in “_bower.json” and “_package.json”. The automated tasks like watching the changes in scripts, minifying CSS and JavaScript files, creating separate distribution folders for web and mobile applications were coded in the “_gulpfile.js” file. The folder structure of the “webmobileapp” generators looks like in the picture 39 below.

![Diagram of folder structure of webmobileapp generator](picture39.png)
After the generator was complete, we could scaffold our project using the command ‘yo webmobileapp’. After giving the necessary information like project name, version, appid, author and so on, the generator creates a starting point for our application with the folder structure as described in the paragraph above, front-end dependencies and libraries we need. This is illustrated in the picture 40 below.

After manually testing the custom generator created, the next step was to create the actual application i.e. Property Management App.
5.3 Property Management App in brief

The application that will be developed will be targeted for the property managers (Tekninen Isännöitsijä in Finnish). Using the mobile version of the application, property managers can add different tasks that are carried throughout the day using handheld devices and the clients are billed according to the tasks that are added under the projects. The tasks can be for example related to renovation inspection, waterproofing system inspection, water damage inspection and renovation etc. The administration / office in other hand can use the web version of the application to perform tasks like creating invoice, adding tasks and price for the tasks etc.

In the context diagram (picture 40), we can see the different actions property manager and office employees can perform using the application example. Property managers can add, edit, update project tasks, property maintainer information, change own password, mark the projects as completed and so on. Whereas office employee can add, edit, update, delete task and prices, users, projects, reset password, generate invoices for the completed projects etc.

PICTURE 41: Context diagram of the Property Management App
Since the application being built is going to have decoupled architecture, development of server side application with REST API's was developed.

5.4 Server-side application – creating REST APIs

A separate server side application was created using Sails.js framework which would act as our REST interface for the client side web and mobile applications. It mainly consists of the following components.

5.4.1 Models

First of all, seven different models were created according to requirement of our application. Each model consists of set of properties that represents a single collection in document-based database. In the picture 42 below, we can see the data model named “Project” which has the attributes “condominium”, “property_id”, “property_manager”, “description”, “project_phase”, “images” and “project_task”.

![List of models created and project model in detail](PICTURE 42)
5.4.2 Blueprint API and Controllers

Since sails provides basic CRUD functionality out of the box aiming to reduce the amount of code required to get an app up and running through what it calls blueprint API. Because of this API, only the custom business logic, which would deviate from basic CRUD functionality, were coded in the controller. We can see an example of a controller and its method in the picture 43; we can see a method named “downloadByFileName” which is a part of “ImageUploadController”. This method is tied to a route and when the request is made with the “projectId” and “file” (file name), it will create a read stream of the file if there were no errors and if the user has permission to download the file.

![Picture showing list of controllers and method to get project image from the server](image)

PICTURE 43: Picture showing list of controllers and method to get project image from the server

5.4.3 Routes

Even though sails provides blueprint routes out of the box for the basic CRUD functionality but need for some custom routes were seen and created. Custom routes in Sails framework can be created in “config/routes.js” file and the pattern is as follows:

“HTTP VERB route” : “controller.method”

example: “GET /posts” : “PostController.Index”
5.4.4 Development and Production environment configurations

Database configurations and port configurations were configured for development and production environment separately. Database URL, database credentials, and application port for the production environment were set using production environment’s environment variable whereas it was hardcoded for the development environment. The difference in environment configuration can be seen in the picture 44 below.

5.4.5 Policies

Policies is a way to implement access control and authorization in Sails framework. Custom policies can be created according to the requirement of project under “api/policies” folder and they can be implemented in the “config/policies.js” file. An example of a custom policy would be when a certain user with a certain role is permitted to invoke a
certain action/method in a controller but others are not allowed to do so. Picture 45 and 46 will illustrate such a scenario, which was implemented in this project.

In Picture 45, a custom policy named “isSuperAdmin” is created where the role of the logged in user is checked. The same policy is later implemented in ‘config/policies.js’ so that only users with “SuperAdmin” role are permitted to invoke ‘create’ and “delete-ByEmailAddress” methods of “UserController”, which can be seen in Picture 46.

5.4.6 Custom Responses

Custom server responses can be created in sails under api/responses folder. These responses could be reused when required on any part of the code later on. Sails come with
common responses that are widely used in most of the server side projects but they are not always sufficient. One of the custom responses, which were created during this project, is illustrated in the picture 47 which can be used in case we have to return a response, which sends the HTTP status code 409 with the custom text ‘Email address is already taken by another user’. This will save us from writing the response code and the response text every time we have to return this type of response and instead we can simply use “res.emailAddressInUse()” in our code.

Picture 47: An example of custom response

5.4.7 Testing APIs manually using Postman

One of the critical phases of the project is to test the codes that are being written. To test the APIs that are being coded were working, a tool called Postman was used. Postman is a REST client through which REST API’s can be called and responses can be seen. A simple example of usages of Postman in this project is shown in the pictures 48 below where we are sending get request to the address “http://localhost:1337/projects” and in return we are getting the list of projects.
5.4.8 Automated integration testing using Mocha test framework

Manual tests are a good way to see if the code is working during development and debugging but automated tests are crucial to any project so that the testing process is faster, reliable with reduced human error. Taking these factors into account, automated test using Mocha testing framework was written to test the back-end project.

Since Mocha is a test framework and it is not needed in the production environment, it was installed as development dependency. Along with Mocha, other node modules like supertest and superagent were also included as development dependencies. The commands used to install the modules are as follows:

```
npm install mocha -save-dev
```
```
npm install supertest -save-dev
```
```
npm install superagent -save-dev
```

After the required dependencies were added, a folder named ‘test’ was created where all our tests would reside. The main goal was to automate the CRUD and authentication functionality fast and reliably. In picture 49, we can see an example where a test is being done for the scenario where a user should not be able to log into the system with wrong credentials. In the process, a CSRF (Cross Site Request Forgery) token is being requested from an endpoint “/csrfToken” and then user credentials is being sent to the endpoint “/login” including the CSRF token received. At last a response code of 401 is
being expected and if the response code received by the server is 401, the test will successfully pass.

5.5 Client-side application – creating the User Interface

Since our application is decoupled in architecture as stated earlier, meaning the back-end application and front-end application are independent of each other and communicate via REST APIs, a new project for the front-end application was created using the yeoman custom generator which was built to scaffold projects of this nature (discussed in 5.2). We can see the process in picture 50, where a command “yo webmobileapp app_name” is run and this command then scaffolds a front-end application.
After scaffolding is complete, we have a starting point for our project. Most of the changes are made in app folder. “index-mobile.html” and ‘index.html’ are the starting point of front-end applications for the mobile and web applications respectively. Similarly, views and style sheets for mobile are put under “mobile_views” and “mobile_style” folders and “views” and “styles” folder are for the web application. All the JavaScript codes will go under the “shared/scripts” folder. The main reason for separating styles and views for the mobile and web application is because of the look and feel for the mobile and web applications will be different.

5.5.1 The JavaScript part (AngularJS)

One of the very important parts of the front-end application is client side scripting part. These scripts are put inside “shared/scripts” folder in the application since both mobile and web application will be sharing the same scripts. The chosen framework for this job i.e. AngularJS has already been discussed in the theory part. The client side scripts written in AngularJs mainly handle fetching data and front-end application logic in our case.

For the interaction with back-end rest APIs, “angular-resource” module was used because of the simplicity it provided for performing the operations like GET, POST, PUT, DELETE etc. AngularJS factory service were created by injecting the “$resource” service provided by “angular-resource”. The methods provided by these factory services were then called from controllers to fetch and manipulate the data. An example of such service can be seen in picture 51 below where factory methods are created under “propertyManagerApp” called “CondominiumsApi” and “CondominiumApi”. These factory methods consist of different methods to handle CRUD functionality of the condominium’s resource.
Controllers are responsible for passing the data to view and vice versa. In another word it acts as a mediator in between services and views and control the flow of data in the application. In picture 52, is the screenshot of “CondominiumListCtrl”, which calls “condominiumsApi.query()” method and the data fetched is then assigned to “$scope.condominiums” which consists of the list of condominiums. This data can be then used in the view and presented to the end user using a view i.e. ‘condominium-list.html’ in this case. The same controller consists of two more functions named “viewCondominium” and “goToCreateCondominiumForm”. The “viewCondominium” function is responsible for opening a detailed view of a condominium when a user clicks on some button and likewise “goToCreateCondominiumForm” is responsible for opening a view to create new condominium when a user clicks a button.
The information about which controller belongs to which view is configured in routes or states (if using ui-router module). In picture 53, we can see that the state “condominium” is bound to url “/condominium” and is assigned “condominium-list.html” as the template and “CondominiumListCtrl” as the controller.

```
angular.module('propertyManagerApp')
  .controller('CondominiumListCtrl', function ($scope, condominiumsApi, $state, toastr) {
    condominiumsApi.query().$promise.then(function (data) {
      $scope.condominiums = data;
      //console.log(data);
      function (err){
        toastr.error('Error: Could not retrieve data');
      };
    $scope.onViewCondominium = function (condominiumId) {
      // $location.path('/condominium/view/' + condominiumId);
      $state.go('condominium.view', {id : condominiumId});
    };
    $scope.goToCreateCondominiumForm = function () {
      $state.go('condominium.create');
    };
  })
  })
```

Picture 52: Picture showing ‘CondominiumListCtrl’

Picture 53: Picture showing different states
5.5.2 Views and Styles for the Web Application

Since the look and feel of web application is different than that of mobile application, it was seen necessary that the styles and views for the web application and mobile application are separated. The views of the web application during the development process are added under the “views” and necessary style overrides was done in the “custom.css” file under “styles” folder in the project. Bootstrap was used as the CSS framework for the web application. Before the deployment, a gulp task “dist-web-mobile” is run and all the HTML, CSS and JS files are packaged in a folder called “dist” which is ready for deployment.

5.5.3 Views and Styles for the Mobile Application

The views of the mobile application during the development process are added under the “mobile_views” and necessary style overrides was done in the “custom.css” file under “mobile_styles” folder in the project. Ionic was used as the CSS framework for the mobile application. Before the deployment, a gulp task “dist-web-mobile” is run and all the HTML, CSS and JS files are packaged in a folder called “www” which is ready for deployment and the folder structure in the development phase will not have any meaning in the production files.

5.5.4 Running Web Application in development environment

A gulp task runs the web application in the development environment and starts the server on address http://localhost:9000.

Once the server is started using the command “gulp”, we now have access to the running application.

Once a user has been authenticated, the application redirects the user to the projects page where all the projects are listed as shown in the picture 54. More of these views can be seen in Appendix 1.
In order to emulate the application in mobile emulators, we must make sure that the necessary tools like android studio for android and Xcode for iOS are installed. We also need to make sure that the android and iOS platforms are added in our project and it is done by using following commands in the root directory of our project.

“ionic platform add android”

“ionic platform add ios”

After the tools are ready and we have the platforms that we need in our project, we can simply run “ionic emulate android” to emulate the project in an android emulator as shown in the picture 55. The command then opens android emulator and opens the project in the emulator, which can be seen in the picture 56. More of Android and iOS views can be seen in Appendix 2,
Similarly, the same application can be simulated in iOS platform by simply running the command “ionic emulate ios”. The command then opens iOS emulator and opens the project in the emulator, which can be seen in the picture 57.

Picture 56: Projects screen in android emulator

Picture 57: Project detail screen in iOS simulator
5.5.6 End-to-end testing

Since manual testing is very time consuming and unreliable due to human errors, it was seen necessary that a test, which would test the flow of the application from start to finish is created therefore it was decided that the complete application flow should be tested using end-to-end testing technique.

In end-to-end testing, the application will be tested using web browser where all the views are tested one by one starting from the authentication to the logout. The test starts from the authentication to the system and then go through all the views expecting certain texts/values in the views, fill the form and verify that the data has been saved etc. These processes will be automated using test framework so that there is no need for a tester to go through all the views and filling form hassle.

First of all, to start writing e2e tests, a testing framework called protractor was chosen and installed globally using the following commands.

“npm install –g protractor”
“webdriver-manager update”

After that, a folder named “e2e-tests” was created in the root of our front-end project. A protractor configuration file was created and configured for the test. The guide to configure protractor can be found in protractor’s official site http://www.protractortest.org.

The test folder consists of three more folders i.e. specs, pages and helpers. The “specs” folder consists of the different test specifications. “Pages” folder consists of page object files. Page objects are interface to a page and specs use the method of these page objects to interact with the pages of application. The “helpers” folder contains the helper functions. The screenshot of the folder structure can be seen in picture 58.

Picture 58: E2e test folder structure
The test consists of spec files and page object file. The spec file consists of the different scenarios that will be tested. In the picture 59, a spec file for testing login process is shown. The first test that is run in this spec is that the application should go to login page and in the second step certain things in the login page like the existence of email input and password input is expected. If these inputs are not available, we can be sure that the login page has bug and test fails. If we have the expected input boxes, the test can proceed to next step.

```
var config = require('./config.js');
var loginPage = require('./pages/login-po.js');
var projectPage = require('./pages/project-po.js');

describe('login test', function() {
  it('should be able to go to login page', function() {
    loginPage.go('login');
  });

  it('should NOT be able to login with incorrect credentials', function() {
    loginPage.login(config.unauthorizedEmail, config.unauthorizedPassword);
  });

  it('should be able to login successfully with correct credentials', function() {
    loginPage.login(config.authorizedEmail, config.authorizedPassword);
  });

  it('should be able to redirect to Projects page', function() {
    projectPage.expectPage();
  });
});
```

Picture 59: A picture showing login test spec file

The command “protractor e2e.conf.js” form the root folder of e2e test is used to start the test. During the test, a browser that is defined in the protractor configuration file opens and the tasks like redirecting to a page, filling the form etc. is done on our behalf. The result of the test being run can be seen in the console as shown in the picture 60. If any of the tests fails, we can see a red error message indicating failure in the console immediately meaning that our test has failed because there is bug in the application or the test itself contains bug.

![E2e test specs with the result](picture60.png)
6 DEPLOYMENT

In order to deploy our web application (server and client side), a cloud Platform-as-a-Service called Heroku was chosen. Heroku is a cloud platform where applications written in Node.js, PHP, Python etc. can be deployed easily without having to worry about the Infrastructure itself. Above all, it provides a free plan to test sandbox applications with limited usages. In order to deploy the application to Heroku, one should create an account. After acquiring an account, one can create sandbox applications and deploy the code.

In order for application to run, a mongo database is also required. Similar to Heroku, a free sandbox plan from cloud service provider for MongoDB named mLab was chosen. The database connection string received from mLabs was set in the “Config Vars” of “Settings” page in Heroku.

After the processes mentioned above, we can now easily deploy the application using a command line tool provided by Heroku known as “toolbelt”. After installing the tool and adding the heroku remote in our existing git repository, the local git repository can be pushed to the heroku remote, which then builds the application and deploys it. The process of deploying an application to Heroku can be seen in picture 61.

![Picture 61: Web application deployment to Heroku cloud in progress](image-url)
Once the app is deployed to the Heroku cloud, it is not live and can be opened from the url that was assigned to the application. One also can use own domain with Heroku and there are guides on configuring them in the Heroku documentation. The picture below shows the application running live.

Picture 62: Project page of live application

When it comes to publishing mobile apps in Google Play or App Store, there are plenty of materials on the Internet, which guides one through the publishing process. In our case, the android application was sent via email to couple of people to be tested since it is not ready to be published yet. The applications made using ionic can be built using the command “ionic build android” which generates the “android-debug.apk” file under “platforms/android/build/apk” folder. This .apk file can be then installed in an actual android device and tested. The process of installing the apk file in an android device can be seen in picture 63.
Picture 63: Installing the .apk file sent via email for testing purpose, installing and running it in android device
7 DISCUSSION

The development task done proved to be very educational experience since it covered a large amount of areas related to software design and development like introduction of different modern technologies that are available currently in the field of web and cross platform hybrid mobile application development, planning and designing the application and database, the implementation of server-side technology, the implementation of client-side technology, software testing and deployment.

One of the main targets of the development task was to learn and make use of the new technologies, concepts that are available or talked about recently in the field of web and hybrid mobile application development and build an application that would have a practical use in the real world. During the process of building the application, the concepts like decoupled architecture, RESTful architecture and benefits of it became clearer as they were put into practice. Also different frameworks that are available were tried and used which gave me a firm grasp in the technologies used, which fulfils the target that was acquired before the start of the development work.

In my opinion one should definitely consider the concept of “Write once, Run anywhere” in software development project especially if the budget or the resource allocated to the project is demanding and the mobile application itself is not that demanding when it comes to performance. The concept of being able to share code and build web and hybrid cross platform mobile applications would also be suitable for the companies that are just starting out and experimenting with new ideas which gives them ability to build a working web and/or mobile applications in different platforms quickly and put it out in the market or for the web developers who are trying to put their foot in the field of mobile development without having to go through the learning curve of the technologies and programming languages that are involved with native mobile application development. If the application is too demanding in terms of performance then I would suggest that one should consider native application development simply because it will have great performance, better user experience and easier access to the built in API’s.
REFERENCES


W3C. Read 18.5.2016. HTML & CSS. https://www.w3.org/standards/webdesign/htmlcss

W3C. Read 18.5.2016. AVASCRIPT WEB APIS. https://www.w3.org/standards/webdesign/script.html


SailsJs. Read 2016. [http://sailsjs.org](http://sailsjs.org)


APPENDICES

Appendix 1. Different views of the web application
## Project

### Property

**Condominium:** As Oy Mäntyläki  
**Property Id:** C 12  
**Property manager:** Mister Tester  
**Description:** Something happened mate, i don't know.

### Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Unit Price</th>
<th>Quantity</th>
<th>Total</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kostuumitauas</td>
<td>89.99</td>
<td>1</td>
<td>89.99</td>
<td></td>
</tr>
<tr>
<td>Kostuumitauas</td>
<td>89.99</td>
<td>1</td>
<td>89.99</td>
<td></td>
</tr>
</tbody>
</table>

### Uploads

- **File 1:** Uploaded at 30.10.2016, 11:14  
- **File 2:** Uploaded at 30.10.2016, 11:14
Appendix 2. Different views of the mobile application